The Design and Implementation of an Android Studio Plugin to Support Task Analysis Annotation for Automated Usability Evaluation

Master Thesis

Rafael Duran Saez

Madrid, December 2017
This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering.

Master Thesis
Master Universitario en Ingeniería del Software – European Master in Software Engineering
Thesis Title: The Design and Implementation of an Android Studio Plugin to Support Task Analysis Annotation for Automated Usability Evaluation
Thesis no: EMSE-2017-12
December 2017

Author: Rafael Duran Saez
B.Sc. in Computer Engineering
Universitat de Lleida (UdL)

Supervisor:
Xavier Ferré Grau
PhD in Computer Science
Universidad Politécnica de Madrid
Languages, Computer Systems and Software Engineering Department (DLSIIS)
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Co-supervisor:
Qin Liu
PhD in Computer Science
Tongji University
Software Engineering
School of Software Engineering
Tongji University

ETSI Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain
Acknowledgements

It could be said louder but not clearer, that this work could have not been done without the support and participation of many people during the performance of the thesis.

First, I want to enormous and honestly thank my thesis tutor Xavier Ferré, who has been continuously helping me via advices, suggestions, feedback and corrections not only for this work, but also for the presentation of the related paper [1] and other academic aspects.

Thanks to Fion Yang and Meihui Li for taking charge of my condition at Tongji University and collaborating on making successful my stay here.

To my friends, old and new ones, to make my life easier and happier providing me this unforgettable stage of my life.

And finally, but not less important, to my family. For being the ones that made my experience in China possible as well as being motivating me to work with courage and dedication for the performance of my master degree.
ABSTRACT

This master thesis proposal aims to contribute to the overall research objective of how to automatically or semi-automatically analyze the usability of mobile applications in order to inform the decision process of what to pursue in each iteration of an iterative development process, in an integrated manner with IDE tools typically used to develop this kind of applications.

And, more specifically, the thesis work will focus on proving the feasibility of extending an IDE with usability annotation features in Android mobile applications, so that usage data is collected for further analysis.

The purpose and significance of the study are as follows:
- Identification and definition of a model of tasks that users are expected to carry out with the app.
- Instrumentation in the app code of data gathering functionality for the modeled tasks.
- Elaboration of an automated usability evaluation method based on the UCD approach from the HCI field.

These aim points will conclude in a more efficient and cost-effective way to apply automated usability evaluations in order to discover usability problems from user usage, to be addressed in future versions.

Usability is a key quality attribute for mobile applications, since the market is highly competitive. The current usability evaluation techniques from the Human-Computer Interaction (HCI) field are really costly, in terms of budget and time, and hard to implement because of the wide range of different contexts of use of mobile apps that nowadays exists, and others that are ready to come. They also imply a high cost of resources for full-scale usability testing.

Mobile app development demands to spend fewer resources on its operations. Automated usability evaluation techniques are an option for managing usability in this demanding environment, automating the three main phases for all usability evaluation: capture, analysis and critique of usage data.
Traditional usability testing takes a lot of resources and it can be supplemented with automatic solutions. Automated usability evaluation under real-life conditions would be focused only on data taken from user’s interaction with the app to be analyzed.

In the software engineering world, usability is a quality attribute that can be defined as the degree to which a software product can be used with effectiveness, efficiency and satisfaction by users with different characteristics. Usability can be decomposed in the following characteristics: learnability, which regards to how easily the user is able to learn how to use one software object the first time the user uses it; efficiency, how fast the users are able to accomplish one specific task; memorability, how easy is for users to efficiently use a software object after a considerable period of time without using it; errors, the amount of errors a user makes when using a software object and also the severity of them; and satisfaction, how pleased and glad the user feels after using the software object. Another related and relevant concept is the user experience (UX), which reflects the user feelings and sensations after using a product or service.

Usability has been a fundamental concept for Interaction Design research and practice, since the dawn of Human-Computer Interaction (HCI) as an inter-disciplinary endeavor. For some, it was and remains HCI’s core concept. HCI is the discipline that aims to develop interactive systems with a high level of usability.

Usability evaluation is a cornerstone of the User-Centered Design (UCD), which is a development approach to achieve a good level of usability in software systems. In particular, according to ISO 9241-210, feedback from users during operational use identifies long-term issues and provides input for future design. Usability testing by a set of representative users is the most relevant technique for usability evaluation. Assessing usability under real-user conditions is one of the hardest challenges the experts in usability evaluation for mobile apps are facing with, due to the changing context of use.

Usability evaluation methods could be the more important methods between the diverse HCI techniques that nowadays exist for attaining usable products. These ones aim to gather data from user actions that can show how users are actually using the system and which problems they face, and comparing the real usage patterns with the expected and optimal ones, to identify usability troubles and be able to solve them for feature releases. Usability evaluation processes imply different activities depending on the methods employed and the features of the procedures, but three main activities are common for all of them: capture, regarding the collection of usage data from users’ activity; analysis, the interpretation of usage data in order to identify usability problems;
and critique, the possible ways to solve and improve the problems found and finally remove them.

The challenge of developing more usable applications has led to the emergence of a variety of methods, techniques, and tools with which to address usability issues. In 2001 Ivory and Hearst studied the state of the art in automated methods for evaluating usability in their influential paper "The state of the art in automating usability evaluation of user interfaces", classifying automated usability evaluation techniques into a taxonomy. From 2001 to present interactive systems have evolved greatly, and with them the potential to automate usability evaluation, for example, the ability to collect and analyze large amounts of data. The continuous delivery paradigm, which is nowadays a feature of mobile app development, naturally fits in with the aim of evaluating usability continuously in order to inform design decisions that are taken to refine design in subsequent deliveries. User interaction in mobile apps involves many more aspects than just navigation issues, since every mobile platform offers a different UX.

A huge amount of resources is consumed when users are selected to carry out field studies. Automated usability evaluation is a potentially good alternative, offering the possibility of gathering information from actual app usage. In particular, data capture for automated usability evaluation involves using software that automatically records usability data. Google Analytics for Mobile Apps (GAMA) offers a cloud service for keeping track of user actions in mobile apps. This tool was designed for marketing purposes, but is also used to measure UX characteristics. Automated solutions provide a manageable solution for the problem of changing contexts of use and user diversity. In addition, we can use data mining techniques to perform the analysis of the gathered data and provide the results to usability experts for future critique.

The main issue is that none of the existing methods offers a comprehensive solution to extend mobile analytics to support automated usability evaluation, including a strong HCI basis that ensures the success of the usability related project goals. We propose to instrument the app code with calls to the Google Analytics logging service and apply data mining to usage log analysis in order to identify possible usability problems. The proposed approach is based on user and task analysis. This is the basis for instrumenting the user interface (UI).

Nowadays, as we have mentioned in the previous section, there is no toolkit, framework or theoretical model able to, without depending on the application domain, log data from an app taking into account the user actions, analyze the gathered information and critique the results to improve them for feature releases.
Our proposal does not preclude the application of alternative usability techniques, such as heuristic evaluation, cognitive walkthroughs or formal usability testing. The challenge that usability and UX poses in any mobile app development effort calls for the application of complementary usability techniques throughout the development process. As mobile apps and usability goals are variable, the aimed method needs to be applied to a variety of mobile app development projects, in order to customize the usability tracking approach for different domains.

Despite the existence of some initial attempts to automatize usability evaluation of mobile applications, none of them fully integrates the definition of usability-relevant user tasks with the logging of usage data that can be later analyzed with data mining techniques to uncover usability problems.

Our proposed solution will address both the theoretical approach to model user tasks with the aim of instrumenting application code for data logging the user events when undertaking such tasks, and the practical integration of such activities into a widely used IDE (Integrated Development Environment) to facilitate the adoption by software development organizations. In particular, by designing and implementing a plugin for Android Studio IDE in charge of carrying out the approach objectives.

Thus, the resulting overall framework and tool will be the first solution to the problem of introducing usability evaluation in mobile application development with a continuous delivery paradigm, in a cost-effective way.

The application of the proposed approach to an industry case study will offer a feasibility prove that will make the research results stand out compared with other partial solutions present in the current literature, with no experimental background to support them.

In order to achieve our goal, we must overcome the following possible difficulties:

The HCI field has a variety of methods offered for modeling user tasks, mainly used for manual assessment of user behavior by usability experts. A thorough analysis of this variety of methods will allow us to identify the most appropriate method for the purpose of the proposed research work: to serve as basis for the instrumentation of the application code to log user events in the undertaking of the modeled tasks.

The identification of the specific user events to log in order to evaluate user behavior when undertaking tasks is not straightforward. The research team where this work will be carried out has already explored this issue and instrumented an application as case study. The results of such case study will be taken as basis for solving this problem in the proposed Master Thesis, generalizing them for any kind of mobile application.
The expertise in usability by the Software Engineering Research Lab at UPM will provide the necessary background to ensure that usability is adequately integrated with the IDE.

As a result of the proposed research work, we will obtain an automated usability evaluation method based on data mining for keeping track of actual usage of mobile applications. It will show how GAMA can be applied to identify possible usability problems experienced by app users. The solution will be focused on automated usability evaluation of usage under real-life conditions due to the changing contexts in which mobile apps are regularly used. The case study of application will prove the feasibility of the approach, suggesting future lines of research in the issue.

**Key Words:** task analysis, automated usability evaluation, Android Studio, plugin, usability.
Content

1 Introduction .................................................................................................................. 1
  1.1 AUE Approach ................................................................................................... 2
  1.2 Project objectives ............................................................................................... 3
  1.3 Document structure ............................................................................................ 4

2 State of the art .............................................................................................................. 5
  2.1 Study on AUE .................................................................................................... 5
      2.1.1 Logging mechanism .................................................................................. 5
      2.1.2 Analysis mechanism ............................................................................... 7
      2.1.3 Applicability ............................................................................................ 8
      2.1.4 Summary of AUE ..................................................................................... 9
  2.2 Study on Task Modeling .................................................................................. 10
      2.2.1 Hierarchical task analysis (HTA) ............................................................. 11
      2.2.2 Goals, Operators, Methods, Set of rules (GOMS) .................................... 12
      2.2.3 Groupware Task Analysis (GTA) ............................................................. 13
      2.2.4 Concur Task Trees (CTT) ....................................................................... 14
      2.2.5 Méthode Analythique de Description de tâches (MAD*) ......................... 15
      2.2.6 Task Knowledge Structure (TKS) ............................................................. 16
      2.2.7 Diane+ ....................................................................................................... 17
      2.2.8 MUSE ........................................................................................................ 17
      2.2.9 TOOD ........................................................................................................ 18
  2.3 Modeling Techniques Analysis ........................................................................ 19
      2.3.1 Models Differences ................................................................................. 19
      2.3.2 Use case .................................................................................................... 23
      2.3.3 HTA ........................................................................................................... 24
      2.3.4 CTT ............................................................................................................ 25
      2.3.5 Selection and Conclusions ....................................................................... 26
      2.3.6 Model Functioning .................................................................................... 26

3 Requirements Specification ....................................................................................... 28
  3.1 Introduction .......................................................................................................... 28
      3.1.1 Purpose ...................................................................................................... 28
      3.1.2 Document Conventions .......................................................................... 28
      3.1.3 Scope ......................................................................................................... 28
      3.1.4 Definitions, Acronyms and Abbreviations .............................................. 28
3.1.5 Overview ................................................................................................... 29

3.2 Overall Description ..................................................................................... 29

3.2.1 Product Perspective ................................................................................ 29

3.2.2 Product Features .................................................................................... 29

3.2.3 User Classes and Characteristics ........................................................... 30

3.2.4 Operating Environment ....................................................................... 30

3.2.5 Design and Implementation Constraints ............................................. 30

3.2.6 User Documentation ............................................................................. 30

3.2.7 Assumptions and Dependencies .......................................................... 30

3.3 Functional Requirements .......................................................................... 30

3.3.1 Task Creation ........................................................................................ 30

3.3.2 Task Edition .......................................................................................... 31

3.3.3 Task Elimination ................................................................................... 32

3.3.4 Task Management and Display ............................................................ 32

3.3.5 Task Data Logging Description ............................................................ 32

3.3.6 GAMA Logging Code Provision .......................................................... 32

3.3.7 Task Diagram Export .......................................................................... 33

3.3.8 Task Diagram Import ........................................................................... 33

3.3.9 Plugin Help ........................................................................................... 33

3.4 External Interface Requirements ............................................................... 33

3.4.1 User Interfaces ....................................................................................... 33

3.5 Non-Functional Requirements .................................................................. 34

3.5.1 Performance Requirements ................................................................. 34

3.5.2 Safety Requirements ........................................................................... 35

3.5.3 Accessibility Requirements ................................................................. 35

3.5.4 Usability Requirements ....................................................................... 35

3.5.5 Reliability Requirements ..................................................................... 35

3.5.6 Efficiency Requirements ..................................................................... 35

3.5.7 Maintainability Requirements .............................................................. 35

4 Design ........................................................................................................... 36

4.1.1 Class Diagram ....................................................................................... 37

4.1.2 Models .................................................................................................. 39

4.1.3 Views ..................................................................................................... 40

4.1.4 Controllers ............................................................................................ 43

4.1.5 Glossary ............................................................................................... 44
5 Implementation .......................................................................................................... 47
  5.1 Technology ........................................................................................................... 47
  5.2 Sequence Diagrams .......................................................................................... 47
  5.3 Implementation Difficulties ............................................................................. 49
  5.4 Plugin Setup ..................................................................................................... 50
6 Application to Case Study ......................................................................................... 51
  6.1 Evaluation Methodology .................................................................................. 53
  6.2 Test Cases ......................................................................................................... 53
  6.3 Results .............................................................................................................. 60
7 Summary .................................................................................................................... 61
  7.1 Conclusions ...................................................................................................... 61
  7.2 Future Work ..................................................................................................... 62
References ...................................................................................................................... 64
Figures

Figure 1: System Solution Approach ......................................................... 3
Figure 3: HTA task model (from [13]) ....................................................... 12
Figure 4: GOMS task model (from [13]) .................................................... 13
Figure 5: GTA task model (from [13]) ....................................................... 14
Figure 6: CTT task model (from [13]) ....................................................... 15
Figure 7: MAD* task model (from [13]) ................................................... 16
Figure 8: TKS task model (from [13]) ....................................................... 16
Figure 9: Diane+ task model (from [13]) .................................................. 17
Figure 10: MUSE task model (from [13]) .................................................. 18
Figure 11: TOOD task model (from [13]) .................................................. 18
Figure 12: Use Case Instagram example development .................................. 23
Figure 13: HTA Instagram example development .................................... 24
Figure 14: CTT Instagram example development ...................................... 25
Figure 15: HTA example ............................................................................. 27
Figure 16: System approach ....................................................................... 36
Figure 17: UML Class Diagram simplified .................................................. 37
Figure 18: Plugin UML Class Diagram ....................................................... 38
Figure 19: LoggingElement, Task and Graph classes’ representation .............. 39
Figure 20: MainGUIFXML - Main plugin UI .............................................. 40
Figure 21: LoggingUsabilityFXML - Task’s usability logging UI ................. 41
Figure 22: GAMA Tracker code ................................................................. 42
Figure 23: Dimensions code ...................................................................... 42
Figure 24: Events code .............................................................................. 42
Figure 25: Screens code ............................................................................ 43
Figure 26: User ID code ............................................................................ 43
Figure 27: User Timing code ..................................................................... 43
Figure 28: MainGUIController and LoggingUsabilityController class's representation ........................................................................................................ 44
Figure 29: Create Task Sequence Diagram ................................................ 47
Figure 30: Edit Task Sequence Diagram ................................................... 48
Figure 31: Delete Task Sequence Diagram ............................................... 48
Figure 32: Get GAMA Code Sequence Diagram ........................................................... 49
Figure 33: Plugin usage sample – Main GUI ................................................................. 51
Figure 34: GAMA code injected view ........................................................................... 52
Figure 35: GAMA code references view ..................................................................... 52
Tables

Table 1: Task Models Semantic Differences (from [13])................................. 19
Table 2: Task Models Syntactic Differences (from [13])................................. 20
Table 3: Comparative amongst Task Models Techniques.............................. 21
Table 4: Task Modeling Techniques Valuation ............................................. 21
Table 5: Plugin classes glossary.................................................................... 45
Table 6: Test Case 1..................................................................................... 53
Table 7: Test Case 2..................................................................................... 53
Table 8: Test Case 3..................................................................................... 53
Table 9: Test Case 4..................................................................................... 54
Table 10: Test Case 5................................................................................... 54
Table 11: Test Case 6................................................................................... 54
Table 12: Test Case 7................................................................................... 55
Table 13: Test Case 8................................................................................... 55
Table 14: Test Case 9................................................................................... 55
Table 15: Test Case 10.................................................................................. 56
Table 16: Test Case 11.................................................................................. 56
Table 17: Test Case 12.................................................................................. 56
Table 18: Test Case 13.................................................................................. 57
Table 19: Test Case 14.................................................................................. 57
Table 20: Test Case 15.................................................................................. 57
Table 21: Test Case 16.................................................................................. 57
Table 22: Test Case 17.................................................................................. 58
Table 23: Test Case 18.................................................................................. 58
Table 24: Test Case 19.................................................................................. 58
Table 25: Test Case 20.................................................................................. 59
Table 26: Test Case 21.................................................................................. 59
Table 27: Test Case 22.................................................................................. 59
Table 28: Test Case 23.................................................................................. 59
1 Introduction

In software engineering world, **usability** is a key quality attribute that can be defined as the degree to which a system, service or product can be used with effectiveness, efficiency and satisfaction by specified users to achieve specified goals in a specified context of use [2]. Usability can be decomposed in five features: learnability, efficiency, memorability, amount of errors and satisfaction. The former one regards to how easily the user is able to learn how to use a software object the first time he uses it. The second one, efficiency, is based on how fast the users are able to accomplish a specific task. Memorability is about how easy is for users to efficiently use a software object after a considerable period of time without using it. The amount of errors a user makes when using a software object and their severity. Finally, but not less relevant, satisfaction, defining how pleased and glad the user feels after using a software object.

Human-Computer Interaction (HCI) is the discipline that aims to develop interactive systems with a high level of usability. The current usability evaluation techniques from the HCI field are really costly, in terms of budget and time, and hard to implement because of the wide range of different contexts of use of mobile apps that nowadays exists, and others that are ready to come.

Usability evaluation is a cornerstone of the User-Centred Design (UCD) [3], which is a development approach to achieve a good level of usability in software systems. Users’ feedback during operational use identifies long-term issues and provides input for future design. Assessing usability under real-user conditions is one of the hardest challenges experts in usability evaluation for mobile apps are facing with, due to the changing context of use.

Usability evaluation methods are possibly the most important methods between the diverse HCI techniques that nowadays exist for attaining usable products. They aim to gather data from user actions that can show how users are actually using the system and which problems they face, and comparing the real usage patterns with the expected and optimal ones, to identify usability troubles and be able to solve them for next releases. Usability evaluation processes imply different activities depending on the methods employed and the features of the procedures, but three main activities are common for all of them: capture, regarding the collection of usage data from users’ activity; analysis, the interpretation of usage data in order to identify usability problems; and critique, the possible ways to solve and improve the problems found and finally remove them.
A huge amount of resources is consumed when users are selected to carry out field studies. Automated Usability Evaluation (AUE) is a potentially good alternative, offering the possibility of gathering information from actual app usage. In particular, data capture for automated usability evaluation involves using software that automatically records usability data.

In order to capture the proper usability-related data and later perform successful usability evaluation activities, it is primordial to first focus on user actions and user-application interaction. Task analysis serves as the basis to understand users’ behaviour by allowing a detailed assessment of all the elements and processes that take place when the user interacts with the application.

1.1 AUE Approach

Google Analytics for Mobile Apps (GAMA) offers a cloud service for keeping track of user actions in mobile apps. This tool was designed for marketing purposes, but it is also used to measure UX characteristics. Automated solutions provide a manageable solution for the problem of changing contexts of use and user diversity. In addition, data mining techniques can also be used to perform the analysis of the gathered data and provide the results to usability experts for future critique.

Our tool will be part of a system aimed to perform automated usability evaluation activities [4]. It will work as a subsystem of a greater one that will allow developers and analysts to carry out all the necessary steps in a usability evaluation. This whole great system (Fig. 1) will offer the possibility of determining the tasks to analyse and selecting the user events to log through the usage of the subsystem we are going to develop in means of a plugin for Android Studio; the extraction of data and statistics from GAMA servers by means of Python scripts; and the final data mining analysis in order to identify usability problems by comparing the application real usage with the tasks design and be able to determine possible solutions.

This thesis work is focused on developing a plugin able to carry out the three first steps of the pre-processing part defined in Fig. 1. It permits the task analysis procedure, the selection of user events to log and the code instrumentation by providing the necessary GAMA calls code that automatically send the logged data to its servers.
1 Introduction

Despite the existence of some initial attempts to automatize usability evaluation of mobile applications, none of them fully integrates the definition of usability-relevant user tasks with the logging of usage data that can be later analysed with data mining techniques to uncover usability problems. We propose an approach based on user and task analysis to instrument the app code with calls to the Google Analytics logging service as the data capture and pre-processing part of a bigger system to evaluate Android mobile applications usability in the most possible automated way.

Our proposed solution addresses both the theoretical approach to model user tasks with the aim of instrumenting application code for data logging the user events when undertaking such tasks, and the practical integration of such activities into Android Studio, a widely used IDE (Integrated Development Environment), to facilitate the adoption by software development organizations.

And, more specifically, the thesis work will focus on proving the feasibility of extending Android Studio with usability annotation features in Android mobile applications, so that usage data is sent to GAMA servers through a plugin implementation.

1.2 Project objectives

Despite the existence of some initial attempts to automatize usability evaluation of mobile applications, none of them fully integrates the definition of usability-relevant user tasks with the logging of usage data that can be later analysed with data mining techniques to uncover usability problems. We propose an approach based on user and task analysis to instrument the app code with calls to the Google Analytics logging service as the data capture and pre-processing part of a bigger system to evaluate Android mobile applications usability in the most possible automated way.

Our proposed solution addresses both the theoretical approach to model user tasks with the aim of instrumenting application code for data logging the user events when undertaking such tasks, and the practical integration of such activities into Android Studio, a widely used IDE (Integrated Development Environment), to facilitate the adoption by software development organizations.

And, more specifically, the thesis work will focus on proving the feasibility of extending Android Studio with usability annotation features in Android mobile applications, so that usage data is sent to GAMA servers through a plugin implementation.
1.3 Document structure

This document is organized as:

- The first chapter introduces “usability” and “AUE” concepts, defines the solution approach and enumerates the content of this document.
- The second chapter is based on a study of the current knowledge about the different AUE methods and the diverse task modeling techniques. Afterwards, an analysis of which modeling technique better fits our project goals is carried out.
- The third chapter specifies the requirements for our project system.
- The fourth chapter defines the design of the system.
- The fifth chapter describes the implementation procedure followed to develop the tool.
- The sixth chapter is focused on a case study that uses the obtained tool and several testing cases to prove the feasibility of our project approach.
- The seventh chapter concludes the thesis work, summarizes its outcomes and advises some lines to possibly follow in future projects.
- Next, the set of references used as a support to complete the master thesis are enumerated.
- Finally, a couple of annexes are attached to show the diverse evaluation questionnaires used to assess the application to the case study.
2 State of the art

Our first step consists in investigating on the AUE existing methods and extracting the most proper ideas for our project. Afterwards, we carry out a research of the most useful existing task models by the professionals in the field in order to perform an analysis to determine which one better fits our needs and will be used for carrying out the task analysis and instrumenting the user events logging process.

2.1 Study on AUE

There exist many AUE methods aimed to mobile applications, but all of them are based on different aspects and characteristics, thus, we have investigated on how such currently existing methods work. All the studied articles have been written following their own guidelines, but we want to collect specific information regarding which are the mechanisms they use to log usability-related data, to analyse the captured data and which level of applicability they cover.

2.1.1 Logging mechanism

Feijó et al. [5] present a tool to automatically log user emotions and reactions in usability evaluation activities by using the front camera of a mobile phone while user is carrying out specified tasks. The different emotions and face signs the user express are recorded in order to get information related to user's emotions when using the app. The applications to be tested should be written using the libraries .dll already implemented. When the application is started by the user, a log file is created in order to gather the data from the specified events by using an emotion recognition software developed using RealSense SDK, allowing face location and expression detection.

MultiDevice RemUSINE is an environment created by Paternò et al. [6] to primarily compare the path followed by the user while is performing a predefined task to the ideal one. The method used is based on a comparison of planned user behaviour and actual user behaviour. The information about the planned logical behaviour of the user is contained in a previously developed CTT task model, and data about the actual user behaviour is provided by the logging tool, which is supposed to be available within the client environment. Timelines are the events that represent the user behaviour. The software records three different types: Simple Timeline, State-timeline and Deviation-timeline.
Balagtas-Fernandez et al. [7] present a framework, called EvaHelper, to help the users to collect usability data for later evaluation. To achieve this, they emphasis on an firstly mobile system preparation. The framework is aimed to log elements like the user interface, the corresponding action or event, like button clicks or combination of the number of steps to achieve one specified aim, and the time the action was performed. All this data is logged into a CSV format file.

Lettner et al. [8] present a toolkit that can be used to create Android mobile applications with a set of facilities to gather data when the users work with their app. This tool does not facilitate any option for modelling tasks. The authors specify that it is not necessary to model any task except when they are connected to a certain context of use. The events to record will depend on the metrics used for the performance. The purpose of the project is not to define events manually, but the idea is based on a record-replay system allowing the analysts to record different tasks and scenarios and compare the input of user, by using aspect-oriented programming (AOP) for the implementation of the mobile framework.

EVAL is a tool introduced by Leichtenstern et al. [9] that requires to be used in a laboratory with some specific facilities like two video cameras recording the actions performed by the user when is working with the analysed application. It records the entire mobile phone screen, collecting the images of the whole interaction of the user with the app, for example the input of commands or text.

MATE is a tool designed by Porat et al. [10] that helps users find usability problems in a specific application, task or screen. This is done by displaying aggregated performance and usage metrics for each application and by highlighting exceptional values that can provide an explanation to the low performance rate. The authors do not indicate any modelling technique but mentions that another tool must be used to define the tasks before using MATE. They define some metrics in order to determine what elements to record. These metrics are classified in effectiveness, focused on task completion; efficiency, based on the time user spends for completing tasks; and frequency of several procedures.

Matsuzawa et al. [11] present a software designed for usability evaluations of Android tablet applications, especially for evaluating the usability of the app’s graphical user interface (GUI). The system uses Logcat, a tool to acquire user operation histories. For recording the operation histories, the evaluators must add a flag of source code to the target application, and then Logcat can output user operation histories. Its operation is as follows: first, before analysing each task, the developer sets up a user test to acquire user
operation histories. The following consists in defining the tasks and their steps depending on the intent of the usability evaluation. Each step is defined by dividing a task for a screen transition and pushing a button that triggers a process. Every event is triggered by user’s button click to go from a step to the following when trying to complete a task. Some examples would be single tap, double tap, scroll and back button. In addition, operation time should be recorded. Once these events are detected, the approach of the system can determine whether the detected problems correspond to problem contents of the 10 usability heuristics for UI design by Jakob Nielsen.

Burzacca et al. [12] discuss the possibilities offered by remote usability evaluation of mobile applications based on logging user interactions and supporting the analysis of such data. The authors provides concrete indications about how the detected issues can be addressed, in particular when Web applications are accessed through mobile devices. Web Usability Probe (WUP) tool is used to exploit a proxy server, which inserts into the accessed Web pages some Java Scripts, which are then used to log user interactions, as those events related to forms, keyboards, touch, GPS, accelerometer, and semantic events, and then send such logs to the usability server.

2.1.2 Analysis mechanism

The tools presented by Feijó et al. and Leichtenstern et al. do not provide any analysis methodology of the data previously gathered.

MultiDevice RemUSINE processes the logged information and provides the necessary visualizations to analyze the usability of the application.

EvaHelper uses graph structures instead of UML diagrams to analyse the data because of its simplicity. The type of the graph used is GraphML format because of its flexibility and because of the available parsers that can read such files. Each UI component accessed by the user is represented as a node in the graph and the edges represent the actions/transitions done by the user. These ones are labelled by a sequence number, the action/event and the time duration between each action/event.

Once the applications created by Lettner et al.’s toolkit are closed, they send data to a backend server based on Google App Engine.

MATE allows the analysis of data by evaluating the metrics previously defined.

The following three phases procedure is used by Porat et al.’s software to identify problem areas: analysis whether data of each action type and the operation times are normalized, analysis of whether data are equal variance, when the data of both expert
users and novice users are normal distribution; and analysis of whether data are significant or not.

Burzacca et al.'s tool uses the Sequence Alignment Method (SAM) to analyse the data. In order to support the usability analysis, some previously defined optimal log representations are provided by the evaluators for each task the user must complete. Then, once the usability data has been collected, WUP enables the evaluator to perform a comparison between actual user behaviour and an optimal sequence of actions in order to identify deviations, which may indicate potential usability problems. This data to be compared could be represented in several ways: timelines identified by a label and a coloured bullet; storyboards representing page levels and navigation between them; and screen dumps of the user interfaces accessed by the user with indications of where the events occurred. Regarding intelligent analysis of the gathered data, SAM method compares and evaluates the different sequences of events by using private defined coefficients, indicating how much diverse are both elements.

2.1.3 Applicability

Feijó et al.'s tool is applicable to any kind of different mobile applications by just adding minor adjustments and taking into account that the software must be run on mobile devices with front camera.

MultiDevice RemUSINE is applicable to any application whose use cases consist in travelling through different app pages/sections to be able to reach the objectives of the user.

EvaHelper is applicable to all existing Android applications due to the framework itself is the one that helps the developer to add the needed code to analyze the app.

Lettner et al.'s toolkit it is supposed to work with all kind of Android mobile applications, and is portable and extendable to other platforms.

EVAL is aimed to work with any kind of app, under the condition of providing the above-mentioned technological instruments.

MATE can be embedded into Android applications and can be used as a remote usability testing.

In order to use Porat et al.’s software, the GUI of the target Android applications is required to be described in XML style.

Burzacca et al.’s tool is applicable to any Web site (accessed by a mobile phone) by exploiting a proxy-based architecture.
2.1.4 Summary of AUE

Usability evaluation procedures can be a difficult and tedious task. Many elements, internals and externals to the applications to evaluate, cognitive aspects and technical issues, take place into the development of the assessment. A wide variation exists among the several AUE existing methods that have been studied. Lettner and Holzmann proposed toolkit is aimed to automated and unsupervised evaluation for mobile applications that tracks the behaviour of the user through the interaction with the app, but they do not take into account task definition and modelling, which are relevant aspects to remark for our project. EvaHelper framework is introduced by Balgatas-Fernandez and Hussmann, allowing the insertion of events-calls code into the evaluated application code in order to indicate when the gathering of data will take place, but, again, it does not considerate task description and modelling.

A significant difference between these two mentioned tools is related to how they are involved into the app code. On the one hand, the first one is based on aspect-oriented programming, where developers do not have to change application code and the configuration can be done by using a wizard directly integrated in the used IDE. They use AspectJ, based on Java, being able to be merged with the application code at compile time. On the other hand, EvaHelper is based on manual insertions of code between the application code lines, giving the developer the responsibility to decide the point where the collection of data will be done. Both methods could be useful for our approach, but the selection of one of them depends on the task modelling technique that will be further chosen.

Others, like Feijó et al. and Leichtenstern et al., focus their work on logging the behaviour of the user while interacts with the application. These methodologies require the existence of electronic devices to record the mentioned interaction. Our proposal does not consider external devices to carry out the automated evaluation processes.

Burzacca and Paternò talk about a tool called WUP (Web Usability Probe), based on the analysis of web applications by focusing on the client-side logs as data source. From this point, they suggest a redefined tool for mobile applications based on web services.

Matsuzawa et al. does not take into account task modelling on their proposal either. Their approach is based on setting up user tests to acquire user operation histories and later analyse the logged data.
Porat et al. highlight the fact that the analyser should spend time on deciding what type of metrics will be used for logging the data, but does not give any clues about what metrics are part of his preferences. Their tool, called MATE (Mobile Analysis Tool for Usability Experts), uses different effectiveness, efficiency and use-frequency metrics for defining task scenarios to use in usability tests. This tool, as almost all the other ones, does not specify any task modelling technique to deploy the tasks, but at least mentions the fact that any model would be helpful for the performance of the evaluation.

Paternò, Russino and Santoro introduce an extension of a tool for evaluating desktop applications, called RemUSINE. Their work is based on the comparison of the ideal path and the one travelled by the user, by logging different kind of timelines. To compare both routes, they predefine the tasks using CTT modelling technique, later analysed in this document.

As we have summarized, most of the techniques focus their features on specific domains, specific types of applications, or external elements to carry out the evaluation. Our approach is aimed to be based on user and task analysis, however, only Paternò et al. mention a specific task modelling technique, what is a relevant issue for our objectives. Therefore, our next step is going to be related on investigating and deciding which task modelling technique is going to be used and, from this point, decide which mechanism will be used to log the data.

2.2 Study on Task Modeling

We call task to the definition of how the user reaches a defined goal in a specific application domain. Different objects and actions make a task. Objects are elements to manipulate in order to perform tasks. There are two different type of objects: perceivable objects, representing those entities that the users can interact with like menus, icons, voice, etc. and internal objects, those ones which belong to the application and need to be linked to other perceivable objects, for example the state of a request, the database, etc. Actions represent the activity or behaviour of the user when interacts with the application and allow the objects to communicate with each other. They can be logical, physical or cognitive.

Task analysis is used for identifying and understanding the structure, the flow, and the attributes of tasks. They identify the actions and cognitive processes required for a user to complete a task or achieve a particular goal. Some of their outputs are detailed
descriptions of all kind of activities involved with each task, duration and variability, frequency, allocation, complexity, environmental conditions, etc.

The fact of applying task analysis may be a very time consuming activity if is used with a high degree of detail on complex problems. However, the benefits of carrying out a good analysis are very worthy to take into account because they can avoid many future troubles.

From task analysis, we can derive to task models. They describe how to perform activities to reach user’s goals. They represent the ideal interaction between the user and the application’s user interface. Task models play an important role in the HCI field, because they represent the logical activities that should support users in reaching their goals.

Task models can be represented at various abstraction levels. On the one hand, when the objective is to specify only requirements regarding how activities should be performed, we consider only the main high-level tasks. On the other hand, when the main aim is to provide precise design indications then the activities are represented at a small granularity, thus including aspects related to the dialogue model of a user interface, which defines how system and user actions should be sequenced.

Those tasks are made of single elements, which represent individual actions that the user should take in order to complete the tasks; these elements might be accompanied by conditional indicators that make clear the execution of the task.

There are many reasons for developing task models. In some cases, the task model of an existing system is created in order to better understand the underlying design and analyze its potential limitations and how to overcome them. In other cases, designers create the task model of a new application yet to be developed. In our case, the purpose is to facilitate us to understand the user’s interaction with the application through its usage and serve as a basis for the instrumentation of the application code to log user events and the consequent performance of a usability evaluation.

There are many representations of task models, but not all of them successfully fit with the requirements and necessities of all projects. Thus, we have carried out a research of the most useful existing task models by the professionals in the field.

### 2.2.1 Hierarchical task analysis (HTA)

HTA decomposes a task based on, as its name says, the hierarchy and the plans of the set of subtasks in which it is divided. Tasks are decomposed into other small elements, called subtasks. This task decomposition is represented by a task hierarchy. Each subtask
should be accomplished according to the description and content of a predefined textual plan. This plan informally explains the relationships between the subtasks, and it describes an ordering in which the subtasks must be carried on. There are two different kinds of plans: the one defining a sequential task performance-ordering (do X, then Y, then Z…) and the implication ordering (if X do Y, else do Z…). There is a plan for each hierarchical level. In order to complete a task, all its goals must be achieved. Each goal has its own status, latent or active, and some conditional restrictions that must be respected for carrying out that task. In addition, for achieving each goal there are diverse operations. Operations are the fundamental unit of analysis. Each operation is directly related to one or more goals, the inputs for achieving such goals and the mentioned conditions and circumstances to complete the goal.

An interesting point of HTA is the failure stop rule. The end of the task decomposition is used to occur when we get all the necessary information that meets the purpose of the analysis, but HTA is characterized by stopping when the probability of failure \( p \) and the cost of failure \( c \) is somehow acceptable for the analysts. It is called \( p \times c \) criterion. This criteria helps to identify the source of error and posteriorly be able to correct it.

All these elements are graphically represented by using labels. The graphical model of a Hierarchical Task Analysis would be the following one:

![HTA task model (from [13])](image)

### 2.2.2 Goals, Operators, Methods, Set of rules (GOMS)

This model was originally defined for quantitative and qualitative predictions. Nowadays is also used to perform task modelling and task design. GOMS is composed of methods that are used to achieve specific goals. In other words, a set of Goals, a set of Operators, a set of Methods for achieving the goals, and a set of Selections rules for choosing among competing methods for goals.
Methods are indispensable to define how the tasks must be carried on. They define the procedure for achieving goals. Some methods, called higher-level methods, can be made by other methods, called lower-level, and described by operators and selection rules. Methods are made of several operators that describe the execution of the tasks. Operators are different actions, cognitive or physical; the user must perform to achieve task goals. Tasks are achieved by achieving goals. Often, different methods are related to the same goal, and then a selection rule is the one that indicates the chosen one.

The lowest level of decomposition in GOMS is the unit task, defined as a task the user is really focused on completing. In order to completing these tasks, user goals must be achieved. Each operator to accomplish the goals has its own execution time. Then, by calculating all these execution intervals of time, the time for performing a task can be predicted.

2.2.3 Groupware Task Analysis (GTA)

It is mainly developed for modelling the complexity of tasks in a cooperative environment. It is basically made by tasks achieved by performing different actions, taking into account roles, objects, agents and events.

Complex tasks are decomposed into unit tasks and basic tasks. The first ones are referred that tasks that belong to the lowest task level and the basic ones are defined by the tool that is used for performing the work. Complex tasks might be split up between agents or roles. Both unit and basic tasks may be decomposed into actions and events. Actions are identifiable components of basic or unit tasks, which have a meaning in performing a unit of work, but which derive their meaning only from the task they are part of.

Each thing that is relevant to the work in a certain situation is an object in the sense of task analysis. Objects may be physical or conceptual things like messages, gestures,
passwords, stories, signatures. Non-material objects, and sometimes physical objects as well, may be referred to by external representations of different character: verbal labels, graphics, metaphors, gestures.

Events are those specific situations under some concrete conditions that must happen for jumping from task to task or changing tasks’ states.

The most attractive feature of GTA is that it is able to represent cooperative tasks. This is possible because of the roles. They are assigned to different agents and determine the behavioural and organizational aspects in a group of tasks.

For its representation, goals and actions are displayed as tasks’ attributes. This fact does not avoid that each goal can be achieved in different ways and each action can be used in different tasks.

![Figure 4: GTA task model (from [13])](image)

### 2.2.4 Concur Task Trees (CTT)

CTT is one of the most famous task models. It is based on five common concepts: tasks, objects, actions, operators (constructors) and roles. The task model is hierarchical represented by different task trees, one for the cooperative section and one for each role that is involved in the task. Tasks, as commonly made in other task models, are decomposed in other basic tasks.

The links between the different tasks on the same level of decomposition are represented by the operators; they help to indicate different possible choices the user might take. In CTT, there also exist temporal operators, helping to describe cooperative tasks. Several actions and objects are specified for each basic task. Actions refer to the
activities or movements the user makes for performing tasks. They can be cognitive, logical or physical. The objects are all those elements that the user plays with in order to carry out the tasks. They can refer either to perceivable or application objects, and they can belong to more than one task at the same time.

One characteristic of CTT is also that both input and output actions associated with an object are also specified. CTT allows describing, apart from sequential tasks, also concurrent tasks, allowing them also to synchronise each other.

![CTT task model](image)

2.2.5 Méthode Analythique de Description de tâches (MAD*)

It is based on a tree structure by decomposing tasks. There are two different types of tasks: elementary and collaborative. The former ones are those that cannot be decomposed and are directly linked to one or many domain objects. Collaborative are those tasks that are broken down into several subtasks and linked by using operators. These ones vary depending on the relation between elements: synchronization, ordering, temporal and auxiliary. Tasks are identified by, among other attributes, a name, ID and a priority value to ease the differentiation of them. Goals are considered as attributes of tasks; however, they can be achieved by performing different tasks. There are two different task states, the initial, which is made by a set of preconditions, and the final state, made by a group of post conditions.
2.2.6 Task Knowledge Structure (TKS)

TKS represents the knowledge of a user (or agent) about a concrete task. Tasks are determined by the role the agent is assuming. Roles are defined as a determined set of tasks the agent is responsible for performing. An agent can perform one or more roles and vice versa. Every TKS has information of the task goal. This goal is made by different sub-goals, being all of them mandatory to accomplish in order to reach the main task goal. Every task can also be hierarchically decomposed in different subtasks, all of them having different sub-goals.

Constructors operate on tasks and goals. A goal can be reached by achieving different paths of sub-goals, what in other words is also called plan. Objects and actions perform the procedures to complete the subtasks. TKS offers a production rule system for selecting each context appropriate procedures.
2.2.7 Diane+

Diane+ allows the representation of hierarchically structured tasks and provides a set of specific annotations for these tasks. It uses a graphical notation to represent task decomposition as well as temporal and logical relationships between the tasks. Tasks are represented by boxes, which contain the name of the task and the constraints on the number of times the task can be executed. The shape of the box represents the actor of the task, for example, whether it is an end user, it is the system, or combination of both. The direction of the arrows that represent relationships amongst the tasks define the ordering on the execution.

![Diane+ task model](from [13])

2.2.8 MUSE

MUSE main purpose is about encouraging the fact of taking into account human factors in all software where an interaction with users exists. It is based on three phases: information selection and analysis, design synthesis and design specification.

Task hierarchy is used to define the task model. The top of the tree is covered by what is called organization, which is decomposed in different sub-organizations, called jobs. Each job is made of three concepts: (a) it has a main goal that can be divided into several sub-goals, as those sub-goals can also be decomposed into more sub-goals until it is necessary, (b) it contains a role list and (c) a function list with all the mandatory functions to carry on in order to perform a role.

Roles and functions are linked to a task, which can be decomposed into more sub-tasks. Each task depends on a finishing event and different conditional statements, existing different constructs to describe uncommon events that provoke failures.
2 State of the art

Figure 9: MUSE task model (from [13])

2.2.9 TOOD

This method is used for modelling in complex interactive systems. As often happens in task modelling, the structure is based on a hierarchy of tasks. Once the structure is defined, objects and elementary and control tasks must be described. Finally, concurrency is integrated. Tasks are made of a main goal, an identification type, the correspondent level in the hierarchical tree and the components that form that task. Each task is linked to a control structure made of six different objects related to the triggering event, conditions, required resources, input and output data and resultative reactions.

Figure 10: TOOD task model (from [13])
2.3 Modeling Techniques Analysis

We are going to perform an analysis of the above studied modeling techniques in order to decide which one better fits our needs and possibly combine the most interesting features of some of such models for building our own final task model. This analysis is based on the different aims of each model, their elemental attributes and domains, and the relationships and structures they take use of.

2.3.1 Models Differences

Several different aspects must be taken into account before deciding which task model is going to be used as a reference for our project goal. All of them are useful for task modeling, but not all fit with the aims and necessities of every project. In [13], there are two comparison procedures defined, by considering semantic and syntactic differences. Semantic differences concern about how the concepts differ across the different models. As can be observed in Table 1, the semantic comparison is based on the original discipline the models are focused on, the formalization of the systems the models are based on, possible existing collaborative aspects, the different contexts of use, the valuation of cognitive aspects, the system response, the scope of the temporal operators and the different types of manipulated objects.

<table>
<thead>
<tr>
<th>Origin</th>
<th>Formalization</th>
<th>Collaborative aspects</th>
<th>Content of use variation</th>
<th>Cognitive aspects</th>
<th>System response</th>
<th>Scope of constructors</th>
<th>Manipulated objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GOMS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GTA</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CTT</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TKS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAD*</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Else+</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOOD</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MUSE</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note. ✓ = supported, ✗ = unsupported.

HTA and GOMS are mainly focused on cognitive analysis whereas the other ones are quite more focused on software engineering processes. However, TKS could be classified in both groups due to basing its analysis in both aspects. The constructors that link tasks work in a different way depending on the methods, for example, they link sibling tasks in CTT and but link parents and children (tasks and subtasks) like in GTA.
or TKS. TOOD and CTT operators and connectors are based on formalized logical and mathematical processes. Some models like TKS and GTA are developed taking into consideration the user’s characteristics, the organization, the platform and other context of use features. For example, GTA was originally developed for analysing the complexity of tasks in cooperative environments.

Syntactic ones refer to the diverse vocabulary that is used for the same concepts in the different models. In Table 2 we can see a comparison of the different task model techniques taking into account the following aspects: how task goals are planned, how to reach high-level goals from low level ones, the lowest level in the task decomposition into subtasks, and the levels where the actions for reaching goals and completing tasks are placed.

<table>
<thead>
<tr>
<th>Task Planning</th>
<th>GOMS</th>
<th>GTA</th>
<th>CTT</th>
<th>MAD*</th>
<th>TKS</th>
<th>Disat*</th>
<th>TOOD</th>
<th>MUSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operationalization</td>
<td>Plan</td>
<td>Operators</td>
<td>Constructors</td>
<td>Operators</td>
<td>Constructors</td>
<td>Plan/constructors</td>
<td>Goals</td>
<td>Goals and construction</td>
</tr>
<tr>
<td>Task tree leaves</td>
<td>Methods/selection rules</td>
<td>Basic tasks</td>
<td>Basic tasks</td>
<td>Actions</td>
<td>Actions</td>
<td>Operations</td>
<td>Task</td>
<td>Task</td>
</tr>
<tr>
<td>Operational level</td>
<td>Tasks</td>
<td>Operators</td>
<td>Actions/system operations</td>
<td>Actions</td>
<td>Actions</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Commonly used words for task planning are *plan, operator, constructor* and *goal*. As can be observed in Table 2, these names are used in almost all of the methods, but sometimes with few meaning differences. Some techniques like GOMS, GTA, TKS, CTT and MUSE use specific syntaxes for referring to the leaves of the task tree decomposition.

From this point, we have to decide which method/s are going to serve as a basis for our final task modelling method. The first step consists in deciding which three methods could better fit the purposes of our work. Afterwards, we will carry out a practical analysis on those selected ones in order to finally determine which is the method our approach will be based on. Our selection is going to depend on, apart from the features of the different commented methods, complexity and clarity. The user should not spend too much time on neither studying and understanding how the technique works nor “drawing” or elaborating the task model to be evaluated, but a graphical definition would be welcome. We are not focused on collaborative tasks so, for this reason, our preferences will be focused on single ones.

A comparative table has been made in order to analyse what features all the diverse methods provide. This assessment is based on the following items:
2 State of the art

- **Graphical Notation**: Whether a specific graphical notation is provided by the technique for drawing a diagram representing the task decomposition.
- **Narrative**: Whether the task decomposition is accompanied by narrative description of each step and the way to move from one to another.
- **Main Environment**: The main task environment the technique is aimed to.
- **Complexity**: Level of complexity when applying the technique taking into account learning time, resources and procedures.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Graphical Notation</th>
<th>Narrative</th>
<th>Main Environment</th>
<th>Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTA</td>
<td>Specified</td>
<td>Yes</td>
<td>Individual</td>
<td>Low</td>
</tr>
<tr>
<td>GOMS</td>
<td>Not Specified</td>
<td>Limited</td>
<td>Individual</td>
<td>Low</td>
</tr>
<tr>
<td>GTA</td>
<td>Not Specified</td>
<td>No</td>
<td>Cooperative</td>
<td>Medium</td>
</tr>
<tr>
<td>CCT</td>
<td>Specified</td>
<td>Limited</td>
<td>Both</td>
<td>Medium</td>
</tr>
<tr>
<td>MAD*</td>
<td>Not Specified</td>
<td>No</td>
<td>Both</td>
<td>Low</td>
</tr>
<tr>
<td>TKS</td>
<td>Not Specified</td>
<td>No</td>
<td>Both</td>
<td>Medium</td>
</tr>
<tr>
<td>Diane+</td>
<td>Specified</td>
<td>No</td>
<td>Individual</td>
<td>High</td>
</tr>
<tr>
<td>MUSE</td>
<td>Not Specified</td>
<td>No</td>
<td>Individual</td>
<td>Medium</td>
</tr>
<tr>
<td>TOOD</td>
<td>Specified</td>
<td>No</td>
<td>Both</td>
<td>High</td>
</tr>
<tr>
<td>Use Cases</td>
<td>Not Specified</td>
<td>Yes</td>
<td>Individual</td>
<td>Low</td>
</tr>
</tbody>
</table>

The fields with a green background represent the desired characteristics for our selection; the red ones mean the opposite, features that do not fit with our objectives; and the yellow would be a halfway. In order to grade the methods that better match our specifications, green fields have a value of 2 points, yellow ones 1 point and red ones 0 points. Then, we have carried out a classification of the mentioned methods from where the three most valued techniques will be the final candidates for our model.

<table>
<thead>
<tr>
<th>Technique</th>
<th>Valuation</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTA</td>
<td>8</td>
</tr>
<tr>
<td>Use cases</td>
<td>6</td>
</tr>
<tr>
<td>CCT</td>
<td>6</td>
</tr>
<tr>
<td>GOMS</td>
<td>5</td>
</tr>
<tr>
<td>Diane+</td>
<td>4</td>
</tr>
<tr>
<td>MAD*</td>
<td>4</td>
</tr>
<tr>
<td>TOOD</td>
<td>4</td>
</tr>
<tr>
<td>MUSE</td>
<td>3</td>
</tr>
<tr>
<td>TKS</td>
<td>3</td>
</tr>
<tr>
<td>GTA</td>
<td>1</td>
</tr>
</tbody>
</table>
As we can observe in Table 4, HTA, Use Cases and CTT are the three best qualified. Task hierarchy and decomposition are basic concepts and necessary for task modelling. Tasks must be broken down into several subtasks in order to facilitate the understanding of the different steps to follow in order to achieve the high-level goals. Tasks must be related and linked in some way, by using temporal constraints through operators and constructors. Those task models concerning cooperative tasks should specify diverse roles to ease the performance of each step to arrive to the main goal and complete the tasks. Actions, plans and task ordering must be properly specified in order to avoid misunderstanding when trying to carry the task out.

HTA structure is basically made by a hierarchical tree, what helps us to identify the different steps the user should follow in order to achieve the expected goals. By having plans and their corresponding goals and operations at each level, it is easier to define, describe and understand what options the user has and the possible future movements.

CTT is pretty similar to HTA but with one significant difference: there are no plans to describe the paths to achieve goals, but there exists a specific nomenclature to define the links between sibling tasks. By this naming, the user is able to understand the direction it has to move and how has to do it.

Use Cases is the clearest technique in terms of tasks description. The analyst is able to freely narrate how to carry on the different steps, exceptions and alternatives the user has to achieve to reach the main goal. The only missing feature is the graphical support.

Our next step consists in assessing which of these three methods is the best for our objectives. In order to understand how they really work, we are going to put them in practice by modeling a same task by using the three pre-selected techniques. This task consists in ‘Posting a photo on Instagram social network app’. This is going to help us to better understand their applicability in real existing cases and a later better requirements specification.
2.3.2 Use case

- **User case:** Post an image from the gallery on Instagram
- **User profile:** Average user
- **User case context:** Instagram app is already installed in the smartphone used for the evaluation; user profile is already created and logged in so that when user opens the app it automatically logs into the user’s profile; location settings are already turned on.
- **Use case steps:**
  1) The user taps on Instagram icon on his smartphone in order to open the app.
  2) Instagram displays the last posted image by any of the followed profiles the user is currently following and the icons correspondent to other user stories on the top of the screen.
  3) The user taps on the icon for uploading an image on the bottom of the screen.
  4) Instagram displays the selected default image and the possible phone gallery photos to upload.
  5) The user selects the image that wants to post.
  6) Instagram displays the image selected by the user and other possible phone gallery photos to upload.
  7) The user taps on “Next”.
  8) Instagram shows the selected picture and the possible filters to apply.
  9) The user searches and selects the filter called “Rise”.
  10) Instagram applies the filter on the displayed image.
  11) The user taps on “Next”.
  12) Instagram displays sharing options.
  13) The user types a title for the picture and taps on “Add location” option.
  14) Instagram displays a list of different locations.
  15) The user selects one location.
  16) Instagram displays sharing options with the selected location indicated.
  17) The user taps on “Tag People” option.
  18) Instagram displays the picture where to tag people.
  19) The user taps on the wished point where to tag any other user.
  20) Instagram displays an option to search users by name.
  21) The user types the user name to tag and selects it from the appeared list.
  22) Instagram displays the picture with the tagged user.
  23) The user taps on the tick.
  24) The system displays sharing options.
  25) The user tags Facebook option to share.
  26) Instagram displays Facebook option selected.
  27) The user taps on “Share”.
  28) Instagram posts the image and displays it on the initial wall view.

Figure 11: Use Case Instagram example development
2.3.3 HTA

Figure 12: HTA Instagram example development

0 - Post Image

plan 0:
do 1-2-5,
  if you want to apply filters do 3 after 2
  if you want set features do 4 after 2
  if you want to apply filters and set features do 1-2-3-4-5

plan 2:
  if you don't need to search the image, do 2.1-2.3-2.4

plan 3:
  if you don't need to adjust the filter, do 3.1-3.2-3.3-3.4

plan 4:
  you don't need to mandatory do any of them.

1 - Open app

2 - Select Image

2.1 Open Gallery

2.2 Search Image

2.3 Select Element

2.4 Tap Next

3 - Apply filters

3.1 Decide Filter

3.2 Search Filter

3.3 Select Filter

3.4 Adjust Filter

3.5 Tap Next

4 - Set Features

4.1 Set Title

4.2 Tap People

4.3 Add Location

4.4 Share Other SN

5 - Share Image

5.1 Check Info

5.2 Tap Share

3.4.1 Set Brightness

3.4.2 Set Saturation

3.4.3 Set Others
2.3.4 CTT

Figure 13: CTT Instagram example development
2.3.5 Selection and Conclusions

Having modelled the proposed example by using the three different techniques, we can better determine the advantages and disadvantages of each of them.

By using Use Cases, we can explain to the user everything we want him/her to do in detail, because it is based on a narrative structure and it is actually the best way to define and describe any kind of problem. However, graphical support is an important factor to take into account, and it is missing in this technique. Use Cases provide easier understanding about the steps to do and how to do them, they clarify more the idea of performing tasks step by step, subtask by subtask. However, sometimes, the fact of having so much text without any graphical elements could turn out to be a little bit tedious.

CTT is able to explain and narrate everything the user should do by just using a graphical diagram, a task tree. Its nomenclature identifies what kind of relation there is between all the linked elements. There exist different kind of linking constructors in order to indicate to the user what to do and how to do it. Although it is not hard to learn the behaviour of this method, it requires a small study of the technique to control its usage and utilize it in the correct way, what implies to consume more time learning and developing the tasks than what we desire.

HTA is a kind of a mix of both previous techniques: Use Cases and CTT. It provides a task tree structure and plans to narratively describe the different subtasks to perform. The plans substitute the nomenclature of CTT and facilitates the understanding and learnability of the method behaviour. This combination offers a better work fluency by providing graphical expressiveness with the added flexibility of natural language and low complexity.

As a summary, we reach the conclusion that the method that better fits our needs and can be the easiest to use is HTA. It takes part of the best points of the other two techniques and its complexity is the lowest. Being able to express what we need, the lowest the complexity and time-consuming is, the more suitable the technique is.

2.3.6 Model Functioning

As shown below in Figure 15, the user will be able to break a task down into its different subtasks by decomposing the main task through a tree structure. Each node or box represents a task to achieve. The plans allow the user to describe the functionality of the boxes and the possible alternatives or orderings to their performance.

The fulfilment of each subtask implies several events to log in order to assess what is happening during the realization of the task. Apart from the general events that our tool
allows to log through Google Analytics, our tool would also allow to control different features. The user will be able to select which boxes (tasks) wants to control and collect data from.

Taking as a sample the example of posting a photo on Instagram, some information that could be logged are the following ones:

- The execution of the tasks is done by following the ordering established by the plans made by the user, in this case 1-2-3-4-5, excepting the ordering of the subtasks 4.1-4.4, which do not follow any rules.
- Name and date of the picture selected in 2.3.
- Number and names of the different filters tapped on 3.2 and 3.3.
- Names of the adjusted filters in 3.4.
- Names of the different settings adjusted on 3.4.1-3.4.3.
- Number of screens for sharing on other social networks on subtask 4.4.
- Time needed to set up the GPS on 4.3.
- Which subtasks are not fulfilled on 4.

These are some of the possible features the evaluator will be able to log in order to later carry on the usability evaluation process. Apart from them, characteristics like time, taps on screen and other general and basic events will be also logged by the logging system from GAMA.
3 Requirements Specification

3.1 Introduction

The current chapter defines a software requirements specification for our project’s system Task Annotation for Usability Evaluation (TAUE). All the content included is created according the user’s needs and the system correct performance.

3.1.1 Purpose

The purpose is to clearly define all the requirements that the TAUE system has to satisfy in order to accomplish its goals. For this reason, this specification accurately provides details about functional and non-functional features as well as design constraints, scope and any important information for its development.

Therefore, this chapter is more than a requirements specification – it is also a statement of how requirements are derived from the input and how the requirements relate to each other.

The audience of this document is anybody who may be concerned in the system, including, but not limited to:

- Software developers
- Software/Usability analysts
- Quality assurance teams

3.1.2 Document Conventions

This specification is based on the guidelines from Software Requirements Specification (SRS) [14].

3.1.3 Scope

TAUE is a plugin for Android Studio that provides support to task analysis annotation for Android mobile applications in order to perform automated usability evaluation activities.

The system’s main goal is to ease the task decomposition through an HTA diagram and the instrumentation of user events’ log into the application under evaluation for each of the annotated tasks.

3.1.4 Definitions, Acronyms and Abbreviations

TAUE – Task Annotation for Usability Evaluation
3.1.5 Overview

Current 3.1 is a brief introduction which objective is to present this specification and a general overview of the SRS as well as the environment in where the requirements are specified.

Chapter 3.2 gives a general description of TAUE software in order to identify the main functionalities that have to be carried out, associated data, constraints and external factors that can affect to the development.

Chapter 3.3 defines every single requirement in detail.

3.2 Overall Description

This section provides a high-level overview of the TAUE and its environment in order to explain the major aspects that can help to understand the requirements detailed below.

3.2.1 Product Perspective

As mentioned above, TAUE has to provide support to task analysis annotation for automated evaluation activities for Android mobile applications by easing the stakeholders task decomposition in and user even logging operations.

3.2.2 Product Features

Task Analysis Annotation
- Task Creation: Creation of a new task.
- Task Edition: Edition of an existing selected task.
- Task Elimination: Elimination of an existing selected task.
- Task Management and Display: Management of the tasks diagram and display of the whole tasks diagram break down.

User Events Log
- Task Data Logging Description: Description of the task data to log.
- GAMA Logging Code Provision: Provided GAMA code to log user’s application events.

Others
- Task Diagram Export: Allows saving and exporting the current tasks diagram the user is working with.
- Task Diagram Import: Allows importing an existing task diagram previously defined.
3 Requirements Specification

• Plugin Help: Provides help for the use of the tool.

3.2.3 User Classes and Characteristics

• Software developers who are developing a software application or still defining its requirements and design.
• Software/Usability analytics who are currently evaluating an application.
• Quality teams in charge of checking the fulfilment of the application requirements and testing its behaviour.

3.2.4 Operating Environment

Windows 10.

3.2.5 Design and Implementation Constraints

TAUE is built on IntelliJ IDEA and developed in Java language, following MVC architectural pattern.

3.2.6 User Documentation

See chapter 5.4 of this document.

3.2.7 Assumptions and Dependencies

TAUE is developed in Java and, therefore, requires Java to be installed on the user’s system. Java 8 is the last version required. This is applied on Windows.

3.3 Functional Requirements

3.3.1 Task Creation

FR1. The system should allow the user to type a name for a new task.
FR2. The system should allow the user to type a position for a new task.
FR3. The system should allow the user to type a plan for a new task.
FR4. The system should allow the user to create new tasks with its defined features.
FR5. The system should be able to automatically create a diagram with the tasks created by the user.
FR6. The system should be able to add a new created task into the diagram.
FR7. The system should not allow the user to create a task with no task name or no position typed.
FR8. The system should not allow the user to create a task if there is no task with position 0 (diagram root) previously created.

FR9. The system should not allow the creation of a task in position \( x \), being subtask of \( y \), if there has not been previously created task \( y \), i.e. a task with position 2.1 should not be created if there is no task with position 2.

FR10. The system should not allow the creation of a task in a position \( X \) if there has not been previously created a task in previous position, i.e. a task with position 3.2.4 should not be created if there is no task with position 3.2.3.

FR11. When the user creates a new task in a position where there already exists one, the system should automatically increment the position of the existing one and all the other subsequent tasks. I.e. task \( X \) is in position 2.2, then user creates a new task \( Y \) in position 2.2, then task \( X \) should be automatically updated by the system to position 2.3, and the same process with all the subsequent tasks.

3.3.2 Task Edition

FR12. The system should allow the user to edit the name of a task selected by the user.

FR13. The system should allow the user to edit the position of a task selected by the user.

FR14. The system should allow the user to edit the plan of a task selected by the user.

FR15. The system should allow the user to save the task modifications done by the user and automatically update the tasks diagram.

FR16. The system should be able to update all the diagram tasks positions when a specific task position is edited.

FR17. When the user edits the position of a task to a position in which another task already exists, in case the new position is before the original one, all the positions of the affected tasks must be automatically changed by the system to a subsequent position. I.e. given the existing tasks in positions 1, 2, 3, 4 and 5, when the user edits the task in position 4 to position 2, the task that was in position 2 will be edited to position 3, and task that was in position 3 will be edited to position 4. In case the new position is after the original one, all the positions of the affected tasks must be automatically changed by the system to a previous position. I.e. given the existing tasks in positions 1, 2,
3, 4 and 5, when the user edits the task in position 2 to position 4, the task that was in position 4 is moved to position 3, and the task that was in position 3 will be edited to position 2.

3.3.3 Task Elimination

**FR18.** The system should allow the user to delete a task selected by the user.

**FR19.** The system should automatically delete all the subtasks of the task deleted by the user.

**FR20.** The system should automatically update the tasks diagram when the user deletes a task.

**FR21.** When the user deletes a task, all the tasks in subsequent positions should be automatically moved by the system to one previous position.

**FR22.** The system should not allow the user to delete the task with position 0 (root task).

3.3.4 Task Management and Display

**FR23.** The system should allow the user to select a task to edit or delete.

**FR24.** The system should display the name, position and plan of a task selected by the user.

**FR25.** The system should display a diagram with all the tasks created by the user.

**FR26.** The system should automatically update the diagram and display it after the user creates a new task.

**FR27.** The system should automatically update the diagram and display it after the user edits an existing task.

**FR28.** The system should automatically update the diagram and display it after the user deletes an existing task.

3.3.5 Task Data Logging Description

**FR29.** The system should allow the user to narratively describe the data to log for a specific task.

3.3.6 GAMA Logging Code Provision

**FR30.** The system should allow the user to determine which user events wants to gather data from.
FR31. The system should allow the user to get the corresponding GAMA code to send the user events logged data to GAMA servers.

FR32. The system should allow the user to get the corresponding GAMA code to collect personalised data related to its application usage.

FR33. The system should allow the user to get the corresponding GAMA code to collect data related to the user application screens.

FR34. The system should allow the user to get the corresponding GAMA code to collect data related to the different user application’s code events.

FR35. The system should allow the user to get the corresponding GAMA code to collect data related to the user application’s user ID.

FR36. The system should allow the user to get the corresponding GAMA code to collect data related to the user application’s user timing.

FR37. The system should allow the user to inject the GAMA code wherever the user desires into his/her application code.

FR38. The system should allow the user to indicate a reference where the GAMA code was injected.

3.3.7 Task Diagram Export

FR39. The system should allow the user to export the tasks diagram and all its information and features.

3.3.8 Task Diagram Import

FR40. The system should allow the user to import a tasks diagram previously created with all its information and features.

3.3.9 Plugin Help

FR41. The system should provide a guideline to support the user to use the system.

3.4 External Interface Requirements

3.4.1 User Interfaces

EIR1. The system main view should display the HTA tasks diagram.

EIR2. The system main view should display a tasks diagram tree view in order to allow the user selecting a single task.
**EIR3.** The system main view should provide a panel to indicate the characteristics of a new task to create.

**EIR4.** The system main view should provide a button to create a new task.

**EIR5.** The system main view should provide a panel to edit the features of the selected task in the tasks diagram tree view.

**EIR6.** The system main view should provide a button to save the modifications done on the task under edition.

**EIR7.** The system main view should provide a button to delete the selected task in the tasks diagram tree view.

**EIR8.** The system main view should provide some text fields to allow the user indicating the name of a task, the position and the plan.

**EIR9.** The system main view should display a popup error message to the user when s/he is trying to operate with the tasks in a not allowed way.

**EIR10.** The system main view should provide a button to export the tasks diagram the user is working with.

**EIR11.** The system main view should provide a button to import an already existing tasks diagram.

**EIR12.** The system main view should provide a button to show a guideline to support the user with the usage of the system.

**EIR13.** The system should provide a view for indicating the description of the events and features to log of the tree view selected task by the user. This view should also provide one button for each different type of GAMA code the user wants to get to further paste into his/her application code.

### 3.5 Non-Functional Requirements

#### 3.5.1 Performance Requirements

**NFR1.** The system should be implemented in Java programming language.

**NFR2.** The system should be built as a plugin for Android Studio IDE.

**NFR3.** The system should run under Microsoft Windows.

**NFR4.** In case of opening windows forms, popping error messages and saving settings, the possibly delay should be less than 2 seconds.
3.5.2 Safety Requirements

NFR5. The system should be able to be stopped if any operational delay is greater than 5 seconds.

3.5.3 Accessibility Requirements

NFR6. The system text will be displayed in English.

3.5.4 Usability Requirements

NFR7. The interaction user-system should be simple and should facilitate the user learning.
NFR8. The system learning time should be less than 1 hour.
NFR9. The system should provide well-structured learning guides for users and administrators.
NFR10. The system should provide informative and orientating error messages to the user.
NFR11. The system should own well-formed graphic interfaces.

3.5.5 Reliability Requirements

NFR12. The system should be consistent with good control and management errors.
NFR13. The time for restarting the system should be less than 1 minute.

3.5.6 Efficiency Requirements

NFR14. The response time of the system must be short, allowing the user to do a continuous and correct use of the system.

3.5.7 Maintainability Requirements

NFR15. The system should periodically be analysed by the administrator.
NFR16. The system should be adaptable to possible future changes without losing its functionality.
NFR17. The system updates should be in accordance to the rest of standards of this specification.
NFR18. All relevant modifications must be informed to all the maintenance team, the development team, the correspondent managers and the users.
4 Design

The plugin has been built for Android Studio because is the main and most used IDE that nowadays exists aimed to program Android mobile applications. We have chosen Model-View-Controller (MVC) as the architectural pattern in which we base the skeleton of our tool. As mentioned above, our approach is just focused on allowing the user to perform tasks analysis and allowing the user’s application to automatically send user events data to GAMA servers meanwhile it is used (Fig. 16 – Pre-processing), but we do not have to log the data that the application is sending (Fig. 16 - Usability logging & extraction). For this reason, we do not make use of a persistent database in order to store the model data, due to the necessities of our tool don’t require it, but we use a couple of Java Beans, one of them following the Singleton pattern. The views are represented by FXML files. There are as many controllers as views in order to manage the user-interface interaction.

Figure 15: System approach
4.1.1 Class Diagram

Fig. 18 represents the structure of the plugin. Below, Fig. 17 is limited to just display the classes’ names but not the attributes and methods in order to clarify the view and understanding of the diagram.

Figure 16: UML Class Diagram simplified
Most of relationships between elements in the UML diagram are dependencies/instantiations. Depending on the interaction of the user with the plugin, new instances of classes are created. The starting class is `PluginAction`, being the one in charge of creating a plugin element and launching `MainSwing` application. The manager of the main application and its view (`MainGUIFXML` file) is `MainGUIController`, which is also the one in charge of ruling the behaviour of the program. On one hand, thanks to an instantiation of `TaskBoxTreePane`, it is able to draw the HTA task decomposition diagram, represented by the controller’s association to an object `Graph`, made of `Tasks`. On the other hand, when `LoggingUsabilityApplication` is instantiated, it behaves in the same way: it is controlled, as well as its view `LoggingUsabilityFXML` file, by `LoggingUsabilityController`.

4.1.2 Models

Both `Task` and `Graph` java classes represent the data to manage. A task is the key element of our approach, signifying the core unit of our analysis. We do not provide a formal database to store all the tasks created by the users, but, in order to organize them and create the links between tasks, we structure them into a graph object.

![Figure 18: LogingElement, Task and Graph classes' representation](image)

In our approach, a task is the core component, for this reason its features, its relationship with other tasks, and its attachment into a graph are the elements that give us its essence. Therefore, a **Task** is made of its representative name, its position into the graph and a plan to follow for its execution. As can be observed in Fig. 19, each task also contains the possible set of subtasks that would complete its performance, and a **LoggingElement** object, which means a collection of the references that indicate where in the user’s code the execution of the task takes place. In order to unify all the tasks that
are part of the same task decomposition, we created the object **Graph**, being a *HashMap* object able to organize and control all the tasks in a same diagram by linking them to their position. This decision allows getting any task by indicating its position, as well as the addition and removal of new tasks into the graph.

### 4.1.3 Views

The plugin UI is mainly composed by two *fxml* views: the one in charge of displaying the task decomposition diagram, as well as its structure and its control panel, called **MainGUIFXML** (Fig. 20), loaded by **Main Swing** class; and **LoggingUsabilityFXML**, loaded by **LoggingUsabilityApplication**, showing the screen that allows the user to determine which characteristics and user events of each task wants to gather data from (Fig. 21).

![Task Annotation for Usability Logging](image)

**Figure 19: MainGUIFXML - Main plugin UI**

The main view is organized in three different areas. The biggest one displays the task decomposition diagram, which, following HTA standards, shows every task with its name and position as well as the links between tasks. Tasks’ plans are not shown on the diagram in order to clarify the graph structure view, nevertheless, the control pane displays the plan, in case of having it, of the selected item by the user. The graph is neither editable nor clickable, however, in order to be able to manage the already created
tasks, the area on the right side of the frame is the one that allows the user to choose the task to interact with. This panel is based on a tree structure and allows the user to fold/unfold the tasks that contain other tasks in order to see the graph level by level. By clicking on any task, its characteristics are shown in the control panel, located in the area at the bottom of the window, allowing the user its edition and its usability data logging preparation. This pane is the one in charge of allowing the management of any task. It provides three text fields to type the name, position and plan that constitute a task, and six buttons: two to execute the creation and elimination actions of any task; one for exporting the HTA diagram with all the correspondent tasks; another one to import the already exported diagrams and be able to continue working with them; one providing help and advices of how to use the tool; and a last one with the text “Logging” yielding to the following view.

![Logging Usability](image)

Figure 20: LoggingUsabilityFXML - Task’s usability logging UI
The functionality of this view is to let the user link the previously selected task with his/her app’s code. The panel provides a text area to narratively describe which data is wished to be collected. The rest of the view is mostly occupied by the elements that permit the users to get the code to paste into their apps to send data to GAMA servers. By clicking on the buttons that appear on the right side of the window, the GAMA calls’ code is automatically copied in the clipboard. Each button provides a different type of code. Depending on the analysis needs and objectives, the user will tap on the corresponding button and paste (Ctrl + V) the obtained code (copied in the clipboard) into the app code under evaluation, i.e. to get the code that logs data related to the screen UI, user will tap on Screens button, or for getting the code to log personalised attributes, user will tap on Dimensions button. For each kind of call, the view also provides a field to set a reference related to where the code was injected. The user can copy in the clipboard such reference by doing Ctrl+Alt+Shift+C just after pasting the GAMA code.

```
GoogleAnalytics analytics = GoogleAnalytics.getInstance(getContext());
Tracker mTracker = analytics.newTracker(R.xml.analytics_global_tracker);
// You should type the context of your Activity or Fragment to create the analytics instance.
```

Figure 21: GAMA Tracker code

```
String label, category, action, dimensionValue;
// Label, category and action are not mandatory.

mTracker.send(new HitBuilders.EventBuilder()
    .setCategory(category)
    .setAction(action)
    .setLabel(label)
    .setCustomDimension(1, dimensionValue).build());
// If you want to add more than one dimension, call the same method
// |setCustomDimension(int index, String dimensionValue)| concatenated with
// |the last setCustomDimension(...) call
// |i.e.: setDimension(1, dimensionValue1).setDimension(2, dimensionValue2)....|build();
```

Figure 22: Dimensions code

```
// Inject the following code inside the event to send.
String label, category, action;

mTracker.send(new HitBuilders.EventBuilder()
    .setCategory(category)
    .setAction(action)
    .setLabel(label)
    .build());
```

Figure 23: Events code
4 Design

Figure 24: Screens code

```java
mTracker.setScreenName(getScreenName()); // Set the name of the screen.
mTracker.send(new HitBuilders.ScreenViewBuilder()
    .build());
```

Figure 25: User ID code

```java
mTracker.set("&uid", user.getId());
mTracker.send(new HitBuilders.EventBuilder()
    .setCategory(category)
    .setAction(action)
    .build());
```

Figure 26: User Timing code

```java
String label, category, action; // Not mandatory
long timing;

mTracker.send(new HitBuilders.EventBuilder()
    .setCategory(category)
    .setAction(action)
    .setLabel(label)
    .setValue(timing)
    .build());
```

The content shown above in Figures 22-27 is the code copied in the clipboard when user taps on the corresponding button to further paste it on his/her own app’s code.

4.1.4 Controllers

These two classes might be the most relevant of the plugin in terms of task background management. They are the bridge amongst the views and the models. **GUIController** manages the main plugin UI (MainGUIFXML) by carrying out the development of the HTA task breaking down procedure. **LoggingUsabilityController** regulates the LoggingUsabilityFXML view, by saving into the tasks their usability logging data.
Everything happening on the UI is handled by these two classes. MainGUIController contains a Graph object in order to manage the creation, elimination and edition of tasks designated by the user, as well as the existing links between them. Another important functionality is the one of displaying the HTA graph with the tasks the user indicates by using the *abego TreeLayout* library [15]. Graphs import and export functionalities are also managed from this controller due to is the one in charge of the diagram displayed in the UI at the moment export button is tapped and has to deal with the creation of the new imported one.

There exists a kind of relationship between both classes. The main GUI controller is the one that launches the *LoggingUsabilityApplication*, which loads its correspondent view (Fig. 21), which is controlled by LoggingUsabilityController class. In other words, the main controller and the logging one exchange the necessary characteristics of the user selected’ task in order to save into the correspondent task the data’s information to send to GAMA server. The “logging” controller determines which type of usability data the user wants to inject in his/her code and provides it to ease the GAMA call. This controller is also in charge of registering into the corresponding task element the code references where the user injected the provided code.

4.1.5 Glossary

The next table shows a lexicon of all the classes that compose the plugin accompanied by a brief description of each of them and their location in the project path.
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Graph.java</strong></td>
<td>It is the definition of an HTA diagram, by containing all the tasks that compose the graph and their links.</td>
</tr>
<tr>
<td>Location: src/models/</td>
<td></td>
</tr>
<tr>
<td><strong>Help.java</strong></td>
<td>Provides a set of instructions explaining how to use the plugin.</td>
</tr>
<tr>
<td>Location: src/javafx/</td>
<td></td>
</tr>
<tr>
<td><strong>LoggingElement.java</strong></td>
<td>Refers to the classes and lines where the user injects the GAMA calls per each task.</td>
</tr>
<tr>
<td>Location: src/models/</td>
<td></td>
</tr>
<tr>
<td><strong>LoggingUsabilityApplication.java</strong></td>
<td>The window that offers the user the option to define which user events wants to log data from and the injection of the code in charge of sending data to GAMA servers.</td>
</tr>
<tr>
<td>Location: src/javafx/</td>
<td></td>
</tr>
<tr>
<td><strong>LoggingUsabilityController.java</strong></td>
<td>This controller manages the definition of the events the user wants to gather data from by allowing him/her to describe the features to control and the injection of the code to perform the data logging.</td>
</tr>
<tr>
<td>Location: src/controllers/</td>
<td></td>
</tr>
<tr>
<td><strong>LoggingUsabilityFXML.fxml</strong></td>
<td>It is the graphical representation of the LoggingUsabilityApplication class.</td>
</tr>
<tr>
<td>Location: src/fxml/</td>
<td></td>
</tr>
<tr>
<td><strong>MainGUIController.java</strong></td>
<td>One of the most important classes of the project. It has the control of almost everything that happen when the user interacts with the plugin. It manages the creation of tasks, their edition and elimination. It is also in charge of updating the HTA diagram GUI and its tree graphical representation as well as the user makes any kind of modification in it.</td>
</tr>
<tr>
<td>Location: src/controllers/</td>
<td></td>
</tr>
<tr>
<td><strong>MainGUILFXML.fxml</strong></td>
<td>The visual representation of the main user interface of the plugin. Controlled by GUIController.java class, it provides an area for the tasks management, a zone for displaying the HTA diagram and another one reflecting a tree structure definition of the graph.</td>
</tr>
<tr>
<td>Location: src/fxml/</td>
<td></td>
</tr>
<tr>
<td>File Name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>MainSwing.java</strong></td>
<td>Main application definition, composed by a Java Swing frame containing a JavaFX scene (application).</td>
</tr>
<tr>
<td>Location: src/main/</td>
<td></td>
</tr>
<tr>
<td><strong>/PluginAction.java</strong></td>
<td>Plugin behaviour management. It basically calls the execution of the MainSwing.java frame. The definition of this plugin action is located in</td>
</tr>
<tr>
<td>Location: src/main/</td>
<td>the file /resources/META-INF/plugin.xml, out of /src package.</td>
</tr>
<tr>
<td><strong>Task.java</strong></td>
<td>This class represents a single task in an HTA Diagram as well as its relevant features like, among others, the task name, the position in the</td>
</tr>
<tr>
<td>Location: src/models/</td>
<td>diagram, the possible plan to follow for the execution of the task and a list of the subtasks that compose it.</td>
</tr>
<tr>
<td><strong>/TaskBox.java</strong></td>
<td>Describes the model of the boxes of the HTA diagram.</td>
</tr>
<tr>
<td>Location: src/swing/</td>
<td></td>
</tr>
<tr>
<td><strong>/TaskBoxNodeExtentProvider.java</strong></td>
<td>Expresses the sizes of the boxes of the HTA diagram.</td>
</tr>
<tr>
<td>Location: src/swing/</td>
<td></td>
</tr>
<tr>
<td><strong>/TaskBoxTreePane.java</strong></td>
<td>This class is in charge of creating the tree graph that represents the HTA diagram and display it to the user.</td>
</tr>
<tr>
<td>Location: src/swing/</td>
<td></td>
</tr>
<tr>
<td><strong>/Utils.java</strong></td>
<td>Extra functionalities like warning messages display, long strings getters…</td>
</tr>
<tr>
<td>Location: src/utils/</td>
<td></td>
</tr>
</tbody>
</table>
5 Implementation

As mentioned above, the implementation of the task annotation approach for usability evaluation activities has been carried out through a plugin for the Android Studio IDE. In this chapter, we present all the aspects that have influenced over the plugin implementation process, as well as an explanation of the reasoning followed to decide how these developing procedures have been carried out and which technology has been used.

5.1 Technology

The plugin has been developed on IntelliJ IDEA, a Java IDE developed by Jetbrains in which Android Studio is based. The tool runs on a Java Swing frame, since IntelliJ is already integrated into a Swing application. To ease the construction of the UI and the user-tool interaction, a JavaFX application has been integrated into the Swing component.

5.2 Sequence Diagrams

The following sequence diagrams represent the most relevant implemented functions in order to achieve the fulfillment of the specified functional requirements.

![Create Task Sequence Diagram](image)
Figure 29: Edit Task Sequence Diagram

Figure 30: Delete Task Sequence Diagram
5.3 Implementation Difficulties

During the development of the plugin we have found several bumps in the road, some related to technical programming issues, others to external reasons.

The first option to graphically represent the HTA task decomposition graph was based on using HTML language. We used a Google HTML template to generate the diagram from the task characteristics the user indicates for each task, but as the user was modifying the diagram, the template should also be updated to display the changes. The problem was that the HTML file had to be included in the plugin, which is a jar file that cannot be neither “broken” nor overwritten, in other words, its internal files cannot be neither accessed nor modified. Therefore, once the plugin jar was installed into the user system and s/he was making use of it, the user could not see the changes on the UI because the HTML file in charge of applying them could not be overwritten. For this reason, we decided to implement this functionality by using Java Swing.

This decision brought more troubles. The base of the plugin was a JavaFX application containing the recently chosen Swing component to load the HTA diagram,
through the external TreeLayout library, which uses this language. The inconsistence appears when IntelliJ IDEA tries to run JavaFX with a Swing node integrated, due to the fact that IDEA is already a plugin integrated into a Swing application, and did not allow overloading our plugin skeleton. However, there is a component called JFXPanel aimed to embed JavaFX elements, independently of its content, into Swing applications. Then, to be able to build our plugin, it became a Swing frame integrating a JFXPanel to represent the whole UI, which in turn integrates a Swing node to display the HTA diagram.

5.4 Plugin Setup

Android Studio 2.3.3 (last version) comes with the 1.8.0_112 Runtime version, which is missing JavaFX, which is used for the implementation of the plugin. Hence users who want to use this plugin must switch to a more recent Java version and use it to run Android Studio instead of _112 one. The solution is to use the default JetBrains Runtime [16].

The plugin provides GAMA calls to send the user’s desired data, but user must first setup his/her application environment. First, the user has to register the app to analyse into GAMA system [17] and later has to include GAMA dependencies into the app code under evaluation [18].

The plugin also provides a support system in order to help the user to use the tool. It describes the different steps to follow for each of the possible actions to do with the plugin, as well as the mentioned setup instructions. This support is accessible from the control panel displayed at the bottom of the main view (Fig. 20), by making click on a button with the text “Help”.

50
6 Application to Case Study

To prove the feasibility of extending Android Studio IDE with a plugin to support usability logging for Android mobile applications, we carried out a case study by using the developed tool with the Universidad Politécnica de Madrid (UPM) official app. The task to analyse was called “Write an email” and was applied by a last year Bachelor student with programming knowledge.

These views reflect the task decomposition allowed by our plugin and the injection of GAMA code into the app code under evaluation. As we can see in Fig. 33, by using the control panel, the user breaks down the main task by creating such ones that compose it; a fact that allows the user to later decide which elements of the task performance wants to gather data from.

Then, the user selects on the panel on the right from Fig. 33 that task to log its usability and proceeds to tap on the “Logging” button. Afterwards, the user determines which events wants to log by getting copied in the clipboard the necessary code tapping on the corresponding buttons for later pasting it on his/her app, as it appears in Fig. 34.
Once the code is injected, then the user can take the reference of that code and save it on the logging section for the task under control (Fig. 35).

The user decided to break a task down into eight other subtasks. For each of such tasks, the user determined whether s/he wants to control it or not by adding the code regarding GAMA into the app code under development. Later, once the user injected the code, also kept the notes of the references where s/he made the injection.
6.1 Evaluation Methodology

In order to evaluate the quality of the developed tool we used several test cases. They have been carried out in order to check whether the development of the project has been done in accordance to the requirements specification and they have been successfully fulfilled.

6.2 Test Cases

The following are the set of testing cases carried out to assess all the requirements established above in this document.

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Goal</strong></td>
<td>User creates a new task in position 0.</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>There is no previous task created yet.</td>
</tr>
<tr>
<td><strong>Inputs</strong></td>
<td>Task name: “Root”, position: “0”, plan: “”</td>
</tr>
<tr>
<td><strong>Expected Output</strong></td>
<td>User taps on “Create” button and the task is successfully created with the user’s indicated characteristics and a diagram containing it is correctly created and displayed.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Goal</strong></td>
<td>The system doesn’t allow the user creating a new task in position 1</td>
</tr>
<tr>
<td><strong>Preconditions</strong></td>
<td>There is no previously root task created.</td>
</tr>
<tr>
<td><strong>Inputs</strong></td>
<td>Task name: “First”, position: “1”, plan: “”</td>
</tr>
<tr>
<td><strong>Expected Output</strong></td>
<td>The system notifies the user it cannot create the task because there is no task in position 0 (root) previously created.</td>
</tr>
<tr>
<td><strong>Requirements Fulfilment</strong></td>
<td>FR8.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Goal</strong></td>
<td>The system doesn’t allow the user creating a new task in position 2.1.1</td>
</tr>
</tbody>
</table>
There exist tasks in positions 0, 1 and 2, but there is no previously created task in position 2.1

Task name: “Two one one”, position: “2.1.1”, plan: “”

The system notifies the user it cannot create the task because there is no previously created parent task for the new task.

FR9

Test Case ID 4
Test Goal The system doesn’t allow the user creating a new task with no name nor position indicated.

Task name: “”, position: “”, plan: “Hello”

The system notifies the user it cannot create the task because there is no name nor position indicated.

FR7

Test Case ID 5
Test Goal The system doesn’t allow the user creating a new task in position 3.

There exist tasks in positions 0 and 1 but here is no previously created task in position 2.

Task name: “Three”, position: “3”, plan: “”

The system notifies the user it cannot create the task because there is no previously created task in position 2.

FR10

Test Case ID 6
Test Goal User creates a task in a position where already exists another task.

There already exists a previously created a task in position 3.

The system creates the new task indicated by the user in position 3 and adds it to the diagram. The task that already was in such position and all the subsequent tasks are moved to a subsequent positions, in this case the already existing task is moved to position 4.

<table>
<thead>
<tr>
<th>Expected Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user taps on “Save” button and the system modifies the task name from “Old” to “New” and the plan from “Hello” to “Bye”, save the modifications and updates the tasks diagram view.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements Fulfilment</th>
</tr>
</thead>
<tbody>
<tr>
<td>FR11, FR12, FR14, FR15, FR27.</td>
</tr>
</tbody>
</table>

| Test Case ID | 7 |
| Test Goal | User edits an already existing task by modifying its name and plan. |
| Preconditions | User selects a task that was previously successfully created with the characteristics: Task name: “Old”, position: 3, plan: “Hello” |
| Expected Output | The system updates the selected task by changing its position from 1 to 3. Task that already was in position 3 is moved to position 2. Task that already was in position 2 is moved to position 1. |
| Requirements Fulfilment | FR13, FR16, FR17 |

| Test Case ID | 8 |
| Test Goal | User selects the task in position 1 and edits its position to 3. |
| Preconditions | There exist tasks in positions: 0, 1, 2, 3 and 4. |
| Inputs | Task position: 3 |
| Expected Output | The system updates the selected task by changing its position from 1 to 3. Task that already was in position 3 is moved to position 2. Task that already was in position 2 is moved to position 1. |
| Requirements Fulfilment | FR13, FR16, FR17 |

| Test Case ID | 9 |
| Test Goal | User selects and deletes task in position 2 |
| Preconditions | Already existing tasks in positions: 0, 1, 2, 2.1, 2.1.1, 2.2, 3 |
| Inputs | - |
### Expected Output
User taps on “Delete” button and task in position 2 is deleted as well as its subtasks (tasks in positions 2.1, 2.1.1 and 2.2). Task in position 3 is moved to position 2. The changes are saved and the diagram update is displayed to the user.

### Requirements Fulfilment

#### Table 15: Test Case 10

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>The system doesn’t allow the user to delete root task (position 0)</td>
</tr>
<tr>
<td>Preconditions</td>
<td>There exists a task in position 0.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>The system shows an error message to the user indicating that the elimination of a task in position 0 is not possible.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR22</td>
</tr>
</tbody>
</table>

#### Table 16: Test Case 11

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>Task characteristic are displayed when users select a task.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>At least one existing task.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>Task appears selected and its features are displayed.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR23, FR24</td>
</tr>
</tbody>
</table>

#### Table 17: Test Case 12

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>The system allows the user describing the data to log for a selected task and determining which user events wants to log.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>The user selected an existing task and tapped on button “Logging”.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>A new frame with a text area is shown to the user and he is able to type the description of the data to log for the task selected. Several buttons for determining which events to log are also displayed to the user.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR29, FR30.</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------</td>
</tr>
</tbody>
</table>

Table 18: Test Case 13

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User gets the GAMA tracker code.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A task has been selected; “Logging” button has been tapped on. Later “GAMA tracker” button has been tapped on.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>Tracker code is copied in the clipboard.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR31</td>
</tr>
</tbody>
</table>

Table 19: Test Case 14

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User gets the GAMA code to log personalised data.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A task has been selected; “Logging” button has been tapped on. Later “Dimensions” button has been tapped on.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>Personalised dimensions logging code is copied in the clipboard.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR32</td>
</tr>
</tbody>
</table>

Table 20: Test Case 15

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User gets the GAMA code to log screen data.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A task has been selected; “Logging” button has been tapped on. Later “Screens” button has been tapped on.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>Screens logging code is copied in the clipboard.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR33</td>
</tr>
</tbody>
</table>

Table 21: Test Case 16

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User gets the GAMA code to collect events data.</td>
</tr>
</tbody>
</table>
Preconditions: A task has been selected; “Logging” button has been tapped on. Later “Events” button has been tapped on.

Inputs: -

Expected Output: Events logging code is copied in the clipboard.

Requirements Fulfilment: FR34

Table 22: Test Case 17

Test Case ID: 17

Test Goal: User gets the GAMA code to collect user ID data.

Preconditions: A task has been selected; “Logging” button has been tapped on. Later “User ID” button has been tapped on.

Inputs: -

Expected Output: User ID logging code is copied in the clipboard.

Requirements Fulfilment: FR35

Table 23: Test Case 18

Test Case ID: 18

Test Goal: User gets the GAMA code to log user timing data.

Preconditions: A task has been selected; “Logging” button has been tapped on. Later “User Timing” button has been tapped on.

Inputs: -

Expected Output: User timing logging code is copied in the clipboard.

Requirements Fulfilment: FR36

Table 24: Test Case 19

Test Case ID: 19

Test Goal: The user injects the GAMA code into his/her application code.

Preconditions: GAMA code has been obtained.

Inputs: -

Expected Output: GAMA code is pasted into the user’s application code.

Requirements Fulfilment: FR37
### Table 25: Test Case 20

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User sets reference where s/he injected GAMA code.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>User gets the reference.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>The reference is indicated by the user in the corresponding text field.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR38</td>
</tr>
</tbody>
</table>

### Table 26: Test Case 21

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User imports an already created tasks diagram.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A tasks diagram was already exported. User taps on “Import” button.</td>
</tr>
<tr>
<td>Inputs</td>
<td>File: tasks.ser</td>
</tr>
<tr>
<td>Expected Output</td>
<td>The tasks diagram is imported and displayed to the user.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR40.</td>
</tr>
</tbody>
</table>

### Table 27: Test Case 22

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>User exports the current diagram s/he is currently working with onto his/her local root directory.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>A tasks diagram created with at least 1 task. User taps on “Export” button.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>The tasks diagram is exported and saved onto his/her local root directory.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR39.</td>
</tr>
</tbody>
</table>

### Table 28: Test Case 23

<table>
<thead>
<tr>
<th>Test Case ID</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Goal</td>
<td>The user wants to read the help option.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>User taps on “Help” button.</td>
</tr>
<tr>
<td>Inputs</td>
<td>-</td>
</tr>
<tr>
<td>Expected Output</td>
<td>The support information is displayed to the user.</td>
</tr>
<tr>
<td>Requirements Fulfilment</td>
<td>FR41.</td>
</tr>
</tbody>
</table>
6.3 Results

As we can observe in the above tables, each of the described cases tests at least one of the functional requirements previously enumerated in the requirements specification. After carrying out all these test cases, we have observed that all the obtained outputs were reflecting the same results than the expected ones, which means that all the functional requirements have been successfully fulfilled, or in other words, that the developed plugin has properly respected the requirements and works as it was expected.
7 Summary

7.1 Conclusions

Until nowadays, there was no toolkit, framework or theoretical model able to, independently on the mobile application domain, fully integrate the definition of usability-relevant user tasks with the logging of usage data that can be later analysed with data mining techniques to uncover usability problems.

We have introduced an approach for providing support to user task analysis’ annotation in Android apps for long-term monitoring of usage. We have carried out a research on the most popular task modelling methods to select the best one for our objectives. HTA has been chosen, being the clearest and one of the easiest for learning its usage, providing a low level of complexity.

We have designed a technological solution aimed to semi-automatically formalize the task analysis activity necessary to carry out user event logging with a UCD focus, by means of a supporting tool built in the form of an Android Studio plugin.

The developed tool offers the mentioned support to developers to task analysis annotation by allowing the decomposition of any task into diverse sub-elements through a simple diagram editor by following the extended HTA task modelling method presented in this document. The plugin provides the necessary code to log the desired features of such tasks the user has been interested in evaluating. Furthermore, at the same time the plugin’s provided code is gathering the data while users are using the application, it is sending the information to GAMA servers for a future analysis and critique.

The results from [19] express the usability level of our system is quite good compared to the average benchmark for usability evaluation in the design of SUS. The obtained values mean that the system is not necessarily complex but needs a decrease in its level of complexity. A better performance can be achieved by adding automatisms in some of the actions to be carried out with the tool and clarifying the user interface.

We have proved the feasibility of our approach by its application to a case study under real-life conditions, confirming that all the functional requirements previously defined have been successfully fulfilled.

Finally, in order to get feedback suggesting future possible improvements to be considered, we asked the user that carried out the case study to fill out some questionnaires regarding usability and impressions issues.
7.2 Future Work

Plugin’s current state of automation is one of the key aspects to focus on increasing for future releases of the system. Automated facilitation of the elements and events for logging user-interaction data requires a deep study on all the domains that Android development encompasses. Characteristics like automatic detection of the classes where the tasks take place, automatic code’s inclusion into such classes and automated processes to get the references where those code injections were done are some examples of next iterations’ work.

Further work could also be dedicated to extend the tool to other integrated development environments, and also to iOS mobile operating system. The essence of the evaluation would be the same, but a preliminary study on Apple mobile applications market would be necessary.

Help support results quite convenient, since user employed it at least two times with much satisfaction. Nevertheless, if user needed to take a look at it 2-3 times means that s/he doubted what to do on different stages of the plugin usage. Although the support seems helpful, we should try to clarify the plugin UI in order to decrease the number of times the user needs help and achieve a more fluent usage. Better explanations of what to do at each step or less graphical elements in the interface could be options to take into account for next iteration.

Mandatory labour to do after the ending of this work will be to apply the implemented tool into more case studies with different kind of users in different contexts of use in order to perform a better analysis of the obtained software to discover possible existing errors and fix them. The more cases we analyse the more feedback and statistics we will get in order to dig into user needs and improve the result of our approach.
Acknowledgements

It could be said louder but not clearer, that this work could have not been done without the support and participation of many people during the performance of the thesis.

First, I want to enormous and honestly thank my thesis tutor Xavier Ferré, who has been continuously helping me via advices, suggestions, feedback and corrections not only for this work, but also for the presentation of the related paper [1] and other academic aspects.

Thanks to Fion Yang and Meihui Li for taking charge of my condition at Tongji University and collaborating on making successful my stay here.

To my friends, old and new ones, to make my life easier and happier providing me this unforgettable stage of my life.

And finally, but not less important, to my family. For being the ones that made my experience in China possible as well as being motivating me to work with courage and dedication for the performance of my master degree.

December 2017
References


