Mobile application development process analysis and proposal of a specific UML extension for Android applications.

Roberto Valente

DIBT - University Of Molise

Phd thesis

Supervisor: Prof. Fausto Fasano

PhD Program Coordinator: Prof.ssa Gabriella Stefania Scippa

Abstract

Every bit of technology is evolving and will continue to do so. Mobile applications, in particular, represent today one of the main evolution in technology. They are now widely used in different sectors. The mobile app ecosystem represents today one of the biggest industries all over the world. It encapsulates millions of app developers, literally billions of smartphone owners who use mobile apps daily and many companies that uses apps and make money with them. This evolution of mobile software requires more attention, more skills and a better comprehension for the development, maintenance and engineering of applications. Due to this evolution and to the growing presence of mobile application in everyday life, we though to analyse mobile context and mobile development process to study if a specific UML extension could facilitate development and maintenance of Android mobile application process. The idea was to model, extending UML standard Class Diagram objects, Android structural and UI components and provide a more specific diagram, and to support development process in all phases. We proposed an UML extension with graphical stereotype to represent information and try to increase the domain comprehension. In this work, after anlaysing mobile applications modeling ad developing issues, we studied Android, iOS and cross platform development. Considering UML standard and extension mechanism, we proposed and evaluated a Droid UML extension. We carried out a controlled experiment defining two maintenance tasks for two open source applications. We submitted surveys to 20 developers, divided in 4 groups with different tasks and UML representation and we analysed results. The obtained data told us that using a specific UML extension could improve code comprehension ad could facilitate maintenance activities. In future we think to repeat so the experiment with a larger number of developers to confirm the obtained results and we will realize a tool to automatically define Droid UML class diagrams. Further-more we think to propose a UML extension for cross platform developed applications.

Contents

List of Figures

Chapter 1

Introduction

Every bit of technology is evolving and will continue to do so. Mobile applications, in particular, represent today one of the main evolution in technology. They are now widely used in different sectors. Innovation is one of the driving factors of this evolution. It is fast and it has led to the launch of many new apps that we use everyday. Comparing the apps we use today with that used a few years ago, the difference is poles apart, just like the sun and moon differ from each other [24]. At the same time, many companies have changed the way they develop software often changing their traditional software and making it compatible with latest mobile devices. 25 years ago, IBM launched the first palmtop that included the use of applications like agenda, calendar, clock, notepad, email, etc. In 2002, RIM made the first Blackberry with integrated phone and in 2007 Apple created the first iPhone with a set of default applications. In 2008, Apple launched the first Apple Store and, in 2009, Google launched the Android Market. In 2011, the number of mobile applications exceeds one billion and the number of applications downloaded from the store exceeds 20 billions (10 billions from Android Market and 10 billions from Apple store). In 2012, more than 15 billions of applications have been downloaded from Google Play Store. [8]. The app market is seeing the revenue of more than $30 billion yearly and still growing. The year of 2014 was witnessed over 138 million app downloads in a single year, with an estimation of downloads reaching 268 million by the year 2017. The growth concerns mobile developers too. There were 19 million software developers across the world in the year 2014, and the number will grow to a whopping 25 million by 2020. Nowadays, India, Russia and China are

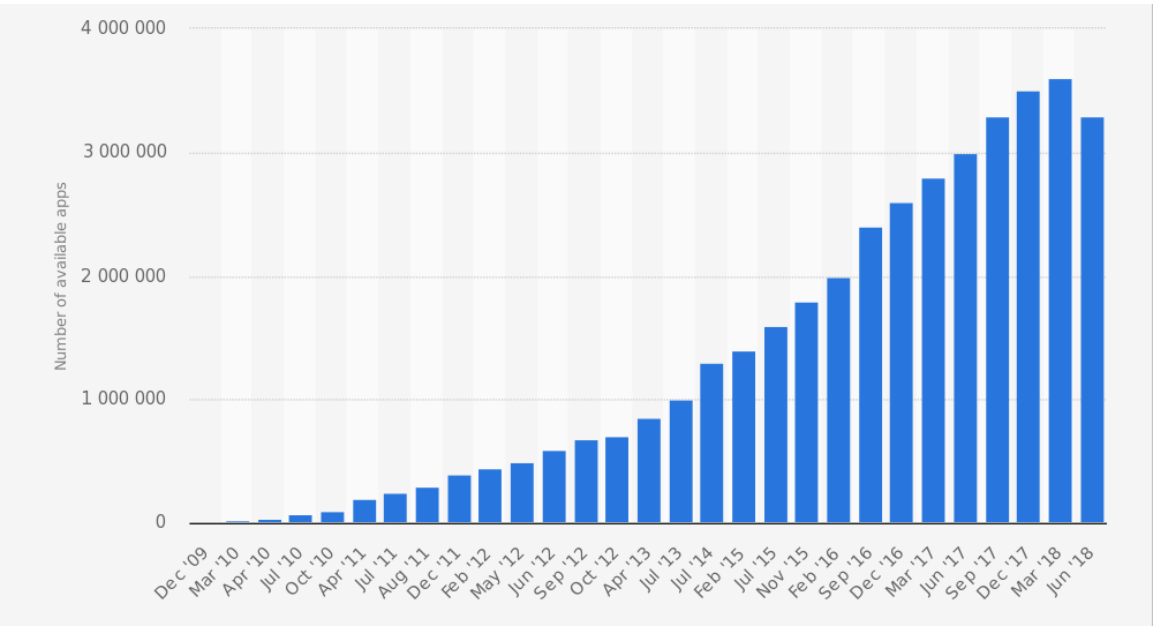

Figure 1.1: Number of applications in the Google Play Store

seeing fast growth in the number of mobile app developers compared to the countries that historically had more mobile applications developers[23].

Figure 1.1 shows the number of available applications in the Google Play Store from December 2009 to September 2018. The number of available apps in the Google Play Store was most recently placed at 2.6 million apps in March 2018, after surpassing 1 million apps in July 2013. [19].

The mobile app ecosystem represents today one of the biggest industries all over the world. It encapsulates millions of app developers, literally billions of smartphone owners who use mobile apps daily and many companies that uses apps and make money with them. In 2015, global mobile app revenues amounted to 69.7 billion U.S. dollars. In 2020, mobile apps are projected to generate 188.9 billion U.S. dollars in revenues via app stores and in-app advertising [48].

These numbers substantiate mobile-first. Mobile first requires a new approach to planning, UX design, and development that puts handheld devices at the forefront of both strategy and implementation. The digital landscape has changed, and companies have realized that consumers are now accessing more contents on their mobile devices than anywhere else. Mobile first shifts the paradigm of a Web-site user experience. Instead of users viewing desktop versions of Web sites on their mobile device with

some adjustments, users are now viewing sites that have been created specifically for their mobile device. This begs the question: how will stationary, desktop computer users view these Web sites? They will still view versions of Web sites that were developed for the desktop Web but designed with mobile in mind. This means designers should tailor site user experiences to the needs of users who are on the go and in multiple contexts. Text must be easier to read and navigate. Photos and maps should be easily accessible, and all content should adjust to display properly on the device on which a user is viewing it. The needs of users change because their context continually changes. Users have a harder time reading in-depth content on a small screen. Without a keyboard, their ability to type is hindered. Mobile devices introduce new modes of interaction such as touch and gestures. [14]. Before the iPhone, presence was equated with keyboard activity. The thinking was that it meant a person was at their computer, likely in their office and available. At the same time it's no longer just a matter of which app gets developed first, but how to maximize productivity. This evolution of mobile software requires more attention, more skills and a better comprehension for the development, maintenance and engineering of applications. Specifically, smartphones, in contrast to desktop and laptop computers, have many sensors that could increment usability. With such sensors it is possible to find position, rumor level, light, usage angle, movement and so on. Mobile applications can use these sensors simultaneously thus changing the way to design, implement and test software. In addition to these hardware innovations, mobile applications runs on small devices, in mobility and with limited battery duration. Furthermore, mobile apps are downloaded and updated quickly, they need to seamlessly interact with back-ends end servers whenever required which can be accomplished with numerous alterations and adjustments during the development phase[34].

In this thesis I decided to focus my attention on mobile application evolution in order to propose an UML extension and facilitate the code comprehension during development and maintenance phases.

Due to this evolution and to the growing presence of mobile application in everyday life, we though to analyse mobile context and mobile development process to study if a specific UML extension could facilitate development and maintenance of Android mobile application process. Why

native Android applications? We faced different issues in choosing what kind of application to consider in our research. Actually mobile applications market is mainly composed by cross platform, Android native and iOS native applications. A set of executives who are developing mobile applications in their own company or helping clients at the question "How has native mobile app development evolved?" answered that:

- Mobile apps have evolved from merely smaller versions of their desktop parents (usually just ported) to applications that are built ground up to take advantage of the rich set of sensors and systems of the devices they run on.
- There are a lot of SDKs available today that werent around two years ago - Crash analytics, user management, real-time SDK. SDK services are all native. Every major language has gone through this maturity cycle. Mobile was very fractured until iOS and Android began to dominate in the last eight years. Mobile development is more mature.
- In recent years, there have been huge debates over whether native, HTML5 or hybrid provided a superior app. Native has won the war - it looks better and provides a better experience. Movement from manual to automated testing. To stay competitive, you must move to automated testing. Deep interaction happens on the app not on the mobile site.
- People used native mobile app development tools. Now more cross-platform development but its going back to native to deliver a more truly responsive experience. Need to go native to optimize on the device. Cross-platform is not optimized for any platform [9].

We choose Android native applications as object of our research because Android applications evolved and are evolving really fast and because source code and development process information are easier to retrieve. We have open source markets, hosting and versioning repositories to analyse and a lot of applications. We thought that the Android applications development represents the most widespread development sectors of recent years, so it could be really interesting and useful for our research objectives.

The idea was to model, extending UML standard Class Diagram objects, Android structural and UI components and provide a more specific diagram, and to support development process in all phases. As studied by Ludwik Kuzniarz, Miroslaw Staron, Claes Wohlin [40] we proposed an UML extension adding graphical stereotype to represent information, to increase the domain comprehension. We started from studying mobile context and analysing Google Play Store and its applications. Then we considered open source F-Droid applications and, to retrieve information about development process, we evaluated GitHub repositories. The goal of this phase of the research was to understand the evolution of Android applications source code and the evolution of development process. Successively, we examined all the research approaches proposing UML extension for web, mobile and, in particular, Android applications. We studied also approaches to evaluate quality of proposed UML extension. In Chapter 3, the involved mobile technologies are presented, analysing differences between Android, iOS, and cross platform development of applications and, in chapter 4, after describing UML standard, a Droid UML extension is proposed. In chapter 5, a controlled experiment is designed and carried out in order to evaluate utility of Droid UML extension discussing results of experiment. Finally, the thesis is concluded with final considerations and proposal for future works.

Chapter 2

The mobile context

In this chapter we report the studies done during the research of the mobile applications modeling and development context. We analysed so different aspects. The first one concerns Android applications source code and development process analysis. We tried to understand if there were an evolution in source code and in development process of mobile applications. So we evaluated the number of apps and some source code metrics of a set of Google Play Store applications. Then we focus our attention on F-Droid repositories to understand if data carried out from decompiled applications from Google Play Store was similar to data obtained from source code applications found on F-Droid. The last study was carried out to better understand the development process of an Android project. Indeed we analysed GitHub repositories and in particular data related to authors, commits and releases during the time. Furthermore, we focused attention on the state of art of mobile application modeling techniques, the attempts done in extending UML to mobile application context and the experiment done to evaluate quality of a formal representation in modeling software.

2.1 Android applications analysis

In order to give a real validation to the effective evolution of applications not only in terms of number but also in terms of functionalities and nature, we decided to analyse code metrics and GitHub repositories of Android applications to understand how source code and development process has evolved during the years.

2.1.1 Google play store analysis

The first analysis was done on Google Play Store applications. We decided to calculate some code metrics to analyse app complexity and source code evolution. To achieve that, we realized a tool written in PHP code to retrieve link to apk files from Google Play Store web site and write them into a txt file. The tool stores in a database table "online_apps" the following information: package name, last update data and downloaded flag, that contains one if package has already been downloaded and zero otherwise. The main PHP file uses an HTML DOM library to parse apps page of Google Play Store.

```
$html = new simple_html_dom();
$html->load_file("https://play.google.com/store/apps");
```

For each ¡¡href¿¿ value in anchor tags of Google Play Store web site the tool gets the link related to category of applications and explode them recursively. The tool then writes in database the package name and link.

```
function writetodb($link, $package){
  $sql="Insert into online_apps (package) values ($package)";
  if(check_if_exists($link, $package)){
  $res = $link->query($sql);
       if($res) return true;
       else return false;
  }else return false;
 }
```

Moreover, during a second phase, the tool writes or updates the data relative to the packages previously found on The Google Play Store. In fact we implemented a second part of the tool that reads from the database all the packages already written before and, for each of them, writes last updated data relative to single application. The date is needed to check

if package has already been downloaded and if it has been updated. In that case the tool sets the downloaded flag to 1 and store package name "updates" table.

```
$sql="select * from online_apps
        where download='0' order by package";
$res = $link->query($sql);
if($res->num_rows >0){
    while($row = $res->fetch_array(MYSQLI_ASSOC)){
        $title=$row['package'];
            $date1=$row['data'];
            $date2=getLastUpdateFromStore($title);
            if($date2>$date1){
              updateDB2($link, "online_apps", $title, $date1);
              writeupdate($link, "updates", $title, $date1);
            }
      }
}
```

The tool now creates a download lists writing it in a txt file. The file is given in input at a java tool that downloads new or updated packages. In this way, we have all the information about applications updates and we can download different versions of the same applications to eventually evaluate metrics of the different versions for the same applications. The Java tool mentioned before, is a Java Google Play Crawler available on GitHub [11] realized by Ali Demiroz. The tool, mainly composed of a jar file, takes in input the downloads list, and downloads apk files in a specific location of the file system. Once downloaded apk files, we launched a script that renames all the files into zip and then unzip them. We obtained so jars files of all applications preciously downloaded. We need so a jar decompiler to obtain so source code of applications.

Now started the metrics calculation phase. Using SonarJava project [21] we calculated so metrics from about 5000 projects with decompiled source code. We calculated 4 code metrics: Number of Classes, Blank Lines, Comments and Lines of Code of applications decompiled code from year 2009 to year 2015. Table 2.1 contains medium values per year.

Year	N.of classes	Blank Lines	Comments	Lines of Code
2009	19,25	227,38	307,51	2633,59
2010	41,67	523,84	827,11	5960,53
2011	121,61	1322,72	2144,28	15135,96
2012	224,46	2361,33	4487,45	25976,75
2013	412,04	55115,504	7713,83	56857,61
2014	1411,01	16973,799	27075,685	166051,1
2015	2167,25	24391,155	36467,42	238373,1

Table 2.1: Medium values of decompiled code metrics from 2009 to 2015

Fig 2.1 shows that metrics grows during the years. Even if the Android applications grows in number and size, we cannot understand exactly what happens in code complexity for two main reasons. First because we considered a different number and different type of applications during the different years so we cannot understand if this influences the obtained data. Then because we need to compare decompiled to source code one to understand if they are different in terms of metrics and if decompilation phase influences them.

We decided so to consider applications with their source code and decompiled code retrieving them from F-Droid store that contains both.

2.1.2 FDroid analysis

At the end of Google Play Stora application evaluation, we noticed that metrics trend is not directly related to size and complexity of applications. This maybe depends by Android SDK evolution in terms of libraries and APIs and by number and type of applications randomly downloaded from Google Play store.

We thought so to analyse source code of applications downloaded from F-Droid store to have a more clear picture of the Android application source code and to understand if metrics calculated on code of decompiled apps defer from metrics calculated on source code ones.

First analysis was done on five F-Droid applications in their last three different versions. We calculated metrics on source code and on decompiled code without SDK and API libraries.

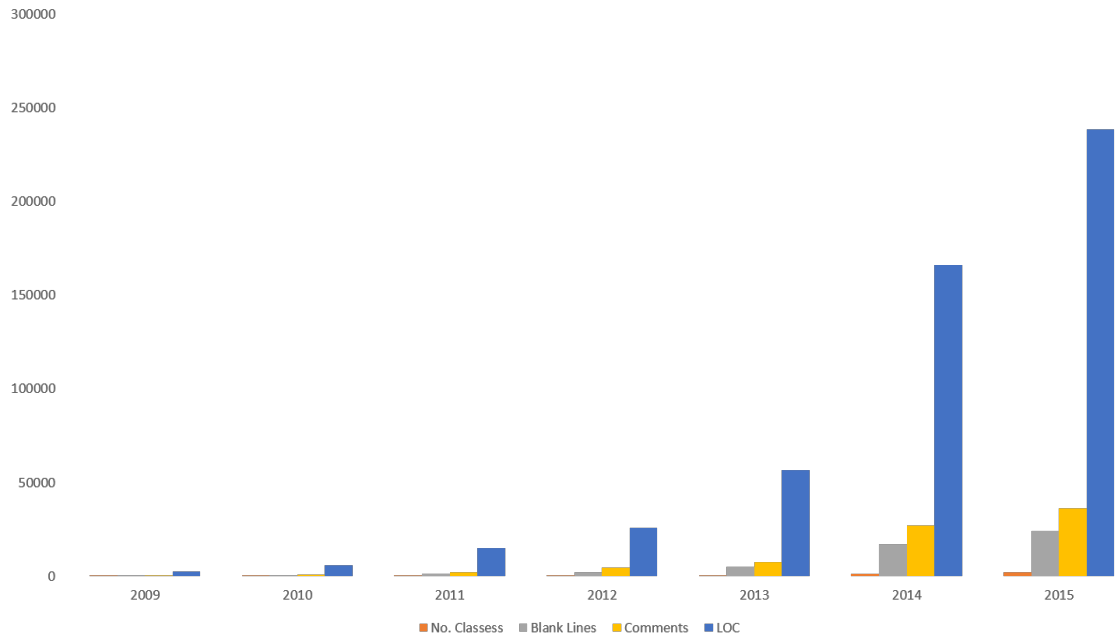

Figure 2.1: Metrics trend from 2009 to 2015

Application Name	LOC	eLOC	WMC
Lightning	19581	14045	2778
DNSSetter	335	156	19
BRouter	3160	1989	336
Alarm Clock	5343	3452	842
AutoAnswer	155	83	17

Table 2.2: F-Droid source code metrics

We used a java tool proposed by Palomba F. et al [44] called Code Smell Detector to retrieve 3 source code metrics: Lines Of Code (LOC), effective Lines Of Code(eLOC) and Weighted Method per Class (WMC) where weighted methods for class measures the complexity of an individual class [31].

Table 2.2 and 2.3 contains the results of the analysis.

As we can observe from tables 2.2 and 2.3 we didn't find more differences between source and decompiled code.

So we decided to proceed with a second analysis to understand if applications code complexity changes during the time. We evaluated LOC, eLOC and WMC metrics for three different versions of the same applications. The data showed in tables 2.4, 2.5 and 2.6 tell us that there isn't a real

Application Name	LOC	eLOC	WMC
Lightning	18973	13998	2719
DNSSetter	306	216	19
BRouter	3141	2503	332
Alarm Clock	5377	3972	618
AutoAnswer	155	104	15

Table 2.3: F-Droid decompiled code metrics

Application name	version 1 LOC	version 2 LOC	version 3 LOC
Lightning	19581	19954	19979
DNSSetter	335	345	347
BRouter	3160	3462	3837
Alarm Clock	5343	5404	3283
AutoAnswer	155	390	567

Table 2.4: F-Droid Lines Of Code for three versions

growth of source code in terms of code metrics. The metrics evaluated in different versions, except application Alarm Clock that probably has been optimized, have similar metrics values.

We can affirm so that there isn't a difference between versions and between source code and decompiled code of applications. This could depend from the fact that we considered a small set of applications so we decided to analyse source code of entire F-Droid market to understand evolution of applications.

We realized so a new Html crawler to retrieve all the links to source code projects of F-Droid store and write them in a txt file. Then we cloned all the applications whose source code is hosted on GitHub. We used Code Smell Detector to calculate metrics again. Once obtained snapshot of FDroid applications we divided information by different years from 2011 to 2016 in order to evaluate code evolution.

Application name	version 1 eLOC	version 2 eLOC	version 3 eLOC
Lightning	14045	14230	14253
DNSSetter	156	163	165
BRouter	1989	2114	2367
Alarm Clock	3452	3477	2011
AutoAnswer	83	256	276

Table 2.5: F-Droid effective Lines Of Code for three versions

Application name	version 1 WMC	version 2 WMC	version 3 WMC
Lightning	2778	2881	2888
DNSSetter	19	19	19
BRouter	336	358	409
Alarm Clock	842	849	377
AutoAnswer	17	49	53

Table 2.6: F-Droid Weighted Methods per Class for three versions

Year	LOC	eLOC	WMC
2011	15152,41	10567,04	1956,111
2012	7828,245	5098,255	927,3804
2013	10898,04	6999,227	1305,752
2014	18462,24	10953,13	2185,005
2015	15550,48	9338,396	1829,087
2016	16233,46	9777,262	1944,695

Table 2.7: F-Droid source code metrics by year

Table 2.7 contains average metrics (LOC, eLOC, WMC) of the entire F-Droid store at 2016. As we can see in Figure 2.2 code metrics does not grow during years so we can observe that application complexity is not representable by evaluating code metrics during the time.

At this time of the research, the idea was to move our attention to development process of applications in order to verify a growth in terms of participation to an application development and maintenance phases. We decided so to retrieve information on GitHub repositories to understand the evolution of the number of commits, number of releases and number of authors.

2.1.3 GitHub analysis

The GitHub analysis had the objective to better evaluate the development process of an Android application during the years and to asses the number of developers that participate to the application development and maintenance so we could propose and evaluate a specific Droid UML extension in order to facilitate developers code comprehension and maintenance phases.

To do that we realized another PHP tool that retrieves F-Droid source code links and write them in a txt file.

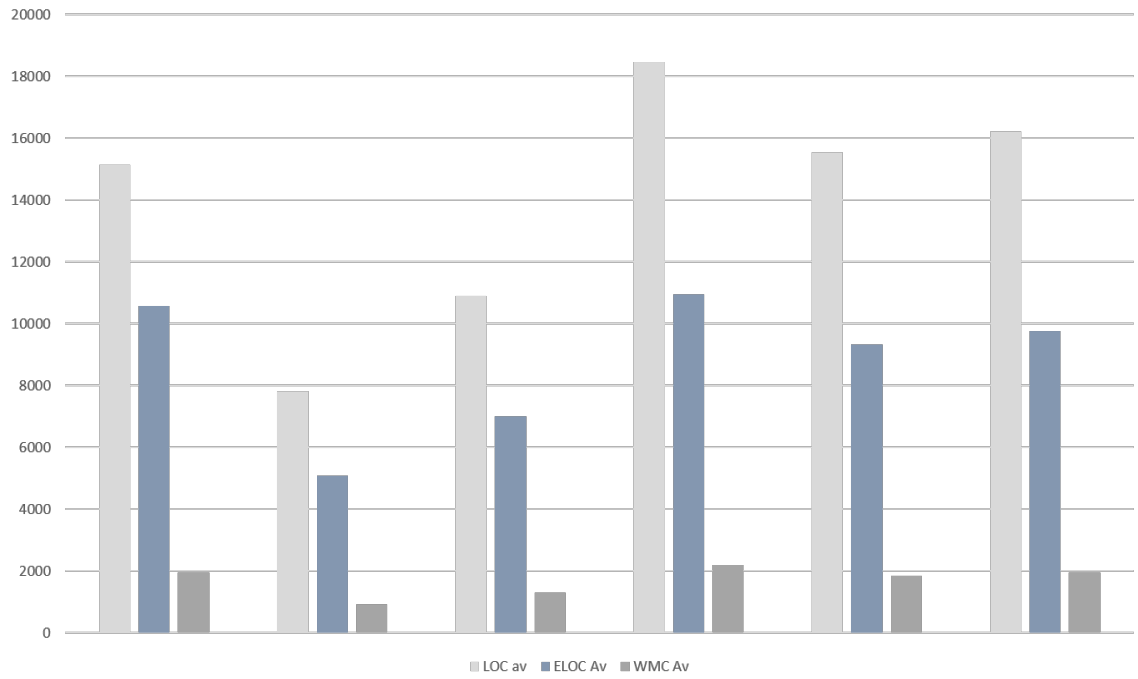

Figure 2.2: F-Droid metrics average from 2011 to 2016

```
$htmlfile = file_get_html('...');
crawlPage($htmlfile);
foreach ($links as $link){
        crawlPage(file_get_html("...".$link));
}
```

For each page of the F-Droid wiki list (contained in variable $links) the tools use the crawlPage() function that retrieves links and use getGitRepo() function to write them in a file called list.txt.

```
function crawlPage($html){
  foreach($html->find('a') as $e){
   if(strpos($e, "/wiki/page/")!=false){
      if(strpos($e->href, ":")==false){
         if(strpos($e->href, "Main_Page")==false){
                  echo ($e->href);
                  getGitRepo(file_get_html($e->href));
               }
         }
      }
   }
}
```

```
function getGitRepo($page){
   $myfile = fopen("list.txt", "a") or die("Error");
      foreach($page->find('p') as $par){
```

```
            if(strpos($par, "Source")!=false){
                fwrite($myfile, $par->first_child()->href, PHP_EOL);
             }
        }
    fclose($myfile);
}
```

At the end of the process we obtained a list of GitHub repositories to work with. We needed so to clone repositories and retrieve information about commits, releases and author and analyse them.

We used GitWrapper PHP library [10] to retrieve all the commits for different time intervals.

```
$result=getCommits($client,$repo_name,$params, "",$link,$k);
while($result){
    $callCount++;
    $result=getLastCommitByTimeInterval($client, $repo_name,
    $params."&sha=".$result, $result, $link, $k);
}
$i++;
```

The function getCommits() writes in database the commit code, the author, the date and the commit message. Figure 4.1 contains the medium number of commits from 2009 to 2017. Value of year 2017 is smaller because the commits are relative to the first five months of the year. As we can observe from Figure 2.3, the average number of commits is quite similar during the year except for year 2009 maybe because there was a small number of repositories with an high number of commits.

As we can observe from Figure 2.3, there isn't a real relationship between time and number of commits. We observed same data trends for number of authors and releases.

We thought so to analyse data considering smaller time intervals. So we calculated commits and authors relative to four months periods as reported in Figure 2.4 and 2.5.

As we can observe in figures 2.4 and 2.5 we obtained the same results of larger time intervals. Number of commits and authors does not grows during the years. We obtained same trend of releases data.

So we moved the attention to the aspects related to the participation of developers the the projects. In particular we thought to evaluate the number of contributors that participate to the applications development. We

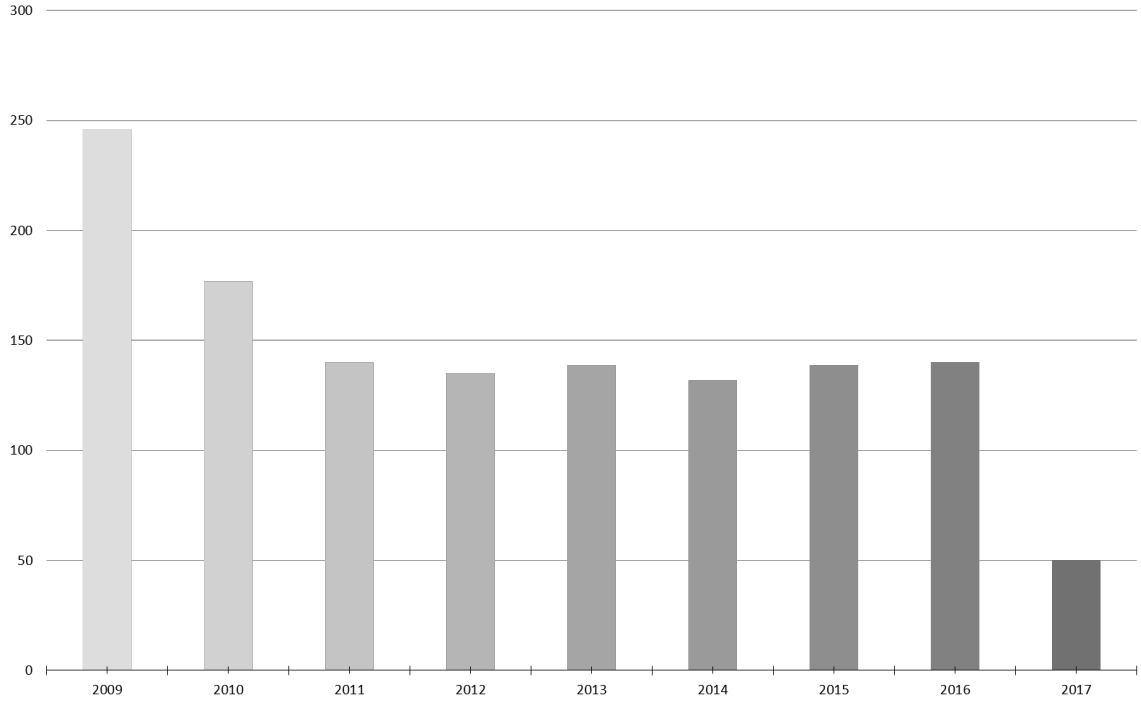

Figure 2.3: Commits average number

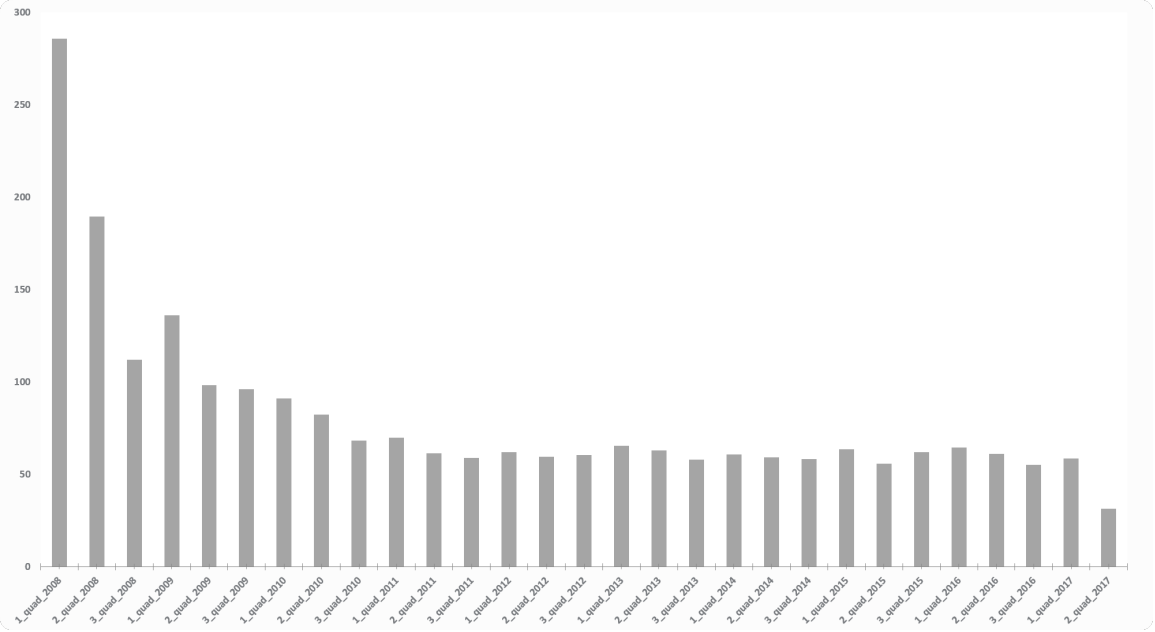

Figure 2.4: Commits average number per four months period

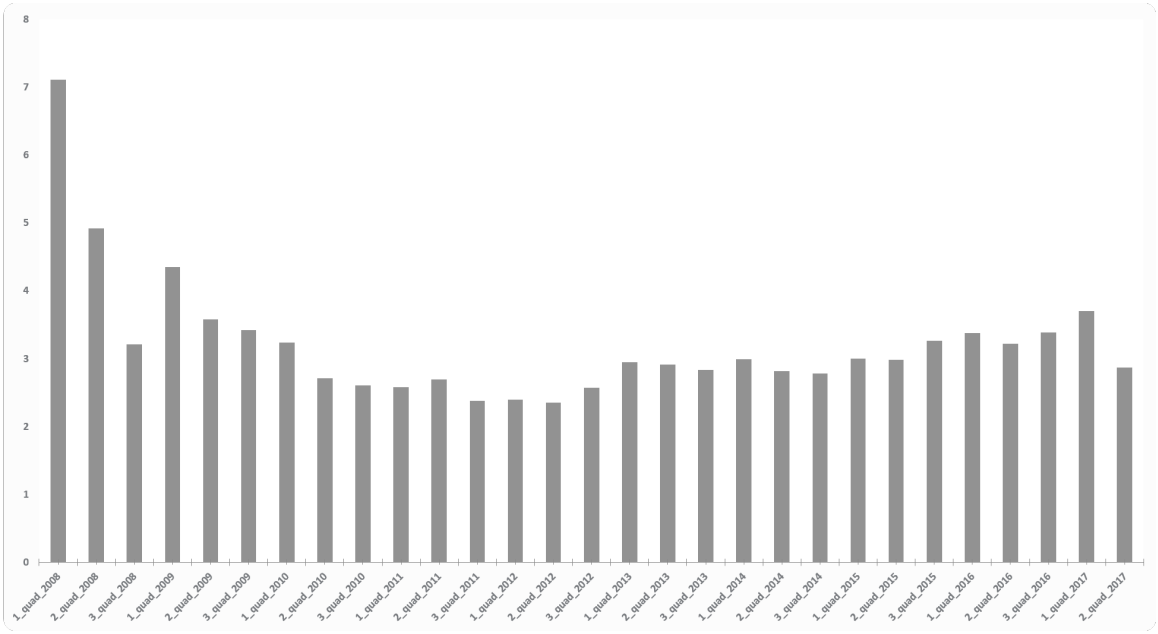

Figure 2.5: Authors average number per four months period

can assume that if there is a significant number of projects with relevant number of contributors it could be useful propose the extension with the scope to facilitate development and maintenance of applications.

To achieve this result, we decide to design an algorithm that, starting from first month of life of an android project repository on GitHub, browse all the commits of the same repository and memorize all the new comers (authors) found in a database.

```
function totalNewcomers(repo) {
  newcomers = 0
  authors = {}
  foreach commit in commits[repo.start,repo.start+1month]{
        if commit.author not in authors
              authors = authors + commit.author
        }
  foreach commit in commits[repo.start+30days,repo.end]{
        if commit.author not in authors
              newcomers++
        }
return newcomers
}
```

The data obtained from the execution of the algorithm are reported in table 2.8. The second column contains the number of repositories with a number of newcomers higher of value contained in first column.

new comers	repositories
>10	200
>15	127
>20	96
>50	22
>100	11

Table 2.8: number of repositories with relevant number of new comers

There is a significant number of repositories, about 500 on 1400, (about 35% of total number of repositories analysed), with a relevant number of contributors (from more than 10 to more than 100). We decided so to define the extension and empirically evaluate it with a controlled experiment.

2.2 Modeling mobile application: state of art

During our context study, we found several research works concerning mobile application development and modeling. The main part of the researches in past years, concerned specific modeling in support of automatic code generation for applications. At the same time, different approaches were realized in order to propose specific UML diagrams for Android and multi-platform applications. In this section we present related works relative Mobile modeling approaches focusing our attention on three main areas:

1. Model driven design in support of development applications.
2. UML general extension approaches.
3. UML extensions for mobile applications.
4. Empirical experiments to evaluate notations utility.

2.2.1 Model driven design in support of development of applications

Different approaches concerns model driven design in support of development of applications. They are focused on code generation in support of multi platform development. Parada and De Brisolara [45] in 2012 proposed a model driven approach for Android applications development. The main objective of this approach was to reduce the gap between the problem domain and the software implementation through the use of technologies that support systematic model transformations. The approach included UML based modeling and automatic code generation to facilitate and accelerate the development of Android applications. Their modeling was based on UML, using class diagram to describe the application structural view, and sequence diagrams to represent the behavioral view. This approach was useful to better understand what kind of components consider in our approaches but misses of a quality evaluation of the extension proposed and of the code generated.

Another interesting approach was proposed by Sabraoiu et al [46] in 2012. They proposed an approach based on MDA, to generate GUI for mobile

applications on smartphones. The adopted approach consists of three main steps (i) analyzing and modeling the GUI under UML; (ii) transforming the obtained diagrams to a simplified XMI schema using JDOM API; and (iii) generating the GUI based on MDA. Their method has the advantages to generate automatically GUI for several platforms, and gives a graphical way for designing in UML. This approach does not propose a real UML extension for mobile applications but examine mobile context defining a metamodel for automatic code generator.

M. Usman et al. [49] in 2014 proposed a model-driven approach to generate mobile applications code for multiple platforms. They proposed a modeling methodology using real use-case for requirement gathering, class diagrams for structural modeling and state machine for behavioral modeling. To generate mobile application automatically, they develop a tool named Mobile Application Generator (MAG) that takes the developed UML models as input and generates application for the specified target mobile platforms. They also proposed an UML profile for modeling domain specific concepts. The approach presented concerns multiple platform applications and focused on code generation phase. It presents a code evaluation too, but misses of an evaluation about effective utility in programming applications.

G. Botturi et al [29], likewise, proposed a model driven approach based on code generation so that no additional library or process is needed on the smartphone to support different platforms. They used UML2 profile to represent the elements of application independently of the target platform. This approaches, like the others, is oriented to code generation and take into account UI elements. It misses os structural components.

Another interesting proposal comes out from F. Freitas et al. [35]. They asserted that Model-driven Engineering (MDE) has emerged as a concrete alternative to automatically generate Android applications and proposed JustModeling, an MDE approach formed by JBModel, a graphical modeling tool with which the user models the application business classes using the UML class diagram and that provides a set of model transformations to generate code for the JustBusiness framework, which automatically generates all necessary resources of the mobile application. This allows

developers to work on a higher level of abstraction, focusing on the application design rather than implementation issues. The approach is mainly oriented to Android Business applications and misses of structural component.

2.2.2 UML extensions approaches

All these approaches concerns modeling and automated generating code for Android or cross platform applications. Since our interest is mainly focused on application modeling to improve code comprehension during development and maintenance phases we analysed approaches that proposed UML extensions.

First important approach to UML extension was presented by Conallen [32] in 2000. He proposed an important extension of UML language to face the modeling of web applications. His work aim at proposing a workable solution for releasing web applications. The proposal privileges client-server interactions and underestimate the logical vs. physical design of both information and navigation structures. It defines stereotypes, tagged values, and OCL constraints to model web pages and hyperlinks, forms, frames, and client-server components at a concrete level. Conallen adapts also all classical phases of software development to web architectures, and tailors almost all UML diagrams to render web related concepts.

Different proposal came out from the Conallen one. Baumeister et al. [27] proposed an UML extensions to model hypermedia applications running on the Internet. In their paper they proposed such an extension for modeling the navigation and the user interfaces of hypermedia systems. Similar to other design methods for hypermedia systems they viewed the design of hypermedia systems as consisting of three models: the conceptual, navigational and presentational model.

Koch, Nora, et al. [39] presented another approach of UML Profile for Web applications. It was a UML extension based on the general extension mechanism provided by the UML that defines specific stereotypes to model aspects related to navigation and presentation of Web applications. This profile is part of a methodology for the analysis and design of Web applications. This methodology performs separate steps for conceptual,

navigational and presentational modeling in a similar way as it is proposed by other methods for hypermedia or Web design. The novelty of this approach consists in the modeling techniques and notation used, that are entirely based on the Unified Modeling Language.

An interesting approach comes from M.Nassar [42] that proposed an extension of UML called VUML (View based Unified Modeling Language). VUML was based on the concept of multi views component whose goal is to store and deliver information according to users' viewpoints. This approach allows for dynamic change of viewpoints and offers mechanisms to describe views dependencies.

Other researches concerns different application sectors. Once of them was realized on Secure Systems Development. Jrjens, Jan.[37] proposed a UML extension to support using UML for secure system development (UMLsec). With their proposal, they encapsulated knowledge on prudent security engineering and thereby made it available to developers witch may not be specialized in security. Jrjens defined new stereotypes and tags and enabled so developers with background in security to make use of security engineering knowledge encapsulated in a widely used design notations.

All these approaches led us to elaborate on mobile applications context. If UML extensions mechanism is useful in web, security and other sectors, why not in development of application form mobile devices?

2.2.3 UML extensions for mobile applications

We found in literature different approaches that concern UML extensions for mobile applications context.

The proposal of M. Ko et al. [38] seemed to be the most interesting approach about extending UML for android applications. They used UML extension mechanism and proposed a meta-model for developing an application on the Android platform using this mechanism. They identified main Android applications features and provided a more specific UML class diagram containing specific domain definitions. Anyway the research misses of experiments to understand effective utility of meta model proposed.

Table 2.9: Related work comparison

Proposal	Platform	UML diagram	Modeled components	Case Study	Experiment	Graphycal Stereotype
Parada et al	Android	Class, Sequence	Interface, hardware	YES	NO	NO
Sabraoiu et al	Android	None	Structural	YES	NO	NO
M. Usman et al	Multiplatform	Class	Business Logic	YES	NO	NO
G. Botturi et al	Multiplatform	Class	Interface	YES	NO	NO
F.Freitas et al	Android	Class	Business Logic	YES	NO	NO
M. Ko et al.	Android	Class	Structural, Dynamic	YES	NO	NO
Bup-Ki Min et al	Windows	Class	Hardware, software	YES	NO	NO
Perego and Pezzetti	Multiplatform	Class	Hardware, lifecicle, interface	YES	YES code quality	NO

Bup-Ki Min et al. [41] proposed an UML metamodel for application based on Windows Phone. They suggested an extended metamodel for modeling applications based on Windows Phone 7 using the UML extension mechanism. To do this, they analyzed Windows Phone 7s features and classified them with respect to software elements and hardware resources. They extended software hardware and fundamental functions using stereotype mechanism provided by UML.

Another interesting proposal comes from Perego and Pezzetti in 2013 [36]. They analysed mobile application development context and proposed an abstract UML meta-model whose instances represent high level models for mobile applications. Then they created a more concrete version of meta-model aimed to define a more detailed model of the applications. They also presented a tool to generate Android and iOS source code starting from meta-model defined. The evaluations done concerned only quality of code generated and not effective utility of the proposal.

Before analysing evaluation experiment approaches we present a summary table about related works and their differences with our proposed approach.

Table 2.9 shows differences between approaches analysed and our works. The table contains the approaches related to the Model Driven design in support of development application and the approaches more strictly related to mobile context UML extension proposal. The first aspect evidenced is about the execution of the controlled experiment to evaluate extension proposed. No one did the evaluation except Perego and Pezzetti works that tried to analyse code generated quality. Further more no one approach consider the utility of graphical stereotype as proposed by Conallen [32] and no one thought to evaluate developers utilization of

such approaches for code maintenance processes. Let analyse now empirical experiment to understand how to design our experimentation.

2.2.4 Empirical experiments to evaluate notations utility

During the research done we analysed different empirical studies for a correct evaluation of the utility of the proposed Droid-UML extension.

The first controlled experiment was conducted by W. J. Dzidek et al [33]. They investigated the costs of maintaining and the benefits of using UML documentation during the maintenance and evolution of a real, non-trivial system, using professional developers as subjects, working with a state-of-the-art UML tool during an extended period of time. Kuzinarz L. et al.

Kuzniarz et al, [40] analysed the use of stereotype to improve understanding of UML models. The paper elaborates on this role of stereotypes from the perspective of UML, clarifies the role and describes a controlled experiment aimed at evaluation of the role - in the context of model understanding. The results of the experiment support the claim that stereotypes with graphical icons for their representation play a significant role in comprehension of models and show the size of the improvement.

in 2005 Briand et al, [30] proposed a controlled experiment that investigate the impact of using OCL on three software engineering activities using UML analysis models: detection of model defects through inspections, comprehension of the system logic and functionality, and impact analysis of changes. In order to investigate the impact of OCL in UML development, they designed, performed, and replicated a controlled experiment. It involved fourth year software/computer engineering students who received substantial training in UML and OCL. We investigated the impact of using OCL on three important software engineering activities: 1) understanding the functionality and internal logic of modeled systems, 2) performing a change impact analysis based on UML models, and 3) detecting defects through model inspections.

Safdar A. et al, [47] realized a controlled experiment for comparison of MDSE modeling tools. They measured the productivity in terms of mod-

eling effort required to correctly complete a task, learnability, time and number of clicks required, and memory load required for the software engineer to complete a task.

Another approach proposed by G.Bavota et al. [28]. They formalized an empirical study aiming at comparing the support provided by ER and UML class diagrams during maintenance of data models. The experiment done was aimed to understand the effectiveness of UML class diagrams and ER diagrams for the purpose of understanding which provides better support with respect to the comprehension and modification of data models.

Chapter 3

Mobile technologies and applications development

In this chapter we report the main mobile operating systems and the application development techniques for each of them. In particular we focus our attention on Android and iOS operative systems, specifying their evolution, their main components and how to develop applications. We describe cross platform development too, with particular attention to different development approaches and different tools used to develop applications.

3.1 The mobile operating systems

A mobile operating system (or mobile OS) is an operating system for phones, tablets, smartwatches, or other mobile devices.

Mobile operating systems combine features of a personal computer operating system with other features useful for mobile or handheld use. Some of this features are considered essential in modern mobile systems: wireless inbuilt modem and SIM tray for telephony and data connection, touchscreen, cellular, Bluetooth, Wi-Fi Protected Access, Wi-Fi, Global Positioning System (GPS) mobile navigation, video- and single-frame picture cameras, speech recognition, voice recorder, music player, near field communication, and infrared blaster [15].

Main mobile OS are:

1. Android OS (Google Inc.): The Android mobile operating system is Google's open and free software stack that includes an operating system, middleware and also key applications for use on mobile devices, including smartphones.
2. Bada (Samsung Electronics): Bada is a proprietary Samsung mobile OS that was first launched in 2010. The Samsung Wave was the first smartphone to use this mobile OS. Bada provides mobile features such as multipoint-touch, 3D graphics and of course, application downloads and installation.
3. BlackBerry OS (Research In Motion): The BlackBerry OS is a proprietary mobile operating system developed by Research In Motion for use on the companys popular BlackBerry handheld devices. The BlackBerry platform is popular with corporate users as it offers synchronization with Microsoft Exchange, Lotus Domino, Novell GroupWise email and other business software, when used with the BlackBerry Enterprise Server.
4. iPhone OS / iOS (Apple): Apple's iPhone OS was originally developed for use on its iPhone devices. Now, the mobile operating system is referred to as iOS and is supported on a number of Apple devices including the iPhone, iPad, iPad 2 and iPod Touch. The iOS

mobile operating system is available only on Apple's own manufactured devices as the company does not license the OS for third-party hardware. Apple iOS is derived from Apple's Mac OS X operating system.

5. MeeGo OS (Nokia and Intel): A joint open source mobile operating system which is the result of merging two products based on open source technologies: Maemo (Nokia) and Moblin (Intel). MeeGo is a mobile OS designed to work on a number of devices including smartphones, netbooks, tablets, in-vehicle information systems and various devices using Intel Atom and ARMv7 architectures.

6. Palm OS (Garnet OS): The Palm OS is a proprietary mobile operating system (PDA operating system) that was originally released in 1996 on the Pilot 1000 handheld. Newer versions of the Palm OS have added support for expansion ports, new processors, external memory cards, improved security and support for ARM processors and smartphones. Palm OS 5 was extended to provide support for a broad range of screen resolutions, wireless connections and enhanced multimedia capabilities and is called Garnet OS.

7. Symbian OS (Nokia): Symbian is a mobile operating system (OS) targeted at mobile phones that offers a high-level of integration with communication and personal information management (PIM) functionality. Symbian OS combines middleware with wireless communications through an integrated mailbox and the integration of Java and PIM functionality (agenda and contacts). Nokia has made the Symbian platform available under an alternative, open and direct model, to work with some OEMs and the small community of platform development collaborators. Nokia does not maintain Symbian as an open source development project.

8. webOS (Palm/HP): WebOS is a mobile operating system that runs on the Linux kernel. WebOS was initially developed by Palm as the successor to its Palm OS mobile operating system. It is a proprietary Mobile OS which was eventually acquired by HP and now referred to as webOS (lower-case w) in HP literature. HP uses webOS in a number of devices including several smartphones and HP TouchPads. HP has pushed its webOS into the enterprise mobile market

by focusing on improving security features and management with the release of webOS 3.x. HP has also announced plans for a version of webOS to run within the Microsoft Windows operating system and to be installed on all HP desktop and notebook computers in 2012.

9. Windows Mobile (Windows Phone): Windows Mobile is Microsoft's mobile operating system used in smartphones and mobile devices with or without touchscreens. The Mobile OS is based on the Windows CE 5.2 kernel. In 2010 Microsoft announced a new smartphone platform called Windows Phone [17].

With the exception of Android (developed by Google), mobile operating systems are developed by different mobile phone manufacturers, including Nokia (Symbian, MeeGo, Maemo); Apple (Apple iOS); Research In Motion (RIM) (BlackBerry OS); Microsoft (Windows Mobile, Windows Phone) and Samsung (Palm WebOS and bada). Android, LiMo, Maemo, Openmoko and Qt Extended (Qtopia) are based on the Linux open-source OS [16].

3.1.1 Android Operating System

Android is a mobile operating system developed by Google. It is based on a modified version of the Linux kernel and other open source software, and is designed primarily for touchscreen mobile devices such as smartphones and tablets. Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008. The operating system has since gone through multiple major releases, with the current version being 9 "Pie", released in August 2018.

3.1.1.1 Android OS evolution

Android made its official public debut in 2008 with Android 1.0 a release so ancient it didn't even have a cute codename.

In 2009's Android released version 1.5: Cupcake. The tradition of Android version names was born. Cupcake introduced numerous refinements to the Android interface, including the first on-screen keyboard, something necessary as phones moved away from the once-ubiquitous physical keyboard

model. This version also brought about the framework for third-party app widgets, which would quickly turn into one of Android's most distinguishing elements, and it provided the platform's first-ever option for video recording.

Android 1.6, Donut, rolled into the world in the fall of 2009. Donut filled in some important holes in Android's center, including the ability for the OS to operate on a variety of different screen sizes and resolutions, a factor that should be critical in the years to come.

Android 2.0 Eclair, emerged just six weeks after Donut and its "point-one" update, also called Eclair, came out a couple months later. Eclair was the first Android release to enter mainstream consciousness thanks to the original Motorola Droid phone and the massive Verizon-led marketing campaign surrounding it.

Just four months after Android 2.1 arrived, Google served up Android 2.2, Froyo, which revolved largely around under-the-hood performance improvements. Froyo did deliver some important front-facing features, though, including the addition of the now-standard dock at the bottom of the home screen as well as the first incarnation of Voice Actions, which allowed you to perform basic functions like getting directions and making notes by tapping an icon and then speaking a command.

Android's first true visual identity started coming into focus with 2010's Gingerbread release. Bright green had long been the color of Android's robot mascot, and with Gingerbread, it became an integral part of the operating system's appearance. Black and green seeped all over the UI as Android started its slow march toward distinctive design.

Android 3.0 (Honeycomb, 2011) came into the world as a tablet-only release to accompany the launch of the Motorola Xoom, and through the subsequent 3.1 and 3.2 updates, it remained a tablet-exclusive (and closed-source) entity. Under the guidance of newly arrived design chief Matias Duarte, Honeycomb introduced a dramatically reimagined UI for Android. It had a space-like "holographic" design that traded the platform's trademark green for blue and placed an emphasis on making the most of a tablet's screen space.

Ice Cream Sandwich, also released in 2011, served as the platform's official entry into the era of modern design. The release refined the visual concepts

introduced with Honeycomb and reunited tablets and phones with a single, unified UI vision.

In 2012 and 2013 three version of Android Jelly Bean was released. They took ICS's fresh foundation and made meaningful strides in fine-tuning and building upon it. The releases added plenty of poise and polish into the operating system and went a long way in making Android more inviting for the average user. Visuals aside, Jelly Bean brought about our first taste of Google Now, the spectacular predictive-intelligence utility that's sadly since devolved into a glorified news feed. It gave us expandable and interactive notifications, an expanded voice search system and a more advanced system for displaying search results in general, with a focus on card-based results that attempted to answer questions directly.Multiuser support also came into play, albeit on tablets only at this point, and an early version of Android's Quick Settings panel made its first appearance.

In 2013's KitKat release marked the end of Android's dark era, as the blacks of Gingerbread and the blues of Honeycomb finally made their way out of the operating system. Lighter backgrounds and more neutral highlights took their places, with a transparent status bar and white icons giving the OS a more contemporary appearance. Android 4.4 also saw the first version of "OK, Google" support but in KitKat, the hands-free activation prompt worked only when your screen was already on and you were either at your home screen or inside the Google app.

Google essentially reinvented Android with its Android 5.0 Lollipop release in the fall of 2014. Lollipop launched the Material Design standard, still present today, which brought a whole new look that extended across all of Android, its apps and even other Google products. The card-based concept that had been scattered throughout Android became a core UI pattern, one that would guide the appearance of everything from notifications, which now showed up on the lock screen for at-a-glance access, to the Recent Apps list, which took on an unabashedly card-based appearance. Lollipop introduced a slew of new features into Android, including truly hands-free voice control via the "OK, Google" command, support for multiple users on phones and a priority mode for better notification management. It changed so much, unfortunately, that it also introduced

a bunch of troubling bugs, many of which wouldn't be fully ironed out until the following year's 5.1 release.

In the grand scheme of things, 2015's Marshmallow (Android 6.0) was a fairly minor Android release, one that seemed more like a 0.1-level update than anything deserving of a full number bump. But it started the trend of Google releasing one major Android version per year and that version always receiving its own whole number. Most important element of this release was a screen-search feature called Now On Tap, something that had tons of potential that wasn't fully tapped. Google never quite perfected the system and ended up quietly retiring its brand and moving it out of the forefront the following year. Android 6.0 did introduce some stuff with lasting impact, though, including more granular app permissions, support for fingerprint readers and support for USB-C.

Google's 2016 Android Nougat (7.0 and 7.1) releases provided Android with a native split-screen mode, a system for organizing notifications and a Data Saver feature. Nougat added some smaller but still significant features, too, like an Alt-Tab-like shortcut for snapping between apps. Perhaps most pivotal among Nougat's enhancements, however, was the launch of the Google Assistant, which came alongside the announcement of Google's first fully self-made phone, the Pixel, about two months after Nougat's debut. The Assistant would go on to become a critical component of Android and most other Google products and is arguably the company's foremost effort today.

Android 8.0 (Oreo) added a variety of niceties to the platform, including a native picture-in-picture mode, a notification snoozing option and notification channels that offer fine control over how apps can alert you. The 2017 release also included some noteworthy elements that furthered Google's goal of aligning Android and Chrome OS and improving the experience of using Android apps on Chromebooks, and it was the first Android version to feature Project Treble, an ambitious effort to create a modular base for Android's code with the hope of making it easier for device-makers to provide timely software updates.

The newest addition to our Android versions list is the freshly baked Android Pie. Android 9 entered the world in early August 2018 after

several months of evolution in public beta previews. Pie's most transformative change is its new gesture navigation system, which trades the traditional Android Back, Home and Overview keys for a single multifunctional Home button and a series of gesture-based commands. elongated Home button and a small Back button that appears as needed. Android 9 boasts numerous other noteworthy productivity features, including a universal suggested-reply system for messaging notifications, a more effective method of screenshot management, and more intelligent systems for power management and screen brightness control. And, of course, there's no shortage of smaller but still-significant advancements hidden throughout Pie's filling, such as a smarter way to handle Wi-Fi hotspots, a welcome twist to Android's Battery Saver mode and a useful new touch for fingerprint sensors. [4]

3.1.1.2 Android platform components

Android architecture components are a collection of libraries that help you design robust, testable, and maintainable apps. Start with classes for managing your UI component lifecycle and handling data persistence. App components are the essential building blocks of an Android app. Each component is an entry point through which the system or a user can enter your app. Some components depend on others. There are four different types of app components: Activities, Services, Broadcast receivers and Content providers. Each type serves a distinct purpose and has a distinct lifecycle that defines how the component is created and destroyed.

- **Activity**: An activity is the entry point for interacting with the user. It represents a single screen with a user interface. An activity facilitates the following key interactions between system and app:
  - Keeping track of what the user currently cares about (what is on screen) to ensure that the system keeps running the process that is hosting the activity; knowing that previously used processes contain things the user may return to (stopped activities), and thus more highly prioritize keeping those processes around.
  - helping the app handle having its process killed so the user can return to activities with their previous state restored;

  - providing a way for apps to implement user flows between each other, and for the system to coordinate these flows.

  An activity is implemented as a subclass of the Activity class.

- **Service**: A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons. It is a component that runs in the background to perform long-running operations or to perform work for remote processes. A service does not provide a user interface. Another component, such as an activity, can start the service and let it run or bind to it in order to interact with it. There are actually two very distinct semantics services tell the system about how to manage an app: Started services tell the system to keep them running until their work is completed. This could be to sync some data in the background or play music even after the user leaves the app. Bound services run because some other app (or the system) has said that it wants to make use of the service. This is basically the service providing an API to another process. The system thus knows there is a dependency between these processes. A service is implemented as a subclass of Service.
- **Broadcast receiver**: A broadcast receiver is a component that enables the system to deliver events to the app outside of a regular user flow, allowing the app to respond to system-wide broadcast announcements. Because broadcast receivers are another well-defined entry into the app, the system can deliver broadcasts even to apps that aren't currently running. Broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs. More commonly, though, a broadcast receiver is just a gateway to other components and is intended to do a very minimal amount of work. A broadcast receiver is implemented as a subclass of BroadcastReceiver and each broadcast is delivered as an Intent object.
- **Content Provider**: A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data if the content provider allows it. To the system, a

content provider is an entry point into an app for publishing named data items, identified by a URI scheme. Thus an app can decide how it wants to map the data it contains to a URI namespace, handing out those URIs to other entities which can in turn use them to access the data. Content providers are also useful for reading and writing data that is private to your app and not shared. A content provider is implemented as a subclass of ContentProvider and must implement a standard set of APIs that enable other apps to perform transactions

- **Intent**: Three of the four component types activities, services, and broadcast receivers are activated by an asynchronous message called an intent. Intents bind individual components to each other at runtime. You can think of them as the messengers that request an action from other components, whether the component belongs to your app or another. An intent is created with an Intent object, which defines a message to activate either a specific component (explicit intent) or a specific type of component (implicit intent).

Before the Android system can start an app component, the system must know that the component exists by reading the app's manifest file, AndroidManifest.xml. Your app must declare all its components in this file, which must be at the root of the app project directory.

The manifest does a number of things in addition to declaring the app's components, such as the following:

- Identifies any user permissions the app requires, such as Internet access or read-access to the user's contacts.
- Declares the minimum API Level required by the app, based on which APIs the app uses.
- Declares hardware and software features used or required by the app, such as a camera, bluetooth services, or a multitouch screen.
- Declares API libraries the app needs to be linked against (other than the Android framework APIs), such as the Google Maps library.

Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus [3].

3.1.2 iPhone Operating System

iOS is a mobile operating system created and developed by Apple Inc. exclusively for its hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod Touch. It is the second most popular mobile operating system globally after Android. iOS has been extended to support other Apple devices such as the iPod Touch (September 2007) and the iPad (January 2010).

iOS utilizes a multi-touch interface in which simple gestures are used to operate the device, such as swiping your finger across the screen to move to the next page or pinching your fingers to zoom out.

3.1.2.1 iPhone OS evolution

Apple announced **iPhone OS 1** at the iPhone keynote on January 9, 2007, and it was released to the public alongside the original iPhone on June 29, 2007. No official name was given on its initial release.

In March 2008 Apple announced **iPhone OS 2**but it was released to the public on July 11, 2008 alongside the iPhone 3G. Apple did not drop support for any devices with this release. iPhone OS 2 was compatible with all devices released up to that time. The release of iPhone OS 2.1.1 brought support for the iPod Touch (2nd generation). The most profound change introduced in this version was the App Store and its support for native, third-party apps. Around 500 apps were available in the App Store at launch. Hundreds of other crucial improvements were also added. Other important changes introduced in the 5 updates iPhone OS 2.0 included podcast support and public transit and walking directions in Maps (both in version 2.2).

iPhone OS 3 was released to the public on June 17, 2009 alongside the iPhone 3GS. The release of this version of the iOS accompanied the debut of the iPhone 3GS. It added features including copy and paste, Spotlight search, MMS support in the Messages app, and the ability to record videos using the Camera app. Also notable about this version of the iOS is that it was the first to support the iPad. The 1st generation iPad was released in 2010, and version 3.2 of the software came with it.

In June 21, 2010 Apple released **OS 4** alongside the iPhone 4. With this release, Apple dropped support for the original iPhone and iPod Touch (1st generation), which is the first time Apple had dropped support for any device in an iOS release. The iPhone 3G and the iPod Touch (2nd generation) were capable of running iOS 4, but had limited features. For example, both devices lack multitasking capabilities and the ability to set a home screen wallpaper. However, iOS 4 was the first major release that iPod Touch users did not have to pay any money for. The release of iOS 4.2.1 brought compatibility to the original iPad and was the final release supported on the iPhone 3G and iPod Touch (2nd generation) due to major performance issues. The release of iOS 4.3 brought iPad 2 compatibility. It became unsupported on 18 December 2013.

iOS 5 was announced on June 6, 2011 at its annual Apple Worldwide Developers Conference (WWDC) event, and it was released to the public on October 12, 2011 alongside the iPhone 4S. Apple did not drop support for any devices with this release; support for the iPhone 3G and the iPod Touch 2nd Generation had already been dropped with the release of iOS 4.3 seven months earlier. Therefore, iOS 5 was released for the iPhone 3GS onwards, iPod Touch (3rd generation) onwards, and all iPad models. With this release Apple responded to the growing trend of wirelessness, and cloud computing, in iOS 5, by introducing essential new features and platforms. Among those was iCloud, the ability to activate an iPhone wirelessly (previously it had required a connection to a computer), and syncing with iTunes via Wi-Fi.

iOS 6 was released to the public on September 19, 2012 alongside the iPhone 5, iPod Touch (5th generation), and iPad 4. With this release, Apple dropped support for the iPod Touch (3rd generation) and the iPad (1st generation) due to hardware limitations, and offered only limited support on the iPhone 3GS, iPad 2, and iPod Touch (4th generation). iOS 6.1.6 was the final release supported for the iPhone 3GS and iPod Touch (4th generation). This version introduced the world to Siri which, despite being later surpassed by competitors, was a truly revolutionary technology. Apple introduced its own Maps app too, which was badly received due to bugs, bad directions, and problems with certain features.

Apple announced **iOS 7** on June 10, 2013 at its annual Apple Worldwide Developers Conference (WWDC) event, and it was released to the public on September 18, 2013 alongside the iPhone 5C and iPhone 5S. With this release, Apple dropped support for the iPhone 3GS (due to hardware limitations) and the iPod Touch (4th generation) (due to performance issues). In this version of the iOS, Apple ushered in a major overhaul of the user interface, designed to make it more modern.

Apple released **iOS 8** to the public on September 17, 2014 alongside the iPhone 6 and iPhone 6 Plus. The release of iOS 8.1 brought support for the iPad Air 2 and iPad Mini 3, and the release of iOS 8.4 brought support for the iPod Touch (6th generation). iOS 8.3 was the first version of iOS to have public beta testing available, where users could test the beta for upcoming releases of iOS and send feedback to Apple about bugs or glitches. The final version of iOS 8 was iOS 8.4.1. With the radical changes of the last two versions now in the past, Apple once again focused on delivering major new features. Among these features was its secure, contactless payment system Apple Pay and, with the iOS 8.4 update, the Apple Music subscription service. There were continued improvements to the iCloud platform, too, with the addition of the Dropbox-like iClould Drive, iCloud Photo Library, and iCloud Music Library.

Apple announced **iOS 9** on June 8, 2015, and released it to the public on September 16, 2015 alongside the iPhone 6S, iPhone 6S Plus and iPad Mini 4. With this release, Apple did not drop support for any iOS devices. Therefore, iOS 9 was supported on the iPhone 4S onwards, iPod Touch (5th generation) onwards, the iPad 2 onwards, and the iPad Mini (1st generation) onwards. This release made the iPad 2 the first device to support six major releases of iOS, supporting iOS 4 to 9. Despite Apple's promise of better performance on these devices, there were still widespread complaints that the issue had not been fixed. iOS 9.3.5 is the final release on the iPhone 4S, iPad 2 and 3, iPod Touch (5th generation) and iPad Mini (1st generation). This release was generally aimed at solidifying the foundation of the OS for the future. Major improvements were delivered in speed and responsiveness, stability, and performance on older devices.

In September 13, 2016 Apple released **iOS 10** alongside the iPhone 7 and iPhone 7 Plus. This version has limited support on the iPhone 5, iPhone

5C, and iPad 4 because those devices have 32bit processors. However, the iPhone 5S onwards, iPod Touch (6th generation) onwards, and the iPad Mini 2 onwards are fully supported. The major themes of iOS 10 were interoperability and customization. Apps could now communicate directly with each other on a device, allowing one app to use some features from another without opening the second app Siri became available to third party apps in new ways.

iOS 11 was announced on June 5, 2017 and released to the public on September 19, 2017 alongside the iPhone 8 and iPhone 8 Plus. With this release, Apple dropped support for the 32bit iPhone 5 and iPhone 5C, and the iPad 4, making iOS a 64bit only OS that only runs 64bit apps. All other devices from the iPhone 6S 6S Plus onwards, iPad Pro onwards, and iPad (2017) onwards are fully supported. iOS 11 contains lots of improvements for the iPhone, but its major focus is turning the iPad Pro series models into legitimate laptop replacements for some users.

Apple announced **iOS 12** on June 4, 2018 and released it to the public on September 17, 2018 alongside the iPhone XS and iPhone XS Max. With this release, Apple did not drop support for any iOS devices. Therefore, iOS 12 was supported on the iPhone 5S onwards, iPod Touch (6th generation) onwards, the iPad Air onwards, and the iPad Mini 2 onwards. The new features and improvements added in iOS 12 aren't as extensive or revolutionary as in some previous updates to the OS. Instead, iOS 12 focused more on making refinements to commonly used features and on adding wrinkles that improve how people use their devices [12] [13].

3.1.2.2 iOS application structure

iOS applications are different from Android ones. When you create an app it must have different resources and metadata so that it can be displayed properly on iOS devices:

- An information property-list file. The Info.plist file contains metadata about your app, which the system uses to interact with your app.
- A declaration of the apps required capabilities. Every app must declare the hardware capabilities or features that it requires to run.

- One or more icons. The system displays your app icon on the home screen of a users device. The system may also use other versions of your icon in the Settings app or when displaying the results of a search.
- One or more launch images. When an app is launched, the system displays a temporary image until the app is able to present its user interface.

These resources are required for all apps but are not the only ones you should include.

During startup, the *UIApplicationMain* function sets up several key objects and starts the app running. Note that iOS apps use a model-view-controller architecture. This pattern separates the apps data and business logic from the visual presentation of that data. This architecture is crucial to creating apps that can run on different devices with different screen sizes.

The *UIApplication* object manages the event loop and other high-level app behaviors. It also reports key app transitions and some special events (such as incoming push notifications) to its delegate, which is a custom object you define. Use the UIApplication object as isthat is, without subclassing.

The app delegate is the heart of your custom code. This object works in tandem with the *UIApplication* object to handle app initialization, state transitions, and many high-level app events. This object is also the only one guaranteed to be present in every app, so it is often used to set up the apps initial data structures.

Data model objects store your apps content and are specific to your app. For example, a banking app might store a database containing financial transactions, whereas a painting app might store an image object or even the sequence of drawing commands that led to the creation of that image. (In the latter case, an image object is still a data object because it is just a container for the image data.)

Apps can also use document objects (custom subclasses of UIDocument) to manage some or all of their data model objects. Document objects are

not required but offer a convenient way to group data that belongs in a single file or file package.

View controller objects manage the presentation of your apps content on screen. A view controller manages a single view and its collection of subviews. When presented, the view controller makes its views visible by installing them in the apps window.

The *UIViewController* class is the base class for all view controller objects. It provides default functionality for loading views, presenting them, rotating them in response to device rotations, and several other standard system behaviors. UIKit and other frameworks define additional view controller classes to implement standard system interfaces such as the image picker, tab bar interface, and navigation interface.

A *UIWindow* object coordinates the presentation of one or more views on a screen. Most apps have only one window, which presents content on the main screen, but apps may have an additional window for content displayed on an external display. To change the content of your app, you use a view controller to change the views displayed in the corresponding window. You never replace the window itself. In addition to hosting views, windows work with the *UIApplication* object to deliver events to your views and view controllers.

Views and *controls* provide the visual representation of your apps content. A view is an object that draws content in a designated rectangular area and responds to events within that area. Controls are a specialized type of view responsible for implementing familiar interface objects such as buttons, text fields, and toggle switches. The *UIKit* framework provides standard views for presenting many different types of content. You can also define your own custom views by subclassing UIView (or its descendants) directly. In addition to incorporating views and controls, apps can also incorporate *Core Animation* layers into their view and control hierarchies. Layer objects are actually data objects that represent visual content. Views use layer objects intensively behind the scenes to render their content. You can also add custom layer objects to your interface to implement complex animations and other types of sophisticated visual effects [5]

3.2 Android applications development

Android applications are mainly based on java language. In general, creating an Android app requires the SDK (Software Development Kit), an IDE (Integrated Development Environment) like Android Studio or Eclipse, the Java Software Development Kit (JDK) and a virtual device to test on.

Android applications can be developed with two different programming languages Java and Kotlin.

3.2.1 Kotlin

Kotlin is a statically typed programming language that runs on the Java virtual machine and also can be compiled to JavaScript source code or use the LLVM compiler infrastructure. It is sponsored and developed by JetBrains. While the syntax is not compatible with Java, the JVM implementation of the Kotlin standard library is designed to interoperate with Java code and relies on Java code from the existing Java Class Library, such as the collections framework. Kotlin is: **Concise** it reduces the amount of boilerplate code. **Safe** it avoids entire classes of errors (null pointer exceptions, etc.). **Interoperable** it leverages existing libraries for the JVM, Android and browsers. **Tool-friendly** applications could be build from any Java IDE or from the command line.

3.2.2 Java

Java is a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture. Java was designed with a few key principles in mind: **Ease of Use**: The fundamentals of Java came from a programming language called C++. Although C++ is a powerful language, it is complex in its syntax and inadequate for some of Java's requirements. Java built on and improved the ideas of C++ to provide a programming language that was powerful and simple to use. **Reliability**:

Java needed to reduce the likelihood of fatal errors from programmer mistakes. With this in mind, object-oriented programming was introduced. When data and its manipulation were packaged together in one place, Java was robust. **Security**: Because Java was originally targeting mobile devices that would be exchanging data over networks, it was built to include a high level of security. Java is probably the most secure programming language to date. **Platform Independence**: Programs need to work regardless of the machines they're being executed on. Java was written to be a portable and cross-platform language that doesn't care about the operating system, hardware, or devices that it's running on.

3.2.3 Android Studio

The most popular Android application development tool is Android Studio. Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development. Android studio installation provide SDK tools and ADB manager to compile and test applications trought emulators.

3.2.4 Other tools

1. ADB (Android Debug Bridge): Android Studio includes the Android Debug Bridge, which is a command-line tool or bridge of communication between Android devices and other computers that can be used during development and the overall debugging and QA process. By connecting an Android device to the development PC and entering a series of terminal commands, a developer is able to make modifications as needed to both devices.
2. AVD Manager: Another useful feature of Android Studio is the AVD Manager, the short form for Android Virtual Device. The AVD Manager is an emulator used to run Android apps on a computer. This

allows developers the ability to work with all types of Android devices to test responsiveness and performance on different versions, screen sizes, and resolutions.

3. Eclipse: As we mentioned above, there was Eclipse before there was Android Studio. For a long time, Eclipse was the officially preferred IDE for all Android application development. Even though Google no longer offers support for Eclipse, many developers still use it to create Android and other cross-platform apps, as it works very well with many different programming languages.
4. Fabric: Fabric is the development platform behind Twitters mobile application. It gives developers the ability to build better mobile apps by providing them with a suite of kits that they can pick and choose from. These kits include everything from beta-testing to marketing and advertising tools. Google purchased Fabric from Twitter in January of 2017. Uber, Spotify, Square, Groupon, Yelp, and more big-name companies have utilized Fabric in developing their mobile application
5. FlowUp: FlowUp allows you to monitor the performance of all your production apps. Handy dashboards let you keep track of your stats and metrics, including CPU and disk usage, memory usage, frames per second, bandwidth, and more.
6. Gradle: Back in 2013, Google endorsed Gradle as a build system for Android apps. Based on Apache Maven and Apache Ant, Gradle is one of the most popular development tools for creating large-scale applications involving Java. Developers like using Gradle in conjunction with Android Studio because its very easy to add external libraries using a single line of code.
7. IntelliJ IDEA: From the developers at JetBrains, IntelliJ IDEA is designed for ultimate programmer productivity. Its extremely fast and features a full suite of development tools right out of the box.
8. RAD Studio: RAD Studio is an integrated development environment that allows you to write, compile, package, and deploy cross-platform applications. It provides support for the full development lifecycle resulting in a single source codebase that can be recompiled and redeployed.

9. Unity 3D: Unity 3D is a cross-platform game development environment used for creating complicated, graphics-intensive mobile games such as those containing virtual or augmented reality. You can still use Unity 3D to create simpler 2D-based gaming experiences, but it is more typically used for advanced gaming development.
10. Unreal Engine: Another advanced gaming development platform, Unreal Engine is a free, open-source, cross-platform solution for creating high-level interactive games.
11. Visual Studio with Xamarin: Visual Studio is Microsofts official integrated development environment and is a free tool for developers to use. It supports several different programming languages and when combined with Xamarin, it can be utilized to create native Windows, Android, and iOS applications [22].

There are literally hundreds of other useful tools such as these available for Android development. Each developer has their own personal preference for what tools and environments they work with based on the particular application they are developing. As the demand for Android applications continues to grow, the pool of platforms and solutions that help save developers time while helping to produce higher quality apps will continue to increase as well.

3.3 iOS applications development

To develop iOS apps, you need a Mac computer running the latest version of Xcode. Xcode includes all the features you need to design, develop, and debug an app. Xcode also contains the iOS SDK, which extends Xcode to include the tools, compilers, and frameworks you need specifically for iOS development.

Languages used to develop applications are Swift and Objective C.

3.3.1 Swift

Swift is a general-purpose programming language built using a modern approach to safety, performance, and software design patterns.

The goal of the Swift project is to create the best available language for uses ranging from systems programming, to mobile and desktop apps, scaling up to cloud services. Most importantly, Swift is designed to make writing and maintaining correct programs easier for the developer. To achieve this goal, we believe that the most obvious way to write Swift code must also be:

Safe: The most obvious way to write code should also behave in a safe manner. Undefined behavior is the enemy of safety, and developer mistakes should be caught before software is in production. Opting for safety sometimes means Swift will feel strict, but we believe that clarity saves time in the long run.

Fast. Swift is intended as a replacement for C-based languages (C, C++, and Objective-C). As such, Swift must be comparable to those languages in performance for most tasks. Performance must also be predictable and consistent, not just fast in short bursts that require clean-up later. There are lots of languages with novel features being fast is rare.

Expressive. Swift benefits from decades of advancement in computer science to offer syntax that is a joy to use, with modern features developers expect. But Swift is never done. We will monitor language advancements and embrace what works, continually evolving to make Swift even better [2].

3.3.2 Objective C

Objective C is an object-oriented language and hence, it would be easy for those who have some background in object-oriented programming languages.

Objective-C is the primary programming language you use when writing software for OS X and iOS. Its a superset of the C programming language and provides object-oriented capabilities and a dynamic runtime. Objective-C inherits the syntax, primitive types, and flow control statements of C and adds syntax for defining classes and methods. It also adds language-level support for object graph management and object literals while providing dynamic typing and binding, deferring many responsibilities until runtime. [1]

3.3.3 Xcode

The tool used to create iOS application is Xcode. Xcode is an integrated development environment (IDE) for macOS containing a suite of software development tools developed by Apple for developing software for macOS, iOS, watchOS, and tvOS. First released in 2003, the latest stable release is version 10.1 and is available via the Mac App Store free of charge for macOS High Sierra and macOS Mojave users.[2] Registered developers can download preview releases and prior versions of the suite through the Apple Developer website

Xcode supports source code for the programming languages C, C++, Objective-C, Objective-C++, Java, AppleScript, Python, Ruby, ResEdit (Rez), and Swift, with a variety of programming models, including but not limited to Cocoa, Carbon, and Java. Third parties have added support for GNU Pascal Free Pascal, Ada, C#,Perl, D and Fortran.

Xcode can build fat binary files containing code for multiple architectures with the Mach-O executable format. These are called universal binary files, which allow software to run on both PowerPC and Intel-based (x86) platforms and that can include both 32-bit and 64-bit code for both architectures. Using the iOS SDK, Xcode can also be used to compile and debug applications for iOS that run on ARM architecture processors.

Xcode includes the GUI tool Instruments, which runs atop a dynamic tracing framework, DTrace, created by Sun Microsystems and released as part of OpenSolaris[26].

3.4 Cross platform applications development

Cross-platform mobile development is the creation of software applications that are compatible with multiple mobile operating systems. Originally, the complexity of developing mobile apps was compounded by the difficulty of building out a backend that worked across multiple platforms. Although it was time-consuming and expensive, it was often easier to build native applications for each mobile operating system (OS). The problem was that the code built for one operating system could not be repurposed for another OS.

Cross-platform mobile development tools have been developed with the purpose to give them the possibility to write the application source code once and run it on different OSs [43].

Major benefits that these tools have brought are:

- Reduction of required skills for developers to develop applications due to the use of common programming languages;
- Reduction of coding, because the source code is written once and it is compiled for each supported OS;
- Reduction of development time and long term maintenance costs;
- Decrement of API knowledge, because with these tools;
- Greater ease of development compared to building native applications for each OS;

Approaches to cross-platform development include: 1. Hybrid mobile app development: developers write the core of the application as an HTML5 or JavaScript mobile app and then place a native device wrapper around it. 2. Rapid mobile app development (RMAD): developers use code-free programming tools. RMAD offers business users the ability to quickly build and manage good-enough internal apps to address specific business issues. 3. Windows universal apps: one codebase for all Windows devices. The goal is to enable the same app to run on a Windows PC, tablet, smartphone, smartwatch or XBox. 4. Progressive web apps (PWAs): websites that look and behave as if they are mobile apps. PWAs are built to take advantage of native mobile device features, without requiring the

end user to visit an app store, make a purchase and download software locally [7].

Most popular cross platform development tools are: Xamarin, PhoneGap, Sencha, Appcelerator, iFactr, Kony, AlphaAnywhere, Redhat.

Anyway, cross platform development present some limitations. Though cross platform mobile apps are faster and friendly, but they have less performance quality when compared to native apps. Since each app is designed on one or more OS platforms, it is hard to be supported by every feature of OS. Each time the OS gets updated with new features, your app also needs an update. A connection needs to be set up for starting up a cross platform app, which is not needed with native apps. Cross platform mobile applications has restrictions to some of the hardware features, as these apps come up with multiple mobile OS platforms. Lot of frameworks are build using Javascript and hence if you wish to move from one OS to another, the code you already used is not going to work out. It can be made reusable by putting in a lot of work on your codes. Hybrid apps come up with features and limitations that could either delight or disappoint any developer.

3.5 Why an UML extension for Android applications?

Studying approaches about mobile applications modeling support and analysing mobile applications context, we decided to focus our researches on Android native applications. The reason that drove us to propose and evaluate Android specific UML extension concerns Android devices popularity. Analysing smartphone usage, stores presence and downloads, we observed a growing usage of Android applications and devices and so we decided to provide support for Android development processes and developers. Further more, we found useful the open source nature of such applications. In fact there are different repositories (F-Droid, GitHub, etc.) and there is an open access to source code of a big number of Android apps. The idea was so to analyse applications as described in chapter 2, propose an UML extension (DROID UML) and evaluate its utility through a controlled experiment.

Chapter 4

Droid UML: an UML extension for Android native applications

In this chapter we present the proposed Droid UML extension. We described android applications components we will model and analysed source code of applications to understand if there were other components to extend. To propose extension we decided to present the Unified Modeling Language definition with particular attention to Class Diagrams and to the extension mechanism. So we formalized the extension.

4.1 UML: Unified Modeling Language

Modeling is the designing of software applications before coding. Modeling is an Essential Part of large software projects, and helpful to medium and even small projects as well. A model plays the analogous role in software development that blueprints and other plans (site maps, elevations, physical models) play in the building of a skyscraper.

The OMG's Unified Modeling Language (UML) helps you specify, visualize, and document models of software systems, including their structure and design, in a way that meets all of these requirements [25]. UML makes these artifacts scalable, secure and robust in execution. UML is an important aspect involved in object-oriented software development. It uses graphic notation to create visual models of software systems.

UML has been evolving since the second half of the 1990s and has its roots in the object-oriented programming methods developed in the late 1980s and early 1990s. The timeline (see image) shows the highlights of the history of object-oriented modeling methods and notation.

Under the technical leadership of those three (Rumbaugh, Jacobson and Booch), a consortium called the UML Partners was organized in 1996 to complete the Unified Modeling Language (UML) specification, and propose it to the Object Management Group (OMG) for standardization. The partnership also contained additional interested parties (for example HP, DEC, IBM and Microsoft). The UML Partners' UML 1.0 draft was proposed to the OMG in January 1997 by the consortium. During the same month the UML Partners formed a group, designed to define the exact meaning of language constructs, chaired by Cris Kobryn and administered by Ed Eykholt, to finalize the specification and integrate it with other standardization efforts. The result of this work, UML 1.1, was submitted to the OMG in August 1997 and adopted by the OMG in November 1997.

UML 2.0 major revision replaced version 1.5 in 2005, which was developed with an enlarged consortium to improve the language further to reflect new experience on usage of its features.

A UML model consists of three major categories of model elements, each of which may be used to make statements about different kinds of individual

things within the system being modeled (termed simply individuals in the following). These categories are:

- Classifiers. A classifier describes a set of objects. An object is an individual with a state and relationships to other objects. The state of an object identifies the values for that object of properties of the classifier of the object. (In some cases, a classifier itself may also be considered an individual; for example, see the discussion of static structural features in sub clause 9.4.3.)
- Events. An event describes a set of possible occurrences. An occurrence is something that happens that has some consequence with regard to the system.
- Behaviors. A behavior describes a set of possible executions. An execution is a performance of a set of actions (potentially over some period of time) that may generate and respond to occurrences of events, including 12 Unified Modeling Language 2.5.1 accessing and changing the state of objects. (As described in sub clause 13.2, behaviors are themselves modeled in UML as kinds of classifiers, so that executions are essentially modeled as objects. However, for the purposes of the present discussion, it is clearer to consider behaviors and executions to be in a separate semantic category than classifiers and objects.)

UML 2.0 defines thirteen types of diagrams, divided into three categories (see Figure 4.1: Six diagram types represent static application structure; three represent general types of behavior; and four represent different aspects of interactions:

- Structure Diagrams: include the Class Diagram, Object Diagram, Component Diagram, Composite Structure Diagram, Package Diagram, and Deployment Diagram.
- Behavior Diagrams: include the Use Case Diagram (used by some methodologies during requirements gathering); Activity Diagram, and State Machine Diagram.
- Interaction Diagrams: all derived from the more general Behavior Diagram, include the Sequence Diagram, Communication Diagram, Timing Diagram, and Interaction Overview Diagram.

Structure diagrams are:

- Class Diagram: represents system class, attributes and relationships among the classes.
- Component Diagram: represents how components are split in a software system and dependencies among the components.
- Deployment Diagram: describes the hardware used in system implementations.
- Composite Structure Diagram: describes internal structure of classes.
- Object Diagram: represents a complete or partial view of the structure of a modeled system.
- Package Diagram: represents splitting of a system into logical groupings and dependency among the grouping.

Behavior and interaction diagrams are:

- Activity Diagram: represents step by step workflow of business and operational components.
- Use Case Diagram: describes functionality of a system in terms of actors, goals as use cases and dependencies among the use cases.
- UML State Machine Diagram: represents states and state transition.
- Communication Diagram: represents interaction between objects in terms of sequenced messages.
- Timing Diagrams: focuses on timing constraints.
- Interaction Overview Diagram: provides an overview and nodes representing communication diagrams.
- Sequence Diagram: represents communication between objects in terms of a sequence of messages.

UML diagrams represent static and dynamic views of a system model. The static view includes class diagrams and composite structure diagrams, which emphasize static structure of systems using objects, attributes, operations and relations. The dynamic view represents collaboration among objects and changes to internal states of objects through sequence, activity and state machine diagrams.

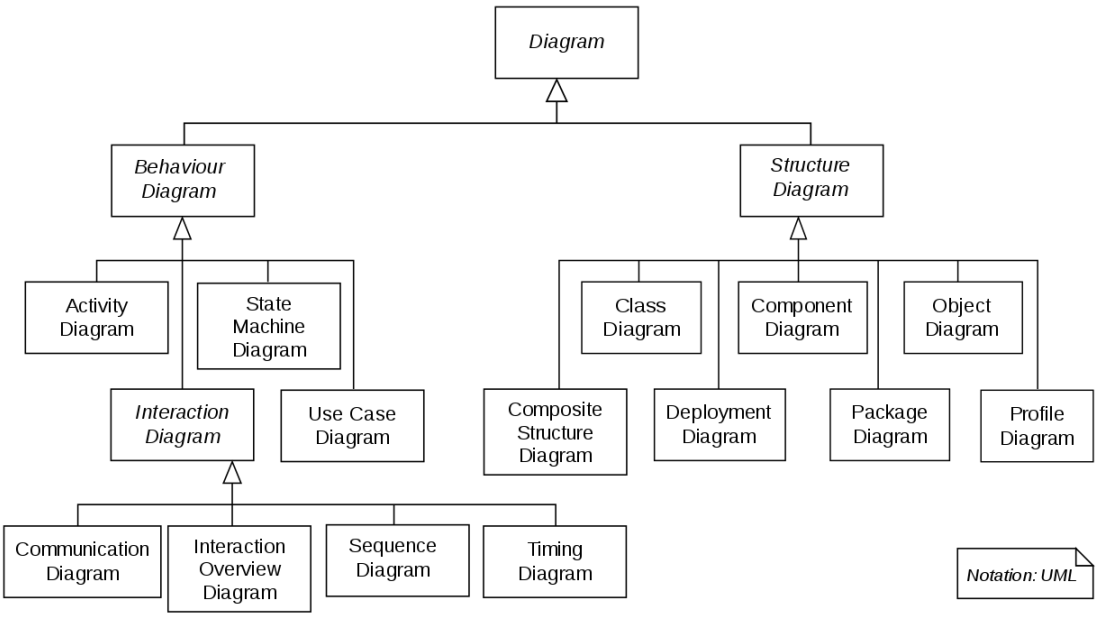

Figure 4.1: UML diagrams overview

In our approach we decided to use Class Diagram of UML standard because it represent the best way to model system classes of an Android application.

4.1.1 UML Class Diagram

Class Diagram represents type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

The class diagram is the main building block of object-oriented modeling. It is used for general conceptual modeling of the systematic of the application, and for detailed modeling translating the models into programming code. Class diagrams can also be used for data modeling.The classes in a class diagram represent both the main elements, interactions in the application, and the classes to be programmed.

Using class diagrams UML provides relationships. A relationship is a general term covering the specific types of logical connections found on class and object diagrams.

UML defines relationship between classes. Instance-level relationships:

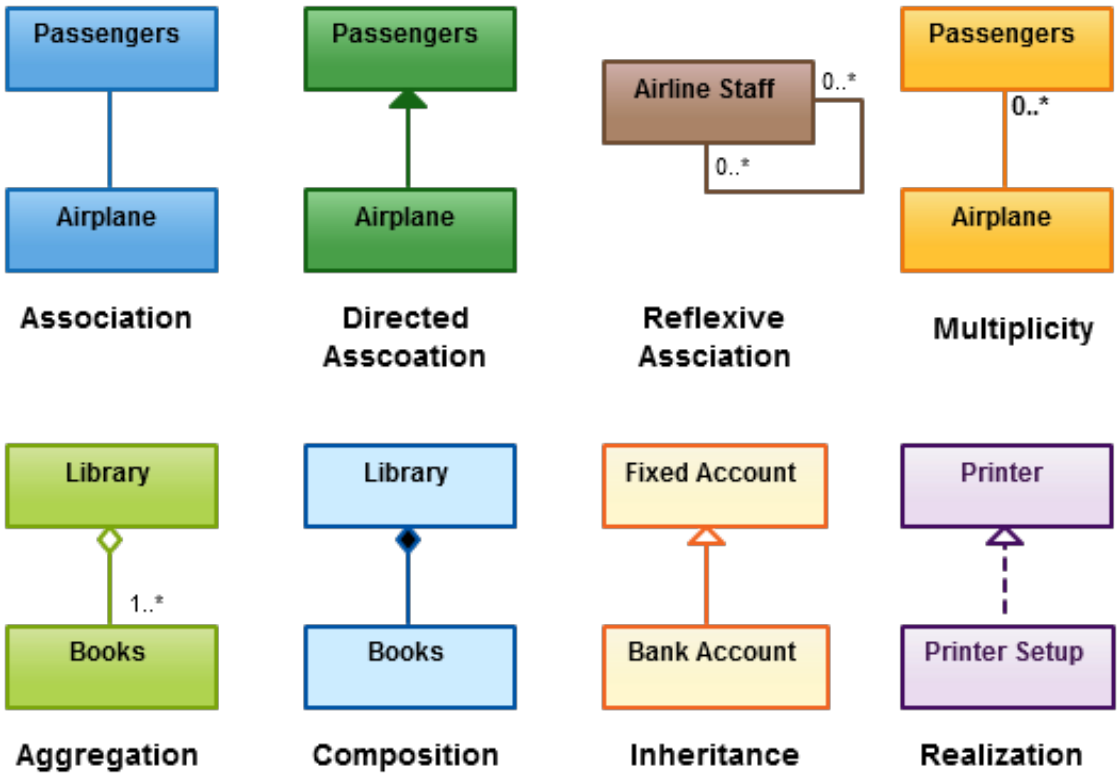

Figure 4.2: Class diagram relationship examples

Dependency, Association, Aggregation, Composition; class-level relationships: Generalization/Inheritance, Realization/Implementation; General relationship: Dependency, Multiplicity [6]. Figure 4.2 show example of class diagrams relationships.

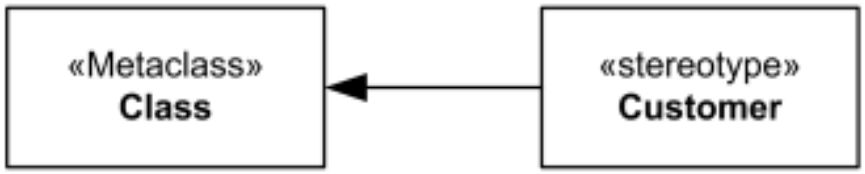

Figure 4.3: Extension stereotype example

4.2 The UML extension mechanism

UML provide an extension mechanism to indicate that the properties of a metaclass are extended through a stereotype, and gives the ability to flexibly add (and later remove) stereotypes to classes.

Extension mechanisms are the means for customizing and extending the UML. UML extension mechanisms are based on Stereotypes, Tagged Values, and Constraints. Briefly, stereotypes are means of extending the UML vocabulary. They are used for introducing new types of model elements. Each stereotype defines a set of properties that are received by elements of that stereotype as well as rules that must be satisfied by elements of that stereotype[50].

The notation for an extension is an arrow with the filled triangle arrowhead pointing from a stereotype to the extended metaclass as shown in figure 4.3.

4.3 Android UML proposed extension

Starting from approach proposed by Ko et al. [38] we decided to model structural features of an Android application considering user interface features and application fundamentals. In particular, we extended the class diagram of the UML standard adding several Android-specific components. To understand what are the most used object by developers and, therefore, to decide which are the most important components that need to be modeled, we analyzed the source code of 100 popular applications. In this section we first describe the most important Android components and we describe in details how we decided the components that needed to be modeled; then, we formally introduce Droid UML, our Android-specific UML extension.

4.3.1 Modeled Android Components

As described in chapter 3, Android is an operating system for mobile devices. It is an open source platform designed to simply reuse components. Android applications could be written using Kotlin, Java, and C++ languages. Android applications are composed mainly by four different types of components: *activities*, *services*, *broadcast receivers* and *content providers*. Each type of component has a distinct purpose and lifecycle. An *activity* is the component interacting with the user; a *service* is a component that keeps app running in the background; a *broadcast receiver* enables the system to deliver events to the app; a *content provider* manages data that need to be stored. Besides, Android provides an asynchronous activation mechanism called *intent*. Intents bind components to each other at runtime. [3]

To understand if Android application components as proposed by Ko et al. [38] were enough to model an application, we analyzed 100 Android apps. Specifically, we considered open-source projects from F-Droid. Starting from a list of repositories, we chose randomly 100 GitHub repository and launched a script that cloned them and analysed source code.To extract the most used components, we associated to each class its superclass. Then, we counted the occurrences of each superclass, filtering only the ones from the Android SDK.

Object type	Number of occurrences	Frequency
Activity	423	11,45%
Fragment	370	10 %
AsyncTask	285	7,7 %
Widget	177	4,8%
Service	114	3%
BroadcastReceiver	92	2,5%
Layout	86	2.32%

Table 4.1: Most used Android components

Table 4.1 contains the result of the analysis done on 3694 classes obtained from source code of 100 Android repositories. Starting from the data outlined by code analysis, we defined our extension (Droid-UML) starting from the core Android components as defined in the official documentation, and we added to such components the most frequent ones we found, *Fragment Adapters*, *Widgets*, *AsyncTasks* and *Layouts*. It is worth noting that *Intents* could not be found in our analysis, since they are usually not extended.

4.4 Droid UML extension definition

We describe the objects extended giving a formal definition, specifying what kind of UML component they extend, their constraints, and showing graphical icons. We report in 4.2 a summary of the components and the icons we used in our model.

Activity. An Activity represent a single screen where the user directly interact with the Android app. It is a container in which developers place visual elements called views (also known as widgets).

Service. A Service is a component that runs in the background and does not have any graphical user interface. While the user works on the interface in the foreground, services could manage processes that needs to run in the background. There are two types of services: unbound services and bound services. The first are services not bounded by any component. Once started they run in the background indefinitely, even if the component that started them is destroyed. They can be stopped after they completed their task. Bound services, instead, run only as long as another application component is bound to them. Multiple components can bind the service at once, but when all of them unbind, the service is destroyed.

Broadcast Receiver. A Broadcast Receiver is a component used to receive messages sent in a broadcast way from Android system or other applications. There are different broadcast messages initiated by the Android system itself and caught by other applications using Broadcast receivers. Battery warning, screen turned off, change time zone, camera used, etc..

Content Provider. A Content Provider provides a flexible way to make the data available across applications. Other applications are able to query, access or modify data using this component. Content Providers give also access to data provided from other utilities.

Intent. It represent the mechanism of activation of components. Intents are the main system of communication they define how to activate other components. There are two types of intents: *explicit intent* and *implicit intent*. An explicit intent is an intent which requires component specification when activated. An implicit intent, instead, sends a message to Android system to find a component that meets the intent.

Fragment Adapter. A Fragment Adapter represents a UI page as a fragment that is persistently kept in memory (in a so called Fragment Manager) as long as the user can return to the such a page. It is often used when in the application there is a set of static fragments to be paged or a set of tabs. The fragment of each page will be kept in memory and destroyed when not visible.

Widget. A Widget is an object that can be embedded in other applications or activities. It is a view object that composes the UI. There are different types of widgets and each of them extends a specific class of SDK (Button, Image, and Clock).

AsyncTask An AsyncTask is a component that allows to perform background operations and publish results on the UI thread without having to create threads and handlers. AsyncTasks should be ideally used for relatively short operations (few second at most).

Layout. It defines the visual structure of a user interface. Layouts represent the way to graphically organize widgets in the UI. There are several types of layouts, such as linear and relative layouts.

The components defined above extend the concept of class object provided by standard UML. We report in figures 4.4 and 4.5 the basic class diagrams for user interface components and structural (non-visual) components, respectively. The idea is to use the extension mechanism adding to class entity of class diagram graphical stereotypes.

Figures 4.6 and 4.7 instead, present proposed Droid UML extension.

Starting from this general definition we later try to propose a controlled experiment with real applications to evaluate utility.

Android object	UML type	Costraints	Icon
Activity	class	It has to contain at least one layout component	
Service	class	It has to be strarted by an activity or another service	
Broadcast receiver	class	none	
Content provider	class	none	
Intent	class	It has to be launched by an activity and it has to launch a new activity	
Fragment adapter	class	It has to be contained in an activity; it has at leas one page (fragment)	
Widget	class	It has to be contained in a layout	
Async task	class	none	
Layout	class	It has to be contained in an activity or subclasses	

Table 4.2: Most used Android components

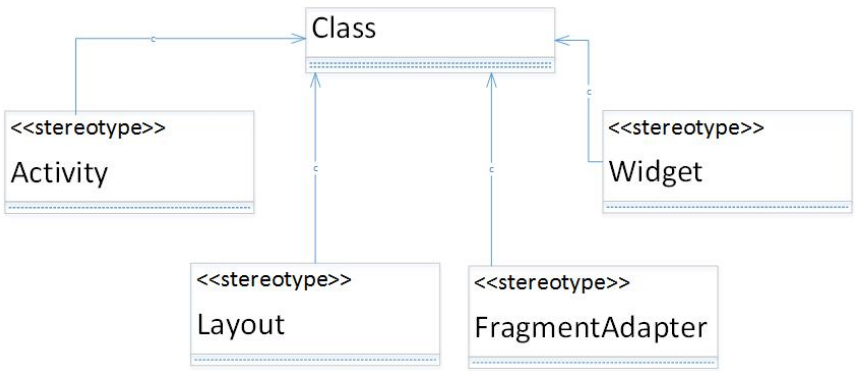

Figure 4.4: Standard UML class diagram for Android user interface features

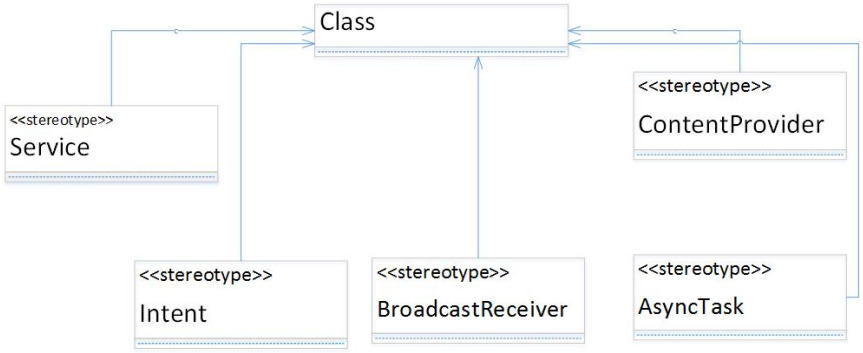

Figure 4.5: Standard UML class diagram for Android structural components

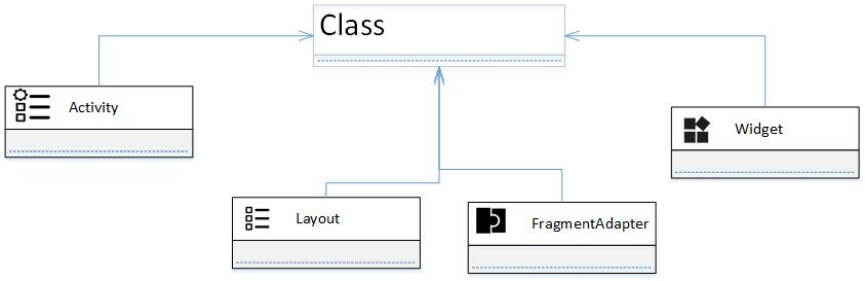

Figure 4.6: Droid UML class diagram for Android structural components

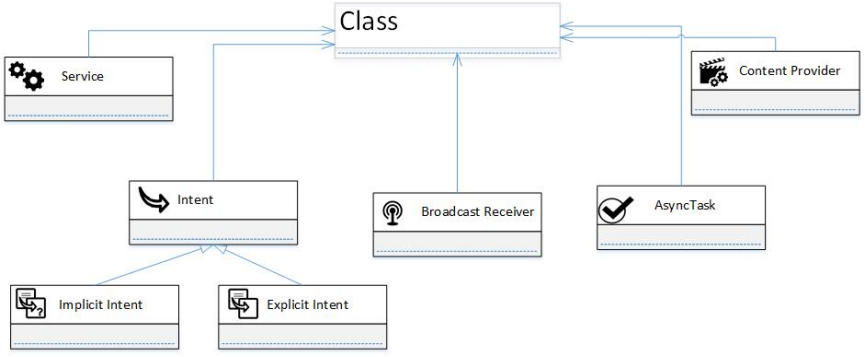

Figure 4.7: Droid UML class diagram for Android structural components

Chapter 5

Case study: a controlled experiment to evaluate Droid UML extension

In this chapter we present the experiment carried out to evaluate extension proposed. The idea is to design and submit, to a set of developers, a survey with two different maintenance tasks and evaluate the differences for the tasks supported by plain UML and the tasks supported by Droid UML.

5.1 Experiment design

The goal of our empirical study is to check if Droid UML can support developers in the comprehension of an Android project while performing a maintenance task.

Our study was steered by the following research questions:

- RQ1: What is the effectiveness of Droid-UML? With this research questions we want to investigate what is the level of accuracy of the responses given during the task completion.
- RQ2: How do developers perceive the Droid-UML extension? With this second research question we try to understand if developers find Droid UML diagrams useful.

To answer the questions we designed a controlled experiment based on a survey submission to developers and on survey result evaluation.

5.1.1 Experiment Design

The context of the experiment is based on two Android applications X and Y and a set of participant selection. We will ask to developer a set of questions to understand a detailed application mechanism. To answer questions, developer will be supported by applications source code and an UML diagram in the different two version (one task with Droid UML and one with plain UML). Every developer will compile a preliminary questionnaire and a post questionnaire. The preliminary one contains questions about programming experience, actual work employment, UML knowledge. More specifically :

1. Current job.
2. General programming experience (years).
3. Java programming experience (evaluation).
4. Android application development/maintenance experience (yes, no).
5. Android application programming experience (years).
6. Android application programming experience (evaluation).
7. UML usage experience (evaluation).

Table 5.1: Experiment survey organization

Group	Task 1	Model Type	Task 2	UML type
1	NextCloud News Reader	Plain UML	Simple Alarm Clock	Droid UML
2	Simple Alarm Clock	Plain UML	NextCloud News Reader	Droid UML
3	NextCloud News Reader	Droid UML	Simple Alarm Clock	Plain UML
4	Simple Alarm Clock1	Droid UML	NextCloud News Reader	Plain UML

The post questionnaire contains questions about qualitative evaluation of the proposed extension:

1. How useful is Droid UML.
2. How simple is to understand Droid UML.
3. Activity stereotype utility.
4. Fragment adapter stereotype utility.
5. Intent stereotype utility.
6. Further information.

For each application we selected a task defined by choosing a solved issue on GitHub repository of the app. Every questionnaire will contain questions about the two tasks of the two applications. We linked source code and UML diagram (Droid and plain) to support each task resolution. The survey organization is resumed in table 5.1. To guarantee coverage of all possible combinations of task and UML extension we need 4 different questionnaires.

5.1.2 Application selection and task definition

The next step, was focused to select two Android native applications from GitHub. Basing on our previous investigation reported in 2.1.3, we selected the two Android apps with highest number of newcomers written in Java programming language. We decided to select Java applications because it is the most popular programming language for Android applications and it is simpler find Android programmers using that languages. There was different applications written in Kodin code but we excluded them. The app selected are Simple Alarm Clock[20], an alarm clock, and NextCloud News Reader[18], an app that allows to connect to the OwnCloud News Reader server app and read RSS feeds. For both such apps,

we used two resolved issues from the GitHub issue tracker to define the tasks for our study. More specifically, each task consisted in the detection of the classes that would be involved to complete the task. Here the 2 task definition:

- Task 1: Simple Alarm Clock application. The task consists to detect classes and object involved in the action of adding a context menu to dismiss alarm snooze notification.
- Task 2: NextCloud News Reader application. The task consists to detect classes and object involved in allowing user to open links in an external browser and made URLs visible.

The two applications selected did not provide any UML class diagrams. Therefore, we downloaded the snapshot of the apps after the fix of the issue was committed and we reverse-engineered both starting from the source code and we defined plain UML class diagrams. We obtained UML class diagram using the UML Generator plugin realized for Eclipse software development tool.Starting from this plain UML class diagrams, we manually refine them and then we manually define corresponding Droid UML diagrams. We report in Figure 5.1 the plain UML class diagram created for Simple Alarm Clock and in Figure 5.2, the Droid UML class diagram created for NextCloud News Reader.

5.1.3 Participant selection and survey submission

Subsequently we needed to find a set of developers to submit forms. We decided to send Google Form mail invitation to more then 40 developers. We send form to students of Universty of Molise, with experience in Android programming, and working developers in different companies. We asked the ones who agreed to provide us with a self-evaluation of their Android app development skills on a Likert scale from 1 (novice) to 5 (expert). We did not invite developers with no experience with Android app development. Based on such information, we selected a sample of 20 developers with diverse Android experience levels. Then, we divided the participants in four groups, with equal median experience level of the developers. Each group of developers needed to complete two tasks: in one of them we provided participants with plain UML, while in the other

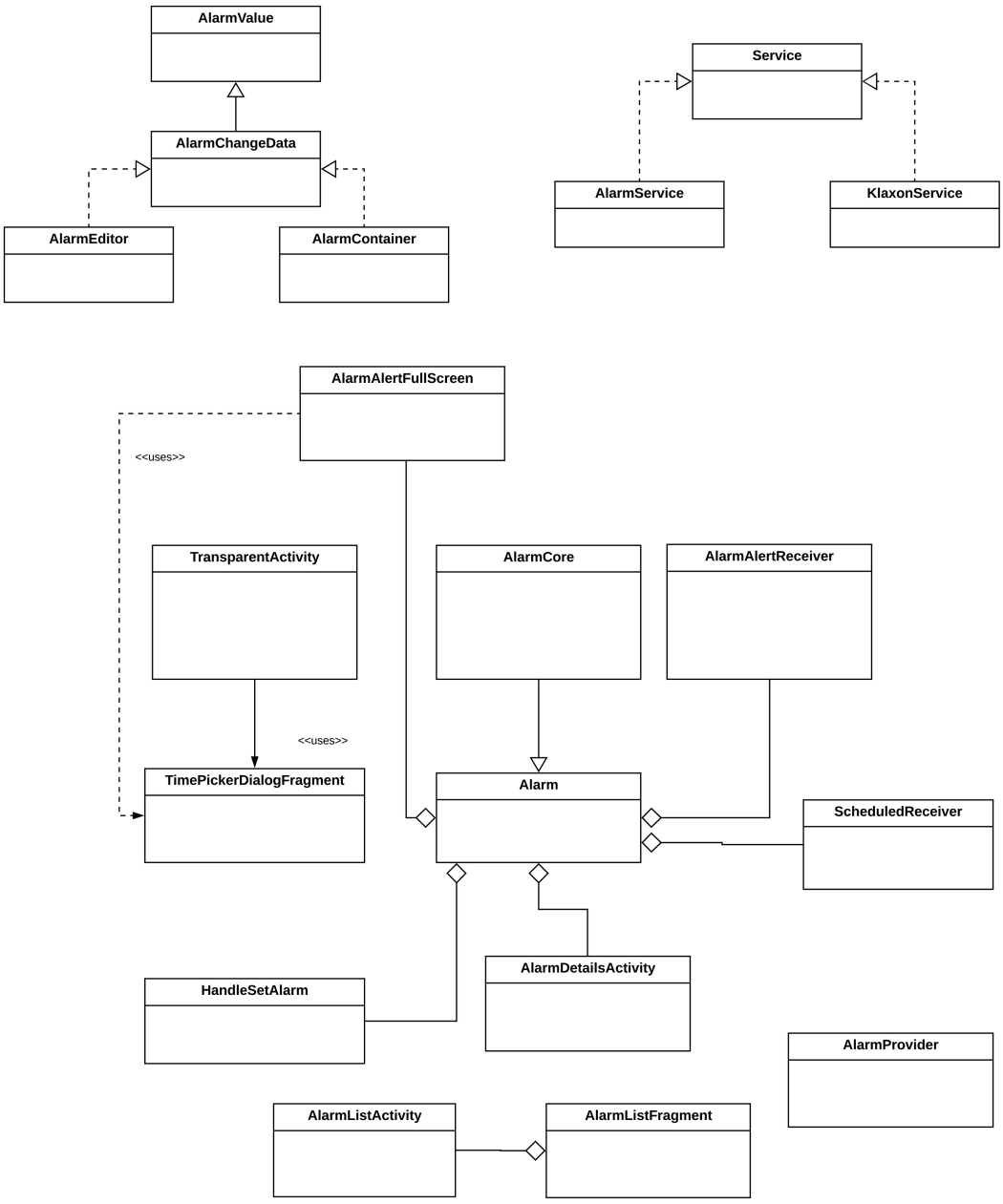

Figure 5.1: Plain UML class diagram for Simple Alarm Clock

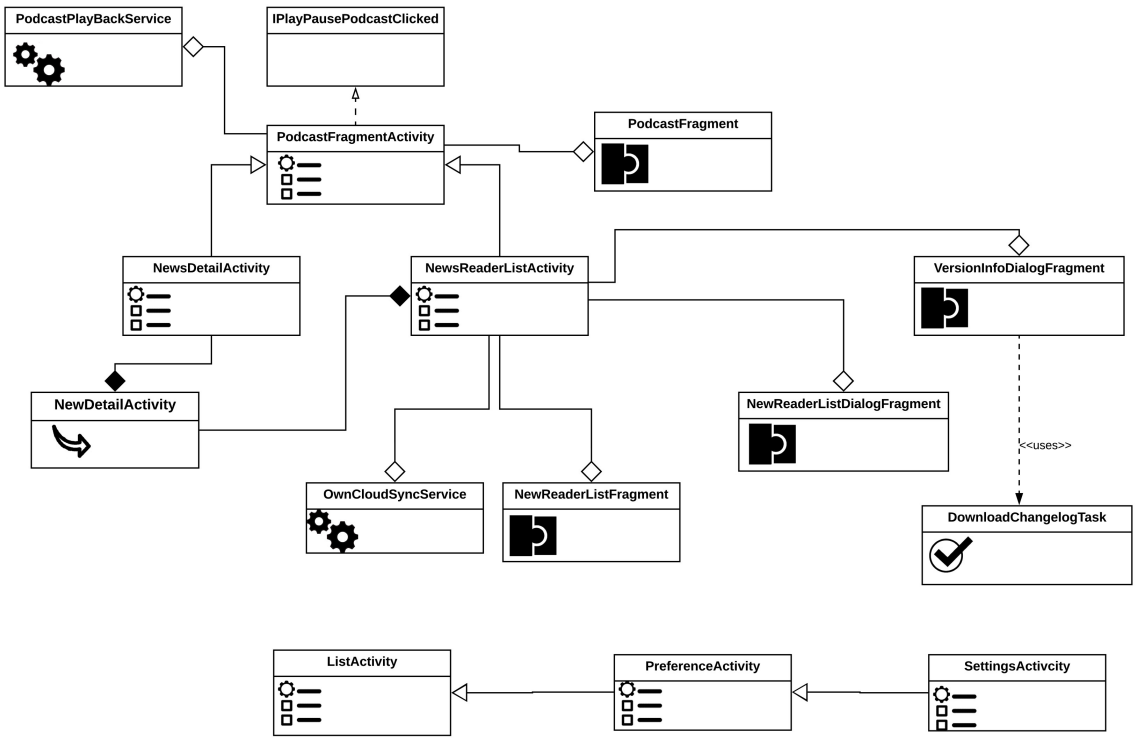

Figure 5.2: Droid UML class diagram for NextCloud News Reader

Table 5.2: Composition of the four groups

Group	Median Android skill	#of developers	# of students
1	3	4	1
2	3	3	2
3	3	3	2
4	3	3	2

one with Droid UML. In both the tasks the participants could also look at the source code. We report the composition of such groups with the task they needed to perform and the diagram they were provided with in Table 5.2. We reduced by design potential biases due to the order in which the tasks were completed: two groups were forced to complete the task using plain UML first, while the other two groups were forced to complete the task using Droid UML first.

The next step was to send Google form invitation to complete the questionnaire. The participants had to complete a pre-questionnaire, as described in section 5.1.1, before performing the tasks. In such a questionnaire we gathered data about the experience in Android, in Java, and in UML. Here we report questions in details :

- Current Job: student, phd student, developer;
- Programming general experience: less then 1 year, from 1 to 3 years, from 3 to 5 years, from 5 to 7 years, more then 7 years;
- Java programming experience:less then 1 year, from 1 to 3 years, from 3 to 5 years, from 5 to 7 years, more then 7 years;
- Java programming personal evaluation : from 1 to 5;
- Have you ever developed an Android application?
- Android programming experience:less then 1 year, from 1 to 3 years, from 3 to 5 years, from 5 to 7 years, more then 7 years;
- Android programming personal evaluation; from 1 to 5;
- UML modeling personal evaluation: from 1 to 5;

After completing the pre-questionnaire, we asked the participants to complete the two tasks in a specific order, based on the groups they belonged to. We did not ask the participants to write code to actually implement the features. Instead, we asked them to just report the classes that would

be involved in the fixing of the issue. Such an operation was performed offline. However, we asked the participants not to take breaks while they were completing a task. On the other hand, they were free to take a break between the two tasks. After each task, the developers were asked (i) in what percentage they used the model and the code, and (ii) the classes that needed to be changed to perform the maintenance task. Finally, after completing both the tasks, we asked the participants to complete a post-questionnaire. Here the questions:

- *Q3*: Which diagram did you find more useful? To what extent?
- *Q4*: How useful would be an Android-specific UML extension, in general?
- *Q5*: How simple is understanding the Droid UML notation?
- *Q6*: How useful is a graphical representation for Activity?
- *Q7*: How useful is a graphical representation for Fragment?
- *Q8*: How useful is a graphical representation for Intent?

Also in this case the participants could answer with a score between 1 and 5. All the developers were invited to read a brief guide about Droid UML before starting the experiment. We did this to allow them familiarizing with the new notation.

To answer RQ1, we first defined an oracle for both the tasks, the classes involved in the issue fixing process. To do this, we looked at the changes made by the original app developers to actually fix the issue. We defined the following oracles for the two tasks:

- **Task 1**: `TransparentActivity`;
- **Task 2**: `NewsDetailActivity`, `NewsDetailFragment`.

We used such a piece of information to compute three metrics commonly used in Information Retrieval (IR): precision, recall, and F-measure.

Precision is calculated as the classes correctly detected by the participants divided by classes to detect and recall is calculated by class correctly detected by participants divided by the number of classes defined in oracle. F-measure is computed as the harmonic mean of precision and recall.

Such metrics do not take into account the actual usage of the diagram. Any difference observed in the metrics could be unrelated to the diagram. For this reason, we first report the difference in the actual usage of both the models declared by the developers; then, we compute precision, recall, and F-measure weighted by the percentage of use of the UML model used for the task. For example, if a developer declared that she used the diagram at 75% and she achieved a recall of 50%, the weighted recall is 0.375. In this case a score of 1 means that the participant used only the diagram and that she achieved the best score for a specific metric. On the other hand, a score of 0 may mean either that the developer did not use the diagram or that the developer achieved a score of 0 for a specific metric.

To answer RQ2 we analyzed the answers to the qualitative questions we asked in our post-questionnaire. Specifically, we asked the following questions after each task:

- *Q1*: What was the complexity of the comprehension task?
- *Q2*: How useful was the UML diagram you used?

The authors could answer with a score between 1 and 5. In this case, we compared the scores achieved by plain UML with the scores achieved by Droid UML.

Table 5.3: Mean precision, recall and F-measure.

		All			T1			T2		
		Precision	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**	**Precision**	**Recall**	**F-measure**
2*Normal	Droid-UML	**0.305**	**0.277**	**0.277**	**0.111**	**0.111**	**0.111**	**0.5**	**0.44**	**0.444**
	UML	0.2	0.133	0.15	0.00	0.00	0.00	0.375	0.25	0.291
2*Weighted	Droid-UML	**0.201**	**0.166**	**0.171**	**0.083**	**0.083**	**0.083**	**0.319**	**0.25**	**0.259**
	UML	0.034	0.018	0.023	0.00	0.00	0.00	0.065	0.033	0.044

5.2 Experiment results and evaluation

We report in this section the results of our empirical study, divided by research question.

5.2.1 RQ1: Actual usefulness

Table 5.3 reports the results of the comparison. First, we observed that participants were more likely to complete the tasks when they were presented with Droid UML (Precision 0,305 and Recall 0,277 for Droid UML vs Precision 0,2 and Recall 0.133 for plain UML). We report in Figure 5.3 the distribution of the usage of the diagrams. When developers are provided with a Droid UML diagram, they are more prone using it. On the other hand, when they are provided with a classic UML diagram, they tend to focus more on the code. As for the actual performance achieved by the developers, Table 5.3 shows the mean precision, recall and F-measure achieved by the developers using both Droid UML and the baseline. First, it is worth noting that the absolute values of such metrics are low, in general. This means that the participants found the tasks quite difficult to complete. Indeed, we found that in 7 cases they could not complete the task at all, while in 23 cases they were not able to identify any of the classes involved in the fix. However, this was partially expected, since the task were not trivial and the code bases of the applications taken into account were quite big. Despite the low values, we observed some interesting results. For T1, none of the developers who used plain UML was able to identify any class involved in the fix. Droid UML, instead, was useful to complete this task for 6 developers. However, when we weight the performance of the developers with the percentage usage of the diagrams they declared, we can see that the difference is, overall, even larger.

In summary, we can conclude that developers use more Droid UML diagrams, and this also allows them to generally perform slightly better.

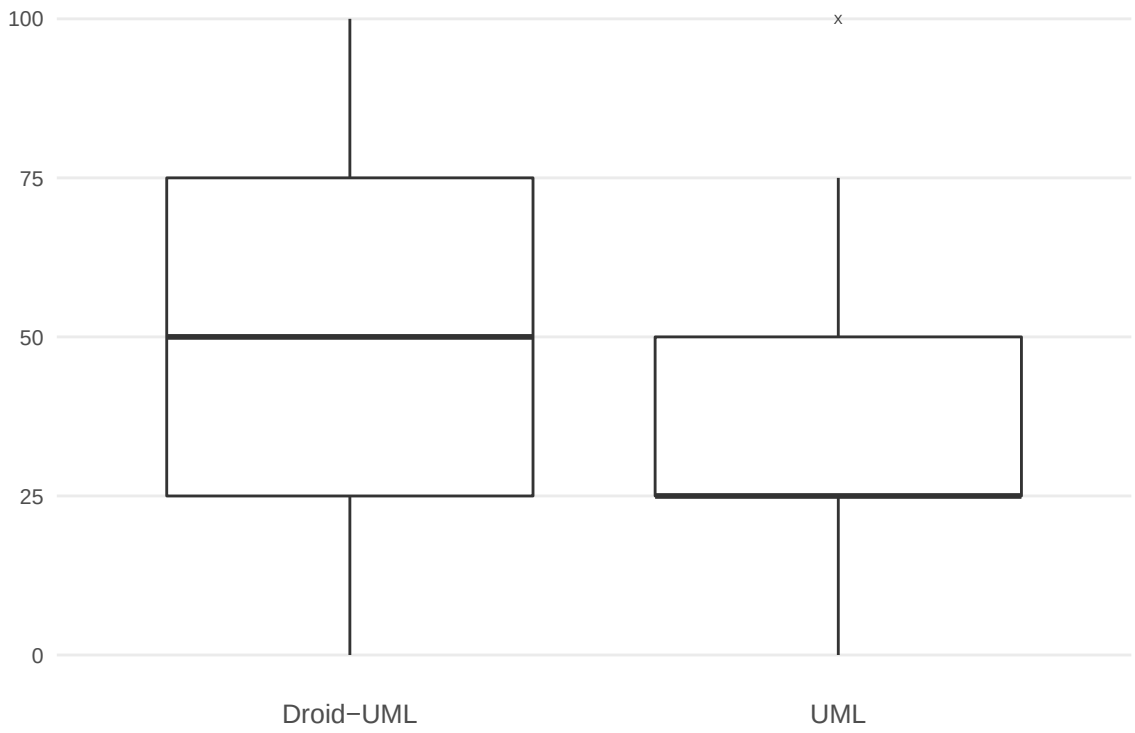

Figure 5.3: percentage usage of the diagram

5.2.2 RQ2: Perceived usefulness

Figure 5.4 reports the frequency of the responses to the question Q4. The bar plot shows that 15 participants think that Droid-UML is somewhat useful (greater than 2) and only 1 participant thinks that it is not. The median value of responses is 4. It represent a high perceived utility of Droid-UML in supporting development and maintenance of Android applications.

Figure 5.5 reports a comparison between the perceived usefulness of plain UML and Droid UML (question Q2). The plot shows us that Droid UML was perceived as more useful than plain UML. The median values computed on given responses confirm visual data contained in Figure 4.6. Although both UML diagrams result useful in development and maintenance processes, the Droid UML was perceived as more useful. Indeed, plain UML median is equal to 3 while Droid UML median is equal to 4.

To answer sub question three of RQ2 we considered question about task perceived difficulty for task with application 1 and task with application

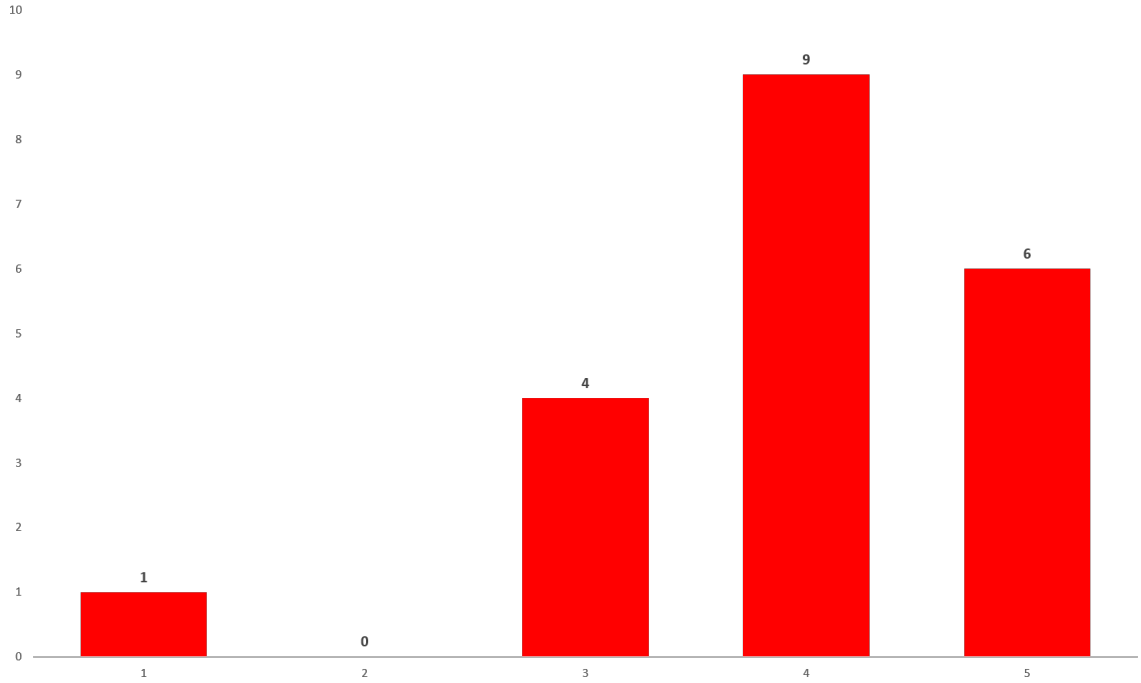

Figure 5.4: Droid-UML perceived utility

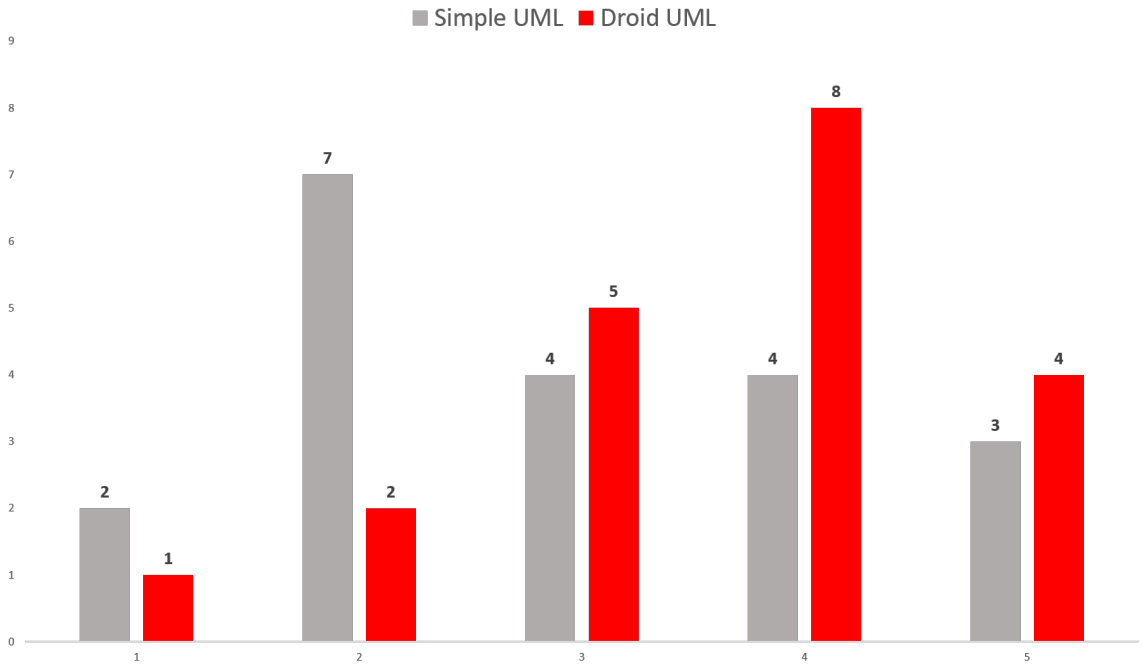

Figure 5.5: Plain vs Droid-UML perceived utility

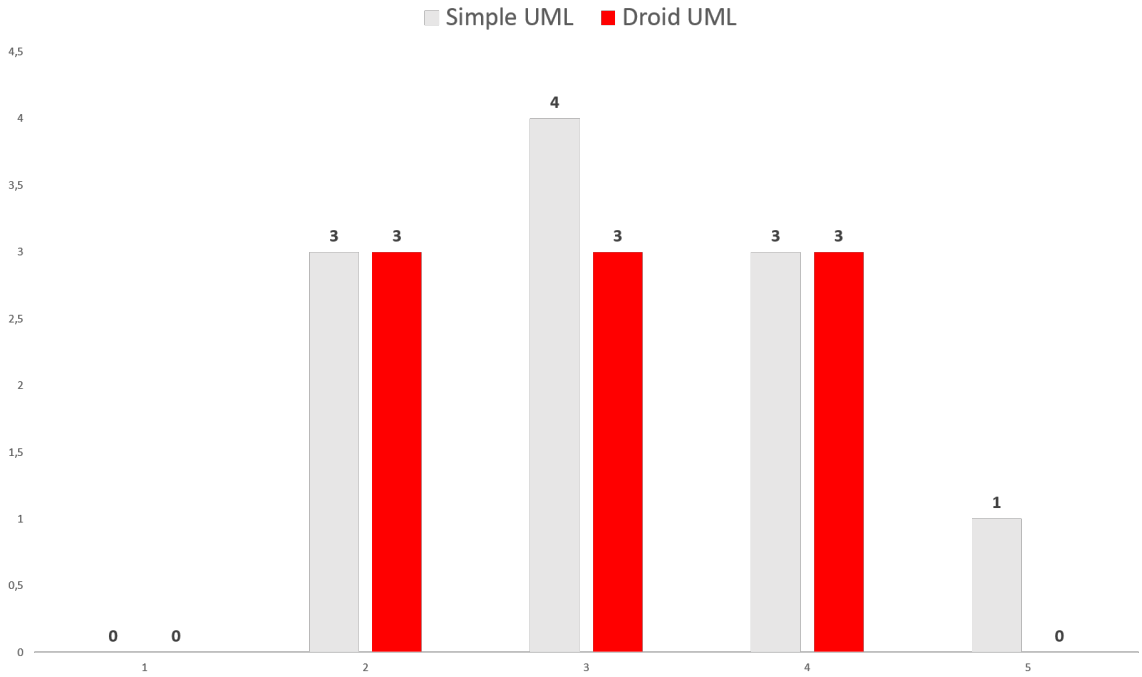

Figure 5.6: Plain vs Droid-UML task with application 1 perceived difficulty

2. We anaysed responses taking into account single task data and overall data. In particular 5.6 contains bar plot about how users found complex to understand and resolve task with application 1. The plots contain frequency of given responses and it shows us that there is a small difference between the uses of the diagrams (plain UML and Droid UML) and some users found more difficult resolve task with plain UML. If we analyse median value of the use of two diagrams we can confirm visual data. In fact the median is both 3 while the average value of the responses is a little bit higher for plain UML diagram (3,18 vs 2,85). Looking at task with application 2 the situation is different. 5.7 contains bar plots of the perceived difficulty to understand and solve task with application 2. The differences for application 2 are more evident. There are more users compared to application 1 data, that found more difficult resolve task 2 with plain UML diagram. The data is confirmed by median value (4 for plain UML and 3 for Droid UML) and by average values (3,75 for plain and 3,17 for Droid). Taking into consideration that task with application 2 is a little bit more difficult of task with application 1, we can assert that a specific Android UML extension could be more useful when developers face with more complex Android apps.

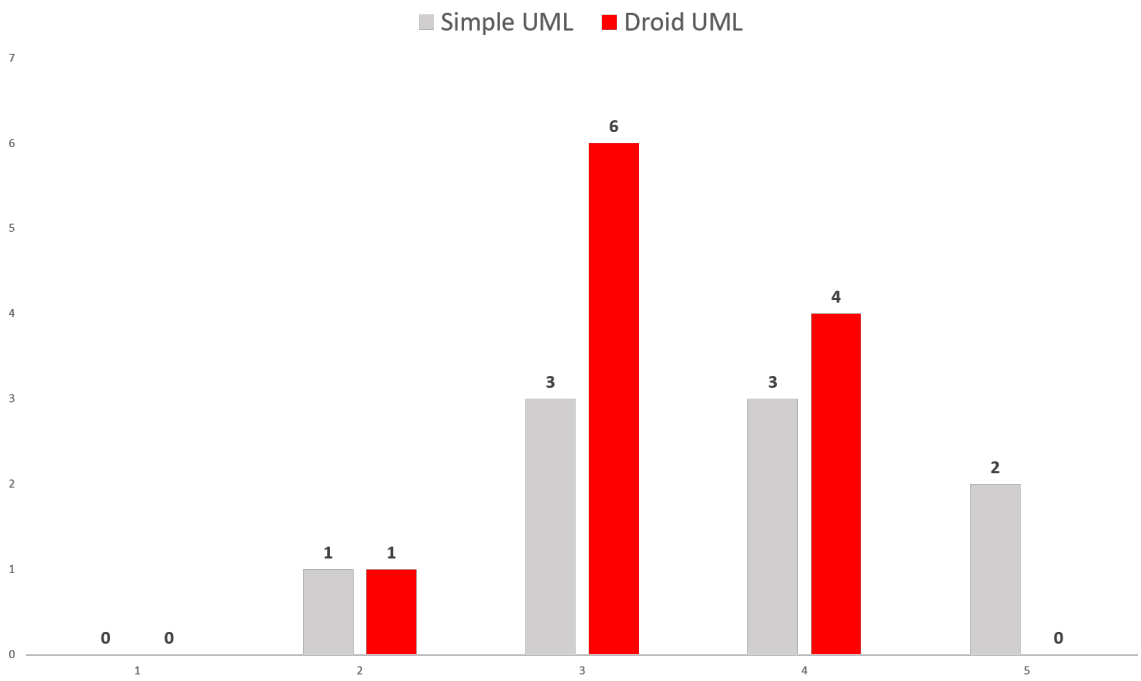

Figure 5.7: Plain vs Droid-UML task with application 2 perceived difficulty

The overall data contained in 5.8 confirm that using a specific UML extension for Android application development (Droid UML) could help in task comprehension. Even if median value is 3 for both extension the average value is higher for tasks completed using Simple UML (3,4 vs 3,15).

Figures 5.6 and 5.7 report different numbers because they refer to single tasks. Application 1 with plain UML has been submitted to 11 participants while Application 1 with Droid UML to 9 participants. For Application 2 the Droid-UML 15 participants that completed task with plain UML are 9, instead participants that completed task with Droid UML are 11.

5.2.2.1 Discussion

Our results from both RQ1 and RQ2 clearly show that Droid-UML helps the developers in achieving a better performance in the tasks they had to complete more than plain UML. While we do not achieve statistically significant results because of the size of our sample, we observe some very clear trends. We have to notice a real difficulty to retrieve developers to task completion. We send invitations to more then 40 developers but only

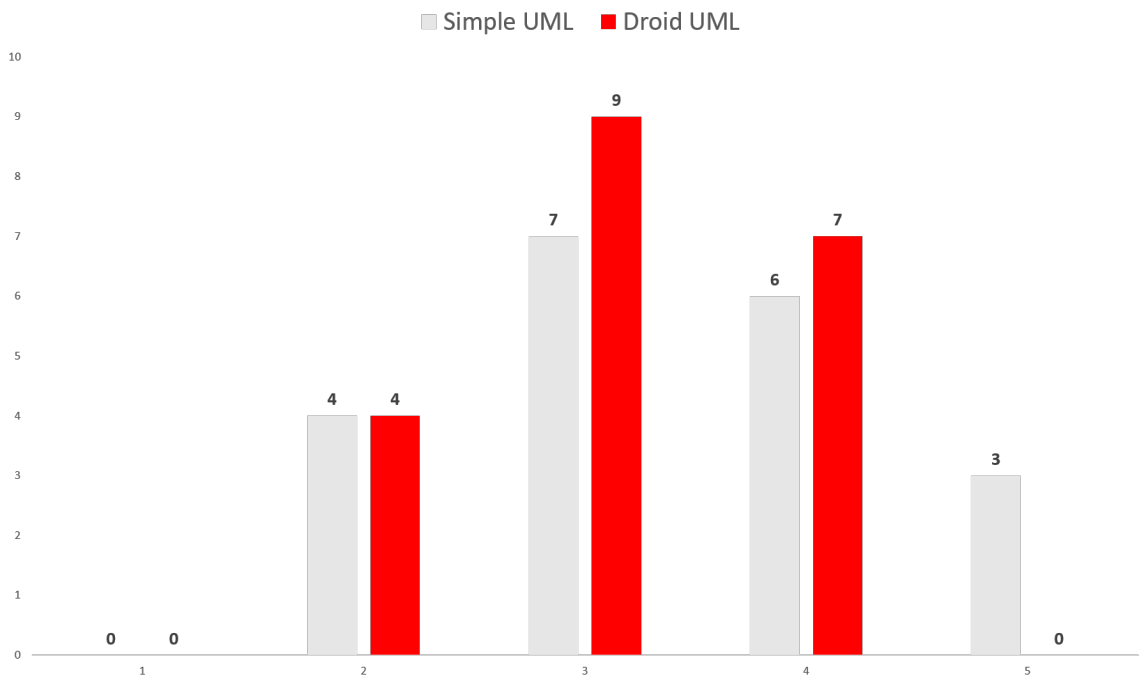

Figure 5.8: Plain vs Droid-UML overall perceived difficulty

the 50% of them completed the survey. Another relevant data was the bad task completion comprehension. We observed that different participants to experiment did not indicate any classes even if them understand the task. Anyway we focused attention on internal and external validity aspects.

5.2.2.2 Internal Validity

. Threats to internal validity concern factors related to the study design. The choice of the Android apps could have affected the results. Specifically, selecting too simple or too complex tasks may not let us observe differences between UML Droid and plain UML. The results show that the performance of the developers was relatively low in absolute terms, suggesting that the tasks were quite complex. This may have flattened the results. A potential threat could be represented by the fact that Droid UML uses icons. Developers could be naturally inclined to prefer a diagram with icons, while such a feature could provide no practical advantage. To mitigate this threat, we used objective measures of the developers performance (precision, recall and F-measure). Therefore, we

conclude that using icons for some types of classes actually helps improving the understanding.

5.2.2.3 External Validity

. Threats to external validity concern the generalization of the results. The sample of developers we took into account is rather small (e.g., 20 developers). To mitigate the potential lack of generalization, we considered developers with heterogeneous experience (variation sampling). We selected, indeed, both students and developers with different level of programming and modeling skills.

In conclusion we observed a quite big difference in the use of both the diagrams: the developers used Droid UML to complete the task more than plain UML. So we can affirm that Droid-UML helps navigating the code more than the plain UML notation.

Chapter 6

Conclusions and future works

In this work we studied the mobile context and the modeling approaches for mobile and in particular for Android applications. We started with an in-depth analysis of source code and decompiled code of Android application. We tried to understand if there was a real evolution of application directly connected to source code but we observed that, despite some differences of code analysed, there isn't a real evolution. The number of application published on Google Play Store and F-Droid store grows and grows the usage of APIs. So we focused attention on development process participation by studying commits and authors of a large set of GitHub repositories. We found interesting that there is a relevant number of projects with a significant number of contributors so we proposed an UML extension to support Android application development and maintenance processes. The idea is to extend the UML class diagram paradigm and define new stereotypes with the support of graphical icons. To better understand mobile context we studied the mobile technologies, the mobile operative systems differences and the differences between Android, iOS and cross-platform applications development techniques. To assess the usefulness of Droid UML we conducted a controlled experiment in two ways: objectively evaluating the actual performance of the participants, and subjectively, considering the perceived usefulness. The results tell us that participants are more prone to use Droid UML than plain UML. Besides, Droid UML helped them achieving better feature location performances. Also, the developers perceived Droid UML as more useful than plain UML. In summary, the results suggest that the Droid UML supports developers that face for the first time a new project, above

all if the task at hand is complex. Anyway we need to reply the study done with a larger number of developers in order to increase the statistical power of our results and understand if there is some aspect we can improve in proposed extension.

Despite proposing a context UML extension could create confusion to developers and result unnecessary, the results of the experiment done during our research seems going in the opposite direction. I tried to intercept market needs derived from a growth in terms of application realized and mobile devices usage. Apps becomes comparable to traditional software and so need more attention in terms of modeling and maintenance. While the software changes, it will be necessary introduce new methodologies and new concept in software design too. Droid UML wants to represent a preliminary approach to better support design and maintenance activities and better understand so complex Android applications. Considering results of the controlled experiment, despite a quite personal satisfaction for the work done, it is necessary to evidence some needed improvements. Droid UML represents a small piece of the big world of Software Engineering that needs to be improved and that has to continuously follow development concepts changes. For example we could try to propose an extension concerning architectural components as well as dynamic components of UML. At the same time it is necessary refine the controlled experiment, changing some question and considering a larger number of developers and better understand so the developers perception.

In future, following this approach, we will try to extend UML components to model dynamic aspect of applications. The idea is to propose extension for UML Sequence Diagram and give so more support in application development and maintenance processes. Android applications are composed by static and behavioural aspects. Using sequence diagram to model behavioural aspects as proposed by Parada et al. [45] and adding graphical stereotype to represent iteration, conditional, and message exchanges, as usual for traditional object-oriented software could help developers in development phases. Another idea to better support developers is the development of a tool for automatic Droid UML class diagram generation. The approach could be realized as a plugin for Android Studio the most used tool for Android application development. The last work to do

concerns proposing an extension to support cross platform applications. Cross platform development is becoming even more widespread so, study cross platform development application and propose UML extension could be really interesting.

Appendix A

Appendix - Evaluation survey

Android e UML

Grazie per aver scelto di partecipare allo studio.

Il questionario è diviso in tre sezioni. Nella prima sezione ti verrà chiesto di inserire informazioni relative al tuo background. Nella seconda e nella terza, invece, ti verrà chiesto di completare due task di comprensione su diverse applicazioni Android e, successivamente, di rispondere ad alcune domande su questi meccanismi.

In entrambi i casi, sarai supportato da diagrammi che descrivono le applicazioni che implementano i meccanismi che dovrai comprendere. In uno dei due task ti sarà proposto di usare un diagramma UML classico; nell'altro, invece, avrai a disposizione un diagramma UML esteso.

Ogni task richiede circa 30 minuti. Ti chiederemo di specificare l'orario di inizio e di fine di ogni task quanto ti verrà chiesto. Puoi fare una pausa tra i due task, ma ti chiediamo, se possibile, di non fare pause durante lo svolgimento del task. Se hai dovuto fare pause, ti chiediamo di specificarlo nell'apposito campo alla fine del task.

Rispondere bene e/o velocemente non ha nessun riscontro pratico. Per questo, ti chiediamo di rispondere nel modo più sincero possibile a tutte le domande e di indicare in maniera accurata gli orari di inizio e fine dei task. Ti chiediamo, infine, di limitare al minimo l'utilizzo di risorse esterne (es: Google, GitHub, StackOverflow).

* Required

1. **Email address** *

2. **Occupazione attuale** *
   Mark only one oval.
   - ◯ Studente laurea triennale
   - ◯ Studente laurea magistrale
   - ◯ Dottorando
   - ◯ Sviluppatore occupato

3. **Anni di esperienza in programmazione (incluso corsi universitari)** *
   Mark only one oval.
   - ◯ Meno di 1 anno
   - ◯ Tra 1 e 3 anni
   - ◯ Tra 3 e 5 anni
   - ◯ Tra 5 e 7 anni
   - ◯ Più di 7 anni

4. **Anni di esperienza in programmazione Java (incluso corsi universitari)** *
   Mark only one oval.
   - ◯ Meno di 1 anno
   - ◯ Tra 1 e 3 anni
   - ◯ Tra 3 e 5 anni
   - ◯ Tra 5 e 7 anni
   - ◯ Più di 7 anni

5. **Valuta la tua esperienza in programmazione Java su una scala da 1 (molto bassa) a 5 (molto alta)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

6. **Hai mai sviluppato o fatto manutazione di app per Android (anche nell'ambito di corsi universitari)** *
Mark only one oval.

- ◯ Si
- ◯ No

7. **Anni di esperienza in programmazione Android (incluso corsi universitari)** *
Mark only one oval.

- ◯ Meno di 1 anno
- ◯ Tra 1 e 3 anni
- ◯ Tra 3 e 5 anni
- ◯ Tra 5 e 7 anni
- ◯ Più di 7 anni

8. **Valuta la tua esperienza in programmazione Android su una scala da 1 (molto bassa) a 5 (molto alta)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

9. **Valuta la tua esperienza nell'utilizzo della notazione UML su una scala da 1 (molto bassa) a 5 (molto alta)** *
Mark only one oval.

1	2	3	4	5
◯	◯	◯	◯	◯

Inizio primo task

Questa sezione è relativa al primo task da svolgere. Puoi fare una pausa prima di iniziare. Appena sei pronto, inserisci l'orario e procedi con il questionario per i dettagli sul primo task.

10. **Inserisci l'orario di inizio del task** *

Example: 8:30 AM

Primo task

Materiale

Scarica il codice dell'applicazione a questo link:
https://drive.google.com/open?id=18KmWN7gs8cctauXmh4Q0GbN1UBh3LWaL

Scarica il diagramma dell'applicazione a questo link:
https://drive.google.com/open?id=1Kdi-Sri7NGhrfsIuwNsYpbACQ1ofy7nP

App

L'applicazione News Reader di NextCloud consente la sincronizzazione di feed tra Android e l'app News di Nextcloud/ownCloud. Tra le varie funzionalità l'app consente di leggere le news offline, visualizzare l'elenco delle news e personalizzarlo, cambiare tema, regolare la grandezza del carattere e molto altro.

Task

Per dare la possibilità agli utenti di aprire i link in un browser esterno e migliorare la sicurezza dell'app specificando così quale link si sta aprendo bisogna apportare delle modifiche a quali file dell'applicazione? Che tipo di componenti Android devono essere modificati?

Appena hai completato il task o se non riesci a completarlo in un tempo ragionevole, procedi con il questionario.

Fine primo task

11. **Inserisci l'orario di fine del task** *

Example: 8:30 AM

12. **Quanti minuti di pausa hai fatto durante il task?** *

Valutazione primo task

In questa sezione ti verrà chiesto di valutare l'attività svolta durante il primo task.

13. **Quanto hai trovato difficile comprendere il meccanismo?** *

Mark only one oval.

	1	2	3	4	5	
Molto facile	◯	◯	◯	◯	◯	Molto difficile

14. **Hai capito pienamente come è implementato il meccanismo che dovevi comprendere?** *

Mark only one oval.

- ◯ Sì
- ◯ No

15. **Se sì, descrivi brevemente, a parole, il meccanismo che hai dovuto comprendere** *

16. **Quanto ti è stato utile il diagramma che hai usato?**
Mark only one oval.

	1	2	3	4	5	
Poco utile	◯	◯	◯	◯	◯	Molto utile

17. **In che percentuale hai utilizzato codice sorgente e diagramma forniti?**
Mark only one oval.

- ◯ 100% codice (non ho usato il diagramma)
- ◯ 75% codice, 25% diagramma
- ◯ 50% codice, 50% diagramma
- ◯ 25% codice, 75% diagramma
- ◯ 100% diagramma (non ho usato il codice sorgente)

Inizio secondo task

Questa sezione è relativa al secondo task da svolgere. Puoi fare una pausa prima di iniziare. Appena sei pronto, inserisci l'orario e procedi con il questionario per i dettagli sul secondo task.

18. **Inserisci l'ora di inizio del task** *

Example: 8:30 AM

Secondo task

Materiale

Scarica il codice dell'applicazione a questo link:
https://drive.google.com/open?id=19_A4CQbKrc_UbtfHL6kjDtkncVL7CgmI

Scarica il diagramma dell'applicazione a questo link:
https://drive.google.com/open?id=1dFxaiRWPaJu356ZYW4K7TL5PLpf9Ij_0
Leggi la spiegazione del diagramma:
https://docs.google.com/document/d/e/2PACX-1vRROHNglp-EXM-U3eV3e67XN9LxlDBJR6ri9tDQ_z1hoJB2v9oVvwLGyxasB8datWx-w82xOh0nqsLt/pub

App

Simple Alarm Clock è una sveglia per smartphone e tablet Android che trasmette l'esperienza di una semplice sveglia combinando funzionalità potenti con interfaccia semplice. L'interfaccia della sveglia è disegnata per essere semplice, intuitiva ed efficiente.

Task

Individuare le activities da modificare nel caso in cui si voglia aggiungere un menu contestuale (es. Dismetti, Riprogramma sveglia) per le notifiche posticipate di un allarme.

Appena hai completato il task o se non riesci a completarlo in un tempo ragionevole, procedi con il questionario.

Fine secondo task

19. **Inserisci l'ora di fine del task** *

Example: 8:30 AM

20. **Quanti minuti di pausa hai fatto durante il task?** *

Valutazione secondo task

In questa sezione ti verrà chiesto di valutare l'attività svolta durante il secondo task.

21. **Quanto hai trovato difficile comprendere il meccanismo?** *

Mark only one oval.

	1	2	3	4	5	
Molto facile	◯	◯	◯	◯	◯	Molto difficile

22. **Hai capito pienamente come è implementato il meccanismo che dovevi comprendere?** *

Mark only one oval.

- ◯ Sì
- ◯ No

23. **Se sì, descrivi brevemente, a parole, il meccanismo che hai dovuto comprendere** *

24. **Quanto ti è stato utile il diagramma che hai usato?**

Mark only one oval.

	1	2	3	4	5	
Poco utile	◯	◯	◯	◯	◯	Molto utile

25. **In che percentuale hai utilizzato codice sorgente e diagramma forniti?**

Mark only one oval.

- ◯ 100% codice (non ho usato il diagramma)
- ◯ 75% codice, 25% diagramma
- ◯ 50% codice, 50% diagramma
- ◯ 25% codice, 75% diagramma
- ◯ 100% diagramma (non ho usato il codice sorgente)

Valutazione complessiva

Ti chiediamo di rispondere a queste domande basandoti su quello che hai potuto vedere svolgendo i due task.

26. **Quale dei due diagrammi hai trovato più utile?** *
Mark only one oval.

- UML classico
- UML esteso
- Ugualmente utili

27. **In che misura pensi che il diagramma che hai scelto nella precedente domanda sia stato più utile dell'altro?** *
Mark only one oval.

	1	2	3	4	5	
Poco più utile						Molto più utile

Valutazione qualitativa

Ti chiediamo di rispondere a queste ultime domande sulle notazioni UML che hai usato, cercando di pensare alla loro utilità in generale e, quindi, non limitandoti all'utilità che hanno dimostrato nei task che hai svolto.

28. **Quanto pensi possa essere utile, in generale, un'estensione di UML specifica per Android?** *
Mark only one oval.

	1	2	3	4	5	
Poco utile						Molto utile

29. **Quanto pensi che sia facile capire la notazione UML estesa, conoscendo la notazione classica?** *
Mark only one oval.

	1	2	3	4	5	
Molto difficile						Molto facile

30. **Quanto pensi sia utile lo stereotipo che rappresenta la componente Activity?** *
Mark only one oval.

	1	2	3	4	5	
Poco utile						Molto utile

31. **Quanto pensi sia utile lo stereotipo che rappresenta la componente Fragment Adapter** *
Mark only one oval.

	1	2	3	4	5	
Poco utile						Molto utile

32. **Quanto pensi sia utile lo stereotipo che rappresenta la componente Intent?** *
Mark only one oval.

	1	2	3	4	5	
Poco utile	◯	◯	◯	◯	◯	Molto utile

33. **Quali informazioni aggiuntive pensi che possano migliorare l'estensione UML?**

Hai quasi terminato...

Clicca su "Submit"/"Invia" per registrare la tua risposta. Ti ringraziamo per aver partecipato!

Appendix B

Droid and plain UML diagrams

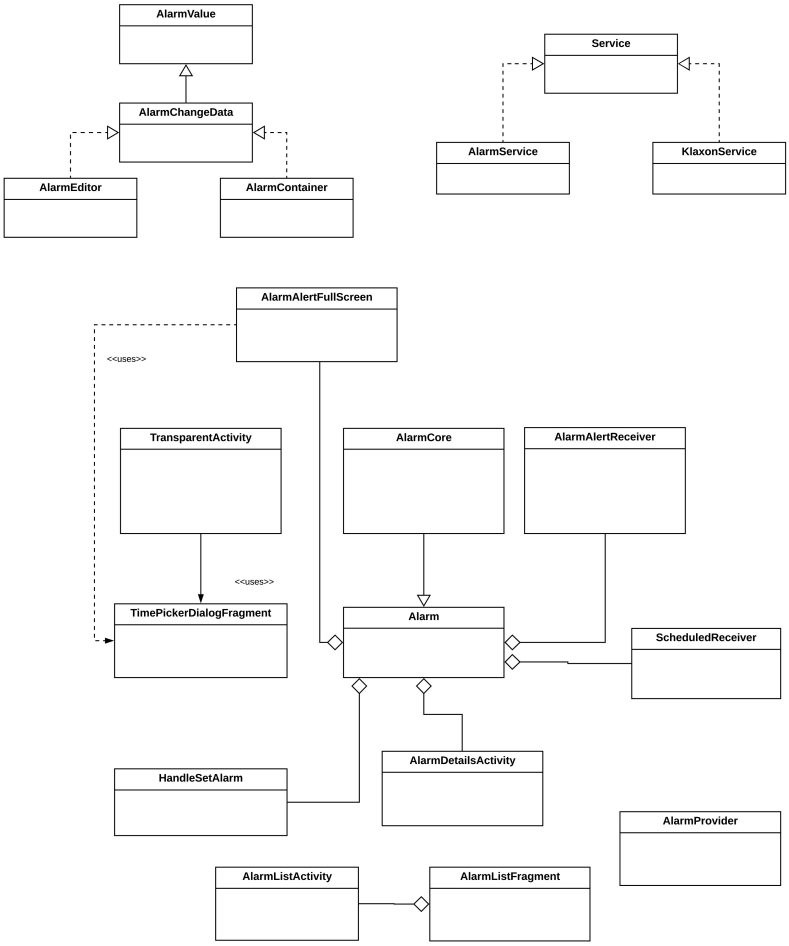

Figure B.1: Simple Alarm Clock - Plain UML diagram

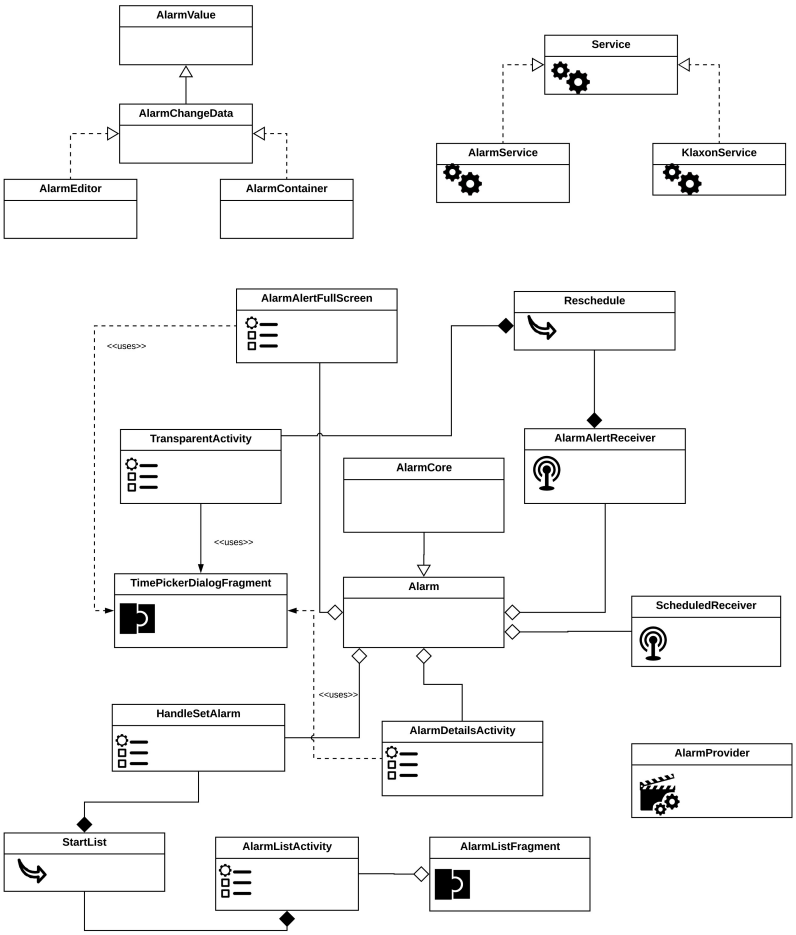

Figure B.2: Simple Alarm Clock - Droid UML diagram

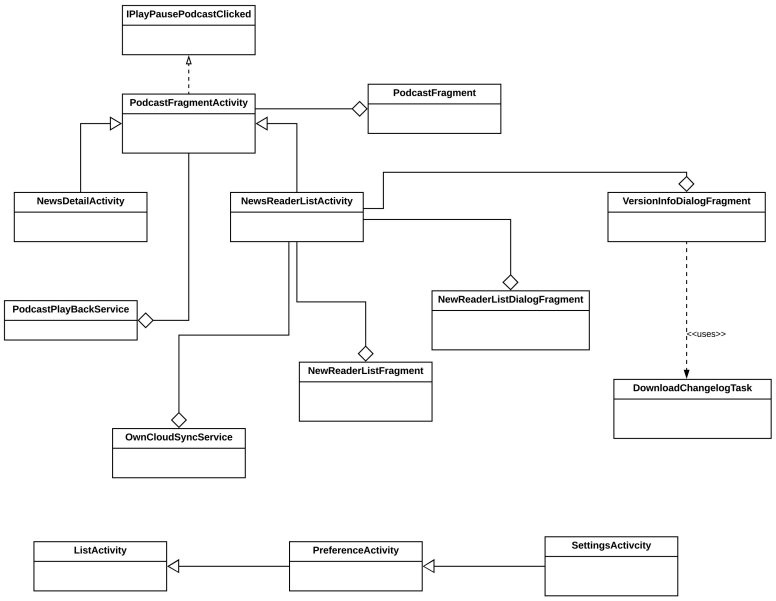

Figure B.3: NextCloud News Reader - Plain UML diagram

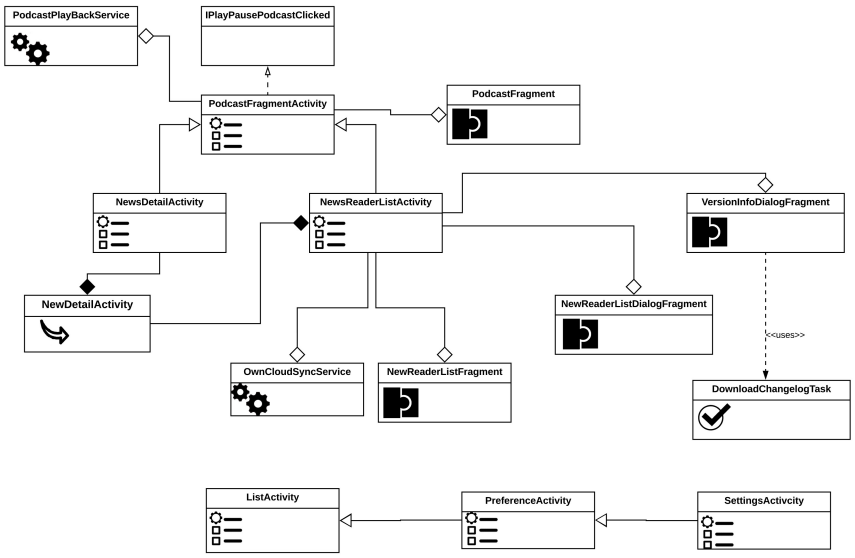

Figure B.4: NextCloud News Reader- Droid UML diagram

Bibliography

[1] About objective-c.

[2] About swift.

[3] Android application fundamentals.

[4] Android versions: A living history from 1.0 to pie.

[5] App programming guide for ios.

[6] Class diagram.

[7] cross-platform mobile development.

[8] The evolution of mobile apps.

[9] The evolution of native mobile app development.

[10] Gitwrapper library.

[11] Google play crawler java api.

[12] The history of ios, from version 1.0 to 11.0.

[13] ios version history.

[14] Mobile first: What does it mean.

[15] Mobile operating system.

[16] Mobile operating system (mobile os).

[17] Mobile operating systems (mobile os) explained.

[18] Nextcloud news reader.

[19] Number of available applications in the google play store from december 2009 to september 2018.

[20] Simple alarm clock.

[21] Sonarjava.

[22] Top 20 tools for android development.

[23] Tracing the history and evolution of mobile apps.

[24] Vision impairment and blindness.

[25] What is uml.

[26] Xcode.

[27] Hubert Baumeister, Nora Koch, and Luis Mandel. Towards a uml extension for hypermedia design. In *UML*, 1999.

[28] Gabriele Bavota, Carmine Gravino, Rocco Oliveto, Andrea De Lucia, Genoveffa Tortora, Marcela Genero, and José A Cruz-Lemus. A fine-grained analysis of the support provided by uml class diagrams and er diagrams during data model maintenance. *Software & Systems Modeling*, 14(1):287–306, 2015.

[29] G. Botturi, E. Ebeid, F. Fummi, and D. Quaglia. Model-driven design for the development of multi-platform smartphone applications, Sept 2013.

[30] Lionel C Briand, Yvan Labiche, Massimiliano Di Penta, and Han Yan-Bondoc. An experimental investigation of formality in uml-based development. *IEEE Transactions on Software Engineering*, 31(10):833–849, 2005.

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

[32] Jim Conallen. *Building Web applications with UML*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[33] Wojciech J. Dzidek, Erik Arisholm, and Lionel C. Briand. A realistic empirical evaluation of the costs and benefits of uml in software maintenance. *IEEE Transactions on Software Engineering*, 34:407–432, 2008.

[34] Harleen K Flora, Swati V Chande, and Xiaofeng Wang. Adopting an agile approach for the development of mobile applications. *International Journal of Computer Applications*, 94(17), 2014.

[35] F. Freitas and P. H. M. Maia. Justmodeling: An mde approach to develop android business applications. In *2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC)*, pages 48–55, Nov 2016.

[36] Stefania Pezzetti Gregorio Perego. Un approccio model-driven per lo sviluppo di applicazioni mobili native. Master's thesis, Politecnico di Milano, 2013.

[37] Jan Jürjens. Umlsec: Extending uml for secure systems development. In *International Conference on The Unified Modeling Language*, pages 412–425. Springer, 2002.

[38] M. Ko, Y. Seo, B. Min, S. Kuk, and H. S. Kim. Extending uml meta-model for android application. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 669–674, 2012.

[39] Nora Koch, Hubert Baumeister, Rolf Hennicker, and Luis Mandel. Extending uml to model navigation and presentation in web applications. In *Proceedings of Modelling Web Applications in the UML Workshop*. York, England, 2000.

[40] Ludwik Kuzniarz, Miroslaw Staron, and Claes Wohlin. An empirical study on using stereotypes to improve understanding of uml models. In *Program Comprehension, 2004. Proceedings. 12th IEEE International Workshop on*, pages 14–23. Citeseer, 2004.

[41] Bup-Ki Min, Minhyuk Ko, Yongjin Seo, Seunghak Kuk, and Hyeon Soo Kim. A uml metamodel for smart device application modeling based on windows phone 7 platform. In *TENCON 2011 - 2011 IEEE Region 10 Conference*, pages 201–205, Nov 2011.

[42] M. Nassar. Vuml: a viewpoint oriented uml extension. In *18th IEEE International Conference on Automated Software Engineering, 2003. Proceedings.*, pages 373–376, Oct 2003.

[43] Manuel Palmieri, Inderjeet Singh, and Antonio Cicchetti. Comparison of cross-platform mobile development tools. In *Intelligence in Next Generation Networks (ICIN), 2012 16th International Conference on*, pages 179–186. IEEE, 2012.

[44] Fabio Palomba, Dario Di Nucci, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. Lightweight detection of android-specific code smells: The adoctor project. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 487–491. IEEE, 2017.

[45] Abilio G Parada and Lisane B De Brisolara. A model driven approach for android applications development. In *Computing System Engineering (SBESC), 2012 Brazilian Symposium on*, pages 192–197. IEEE, 2012.

[46] A. Sabraoui, M. E. Koutbi, and I. Khriss. Gui code generation for android applications using a mda approach. In *2012 IEEE International Conference on Complex Systems (ICCS)*, pages 1–6, Nov 2012.

[47] Safdar Aqeel Safdar, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. Empirical evaluation of uml modeling tools–a controlled experiment. In *European Conference on Modelling Foundations and Applications*, pages 33–44. Springer, 2015.

[48] Graziano E. Tufano A., Valente R. and Matarazzo M. Tech & knowledge based economy: how mobile technologies influences the economics of small and medium activities. In *Integrated Economy and Society: Diversity, Creativity, and Technology*, pages 847–852. International School for Social and Business Studies, Slovenia, 2018.

[49] M. Usman, M. Z. Iqbal, and M. U. Khan. A model-driven approach to generate mobile applications for multiple platforms, Dec 2014.

[50] Aida Atef Zakaria, Hoda Hosny, and Amir Zeid. A uml extension for modeling aspect-oriented systems. In *International Workshop on Aspect-Oriented Modeling with UML, Germany*, 2002.