Usability-Based User-Centered Design of Android Applications

Swagatika Kar

Department of Computer Science, Berhampur University
Berhampur-760007, India
swagatikakar31@gmail.com

Abstract: *Usability is a quality attribute which makes the user interfaces easier to use. Now-a-days the use of mobile devices in increasing rapidly, which are mostly used by common people. So these mobile devices and their applications must have good usability features to achieve wide acceptance by the users. MobileHCI community has proposed various recommendations for these usability features to be included at the time of development to improve the usability. But there is a significant gap lying between the mobileHCI community and application development teams since many years. Many attempts have been made to bridge this gap. Our proposed work provide guidelines to the development teams of Android applications for including various usability features into their applications.*

Keywords: Usability, Application design patterns.

1. Introduction

What is “Usability”? Usability is a quality attribute which makes the user interfaces easier to use [4]. An application or product having a easier and pleasant user interface is widely acceptable by the users. User-Centered Design emphasizes on the users. A system design is considered to be user-centered, if it behaves as the user expects.

Now-a-days the use of mobile devices is increasing rapidly and these devices are mostly used by the common people who don’t have sound technical knowledge. Most of the smart phones available in the market, come with many applications. Android devices and their applications provide significant benefits to their users, in terms of portability, location awareness and accessibility. Because of the tremendous use of android devices, huge number of applications (or “apps”) has been developing since many years.

The usability of android devices and their applications differs from other computer systems. The software needs of these devices affect the development process of application, as these are embedded in the phones during manufacturing or installed by various mobile software distribution platforms such as Google Play. Users prefer to choose applications that are easy to learn, take less time to complete tasks and more user-friendly. Therefore, it is essential to include the usability features in the android applications.

Mobile Human Computer Interaction (mobileHCI) is the “discipline concerned with design, evaluation and implementation of interactive mobile applications for human use and with the study of major phenomena surrounding them”[46]. The mobileHCI community deals with these usability features since many years and proposed various recommendations that are needed to be included in the mobile applications to improve their usability. The Software Engineering community is also struggling to transform these solutions into actual code. Though both the disciplines sounds to be overlapping but there is a large gap in between these.

This work focuses on bridging this gap by proposing a process along with a guideline , which will help the developers to include the usability features in their android application in order to improve the usability.

2. Related Works

There has been extensive amount of research carried out on usability since many years. Here we have gone through the work proposed by *Juristo and Moreno*[3] for including usability in software applications with the help of a guideline. They termed these usability features are Functional Usability Features. They proposed the above mentioned guideline for development of computer-based applications, in high level of abstraction along with no traceability between the solution and the requirements.

Our work extends the work of *Juristo and Moreno*[3] for including usability features in the development of android applications. It also provides traceability between the solution and requirements along with low-level of abstraction and complete empirical validation.

3. Methodology Adopted

A mobile application development life cycle contains the following phases:- Requirement Specification & Analysis phase, Design phase, Development phase, Testing phase, Deployment phase, Maintenance and update phase.

Our proposed process activities are carried out in the first two phases of mobile application development life cycle. Some basic usability features are considered here namely, Navigation, Feedback, Undo, Warning, Help, Favorites.

By studying all of these usability features in detail and analyzing the recommendations proposed by mobileHCI community, Android guidelines for UI and the user views on different android applications on GooglePlay we prepared a guideline which will help the developer in including these usability features in their applications.

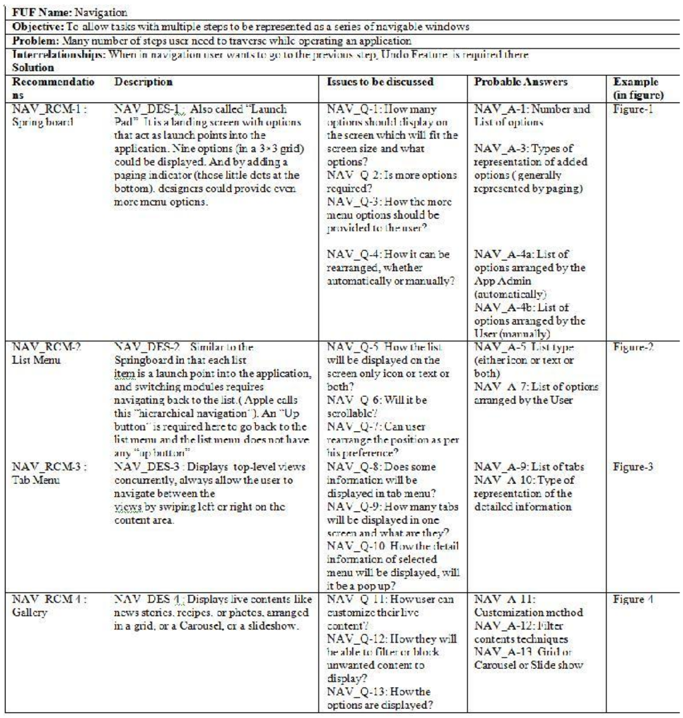
Navigation Usability Feature

“Navigation” is the most essential usability feature required in each and every mobile application. The users prefer simple and quick navigations. There are several navigation techniques being used by various android apps (a detail view explain in Extended Elicitation Guideline).

3.1 Extended Elicitation Guideline for Usability

The original guideline proposed previously [3] is modified and presented as follows in the Table-1 [5][6][8].

Table- 1: Extended Elicitation Guideline for Usability



FUF Name: Navigation				
Objective: To allow tasks with multiple steps to be represented as a series of navigable windows				
Problem: Many number of steps user need to traverse while operating an application				
Interrelationships: When in navigation user wants to go to the previous step, Undo Feature is required there				
Solution				
Recommendations	**Description**	**Issues to be discussed**	**Probable Answers**	**Example (in figure)**
NAV_RCM-1 : Spring board	NAV_DES-1 : Also called "Launch Pad" It is a landing screen with options that act as launch points into the application. Nine options (in a 3×3 grid) could be displayed. And by adding a paging indicator (those little dots at the bottom). designers could provide even more menu options.	NAV_Q-1: How many options should display on the screen which will fit the screen size and what options? NAV_Q-2: Is more options required? NAV_Q-3: How the more menu options should be provided to the user? NAV_Q-4: How it can be rearranged, whether automatically or manually?	NAV_A-1: Number and List of options NAV_A-3: Types of representation of added options (generally represented by paging) NAV_A-4a: List of options arranged by the App Admin (automatically) NAV_A-4b: List of options arranged by the User (manually)	Figure-1
NAV_RCM-2 List Menu	NAV_DES-2 Similar to the Springboard in that each list item is a launch point into the application, and switching modules requires navigating back to the list.(Apple calls this "hierarchical navigation"). An "Up button" is required here to go back to the list menu and the list menu does not have any "up button"	NAV_Q-5 How the list will be displayed on the screen only icon or text or both? NAV_Q-6: Will it be scrollable? NAV_Q-7: Can user rearrange the position as per his preference?	NAV_A-5 List type (either icon or text or both) NAV_A-7: List of options arranged by the User	Figure-2
NAV_RCM-3 : Tab Menu	NAV_DES-3 : Displays top-level views concurrently, always allow the user to navigate between the views by swiping left or right on the content area.	NAV_Q-8: Does some information will be displayed in tab menu? NAV_Q-9: How many tabs will be displayed in one screen and what are they? NAV_Q-10 How the detail information of selected menu will be displayed, will it be a pop up?	NAV_A-9: List of tabs NAV_A-10: Type of representation of the detailed information	Figure-3
NAV_RCM-4 : Gallery	NAV_DES-4 : Displays live contents like news stories, recipes, or photos, arranged in a grid, or a Carousel, or a slideshow.	NAV_Q-11: How user can customize their live content? NAV_Q-12: How they will be able to filter or block unwanted content to display? NAV_Q-13: How the options are displayed?	NAV_A-11: Customization method NAV_A-12: Filter contents techniques NAV_A-13 Grid or Carousel or Slide show	Figure-4

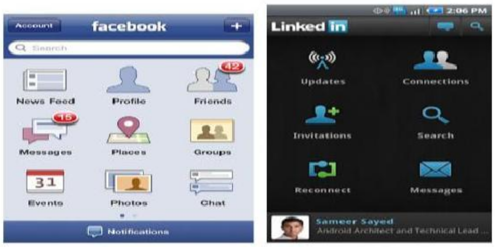


Figure-1 "Spring board" Navigation

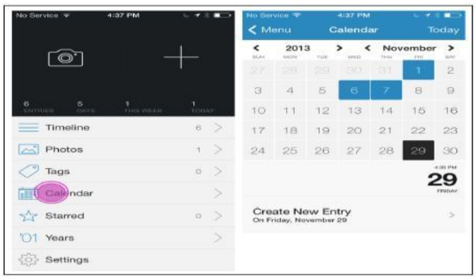


Figure-2 "List Menu" Navigation

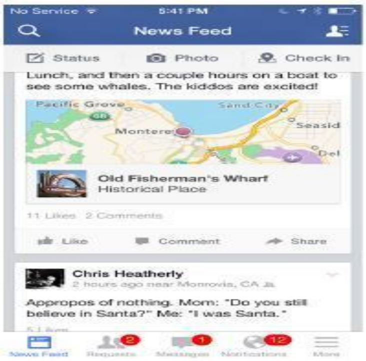

News Feed
Status
Photo
Check In
Lunch, and then a couple hours on a boat to see some whales. The kiddos are excited!
Old Fisherman's Wharf
Historical Place
Like
Comment
Share
Chris Heatherly
Appropos of nothing. Mom: "Do you still believe in Santa?" Me: "I was Santa."

Figure-3 "Tab Menu" Navigation

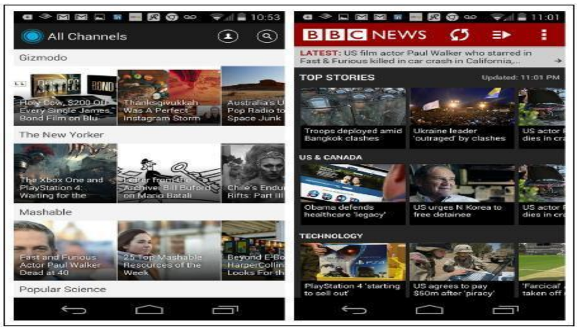


Figure-4 "Gallery" Navigation

3.2 Extended Use Case Meta-model for Usability

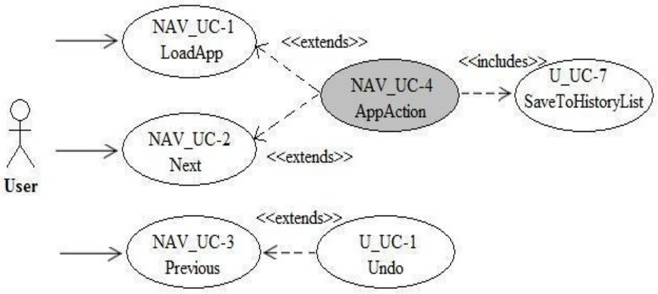


Figure-5: Extended Use Case Meta-model

The Use Case meta-model for Navigation feature is shown in Figure-1, in which six use cases are identified. Among these use cases, two are borrowed use cases from the Undo usability feature. Here NAV_UC-4 is the domain specific use case (represented in grey).

3.3 System Responsibilities for Usability

From the Extended Elicitation Guideline for usability, these System Responsibilities are derived. System Responsibilities can be defined as what the system supposed to do to achieve the mentioned goals.

Table- 2: System Responsibilities for Usability

System Responsibilities for Usability
NAV_SR-1 : Load App
NAV_SR-2 : Next Step
NAV_SR-3 : Previous Step
NAV_SR-4 : Back To Home
NAV_SR-5 : Update Current Position
NAV_SR-6 : Add More Options
NAV_SR-7 : Rearrangement Method of Added Options

3.4 MVC (Model-View-Controller) Architecture

The MVC architecture includes two design patterns for Object-Oriented Design out of the twenty-three design patterns proposed by GoF [7]. Here each System Responsibilities are further divided and assigned to classes and objects.

The MVC architecture of Navigation usability feature is given below.

Table- 3: MVC architecture (Navigation Usability Feature)

SR	Objects					
	View	**Controller**	**NavTree**	**NavNode**	**Concrete Command**	**Domain Class**
NAV_SR-1 : Load App	1. **View** listens user calls to load( ) the App. Upon receiving such a call, it forwards it to the **Controller**. 5. **View** receives the firstNode and displays it.	2. **Controller** locates the appropriate NavTree and orders it to load( ). 4. **Controller** passes on the firstNode to **View.**	3. **NavTree** will return the first **NavNode.**			
NAV_SR-2 :Next Step	1. **View** listens user calls to next( ). Upon receiving such call it forwards it to the **Controller**, together with the information entered by the user in the current step	2. **Controller** locates the appropriate **NavTree** and order its to load the next( ) node, sending along the information provided by the user through the **View**	3. Using user entered information provided by **Controller** (if needed) the **NavTree** determines the next **NavNode** in the navigation and orders it to setUp( ) with the user entered information	4. When ordered to setUp( ), the **NavNode** processes the information and if it is supposed to execute any actions it calls upon the corresponding Concrete Command to execute( )	5. When ordered to execute( ), the ConcreteCommand calls on its respective DomainClass to doAction( )	6. DomainClass executes the called action
NAV_SR-3 :Previous Step	1. When Previous button is clicked, the **View** forwards it to the **Controller** 5. **View** displays( ) the previousNode	2. **Controller** locates the appropriate NavTree and orders it to go back( ) to the previous **NavNode** 4. **Controller** forwards previousNode to **View**	3. **NavTree** returns the previousNode to the **Controller**			
NAV_SR-4 :Back To Home	1. When Home button is clicked, the **View** forwards it to the **Controller**	2. Whenever a call to goBackToTheFirstNode( ) is received, the **Controller** orders to undo every action in the HistoryList 3. Then it orders the **NavTree** to load the firstNode again 5. **Controller** passes on the firstNode to the **View**	4. NavTree loads the firstNode and returns it to the **Controller**			
NAV_SR-5 :Update Current Position	2a. If load(n) returns only the firstNode, the **View** will display it as the first element of the list. Every time a newNode is loaded, its name will be appended to the said list and highlighted as current 2b. If load(n) returns a list of allNodeNames [ ] in addition to the firstNode, **View** will display all names in a list, highlighting the name of the firstNode. Every time a newNode is added to the list, its name will be highlighted as current		1. When a **NavTree** is first called to load( ), if it is a linearTree, it will return a list of all node names, in addition to the firstNode in the sequence			
NAV_SR-6 :Add More Options	1. **View** listens the user call to add( ) more options to Springboard. Upon receiving such call it forwards it to the **Controller**	2. **Controller** locates the appropriate **NavTree** and order to the add( ) node, sending along the information provided by the user through the **View**	3. Using user entered information provided by **Controller** (if needed) the **NavTree** determines the lastNode in the navigation and orders it to setUp( ) with the user entered information	4. When ordered to setUp( ), the **NavNode** processes the information and if it is supposed to execute any actions it calls upon the corresponding ConcreteCommand to execute( )	5. When ordered to execute( ), the ConcreteCommand calls on its respective DomainClass to doAction( )	6. DomainClass executes the called action
NAV_SR-7 :Rearrangement Method of Added Options	1. **View** listens for the user call to rearrange( ) the options. Upon receiving such call it forwards it to the **Controller** 5. **View** listens for user call to move. Upon receiving such call it forwards it to the **Controller**	2. **Controller** selects the appropriate **NavTree** and orders it to load the currentNode 4. **Controller** notify the user that current node is selected 6. **Controller** locates the **NavTree** and orders it to move the currentNode moveTo(prev,node,next)	3. **NavTree** loads the currentNode and returns the currentNodePosition 7. After receiving this information provided by **Controller NavTree** orders it to move( )	8. When ordered to move( ), the **NavNode** processes the information, and calls upon corresponding **Concrete Command** to execute( )	9. When ordered to execute( ), the **ConcreteCommand** calls the respective **DomainClass** to doAction( )	10. **Domain Class** executes the called action

3.5 Design Meta-models

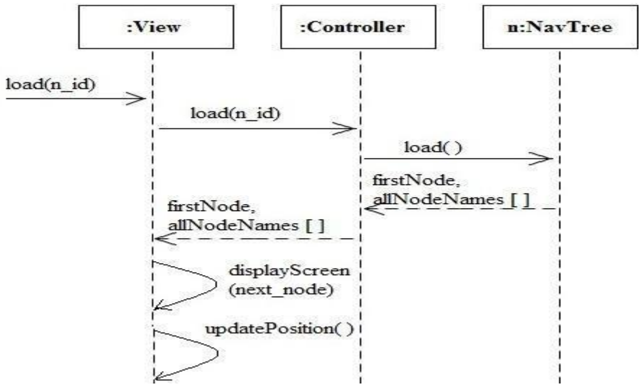


Figure-6: Sequence Diagram “Load App”

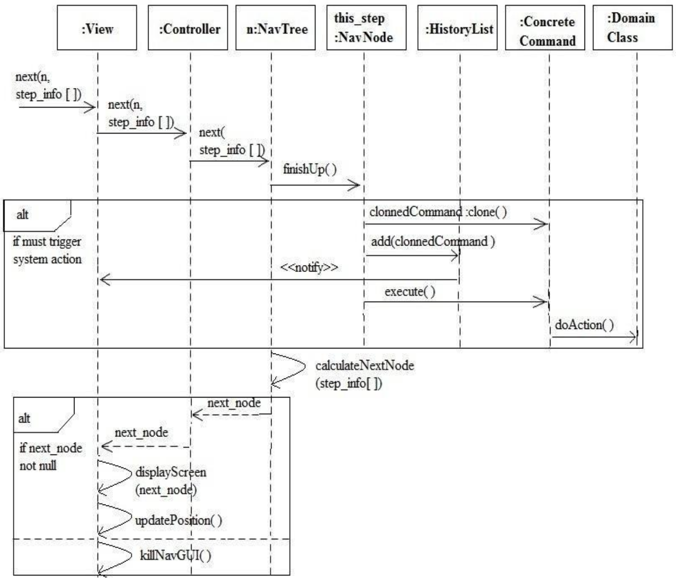


Figure-7: Sequence Diagram “Next Step”

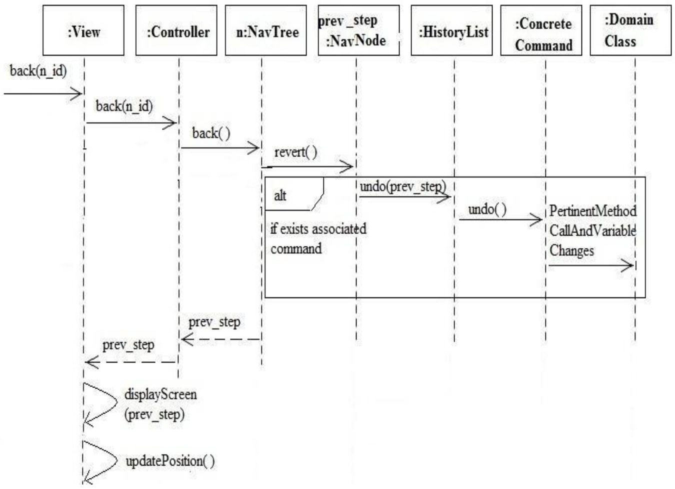


Figure-8: Sequence Diagram “Previous Step”

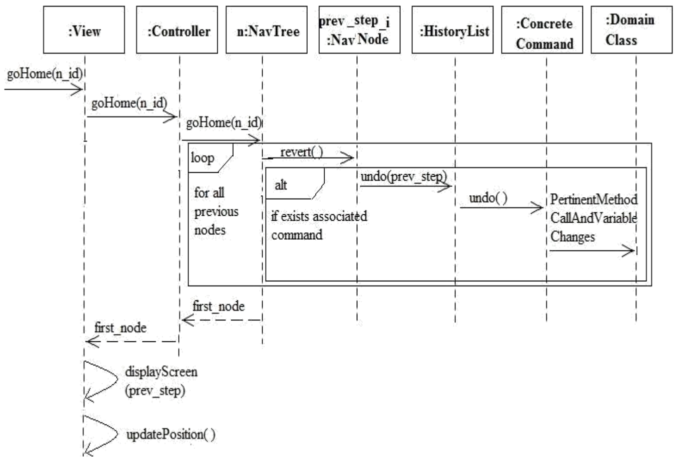


Figure-9: Sequence Diagram "Go To Home"

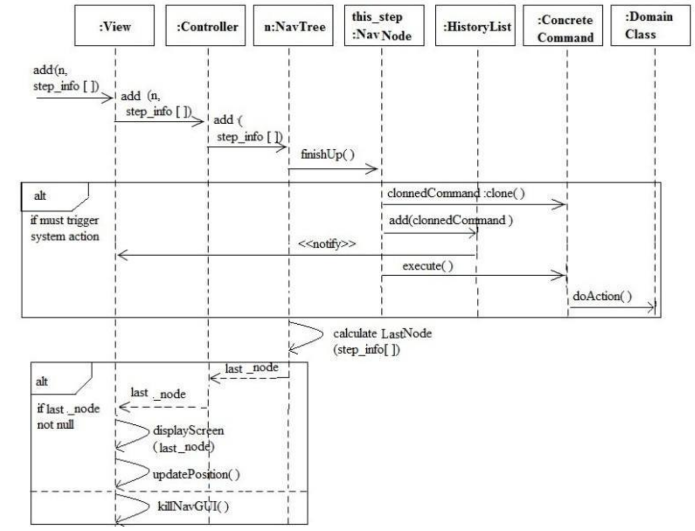


Figure-10: Sequence Diagram "Adding More Options"

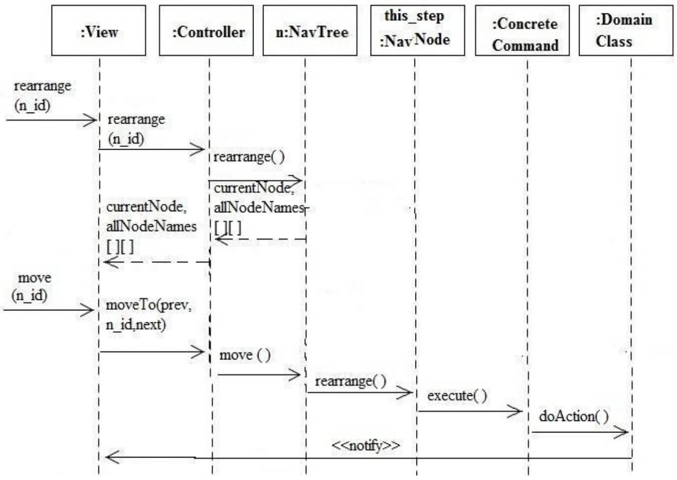


Figure-11: Sequence Diagram "Rearranging Options"

4. Results

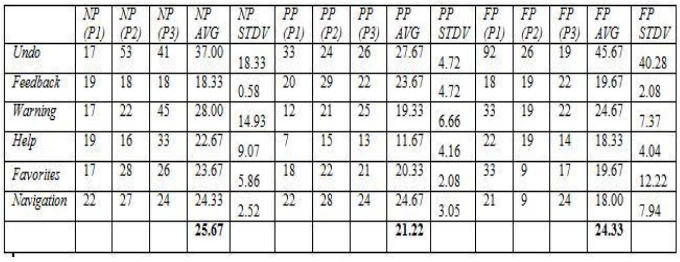
The proposed guidelines were applied during the development of three different Android Apps by nine developers (three developers per project). FP(Full Process)-Guidelines were provided both Analysis and Design phases, PP(Partial Process)-Guidelines were provided only in the Analysis phase, NP(Null Process)-No guidelines were provided.

1. **Electronic Business (E-biz):** An application where Traders and Merchants can sell or

   buy the products or services online.
2. **Hospital Management (Medisoft):** An application to manage the entire process of each patients of hospital with systems to provide better facilities and services with minimal time.
3. **Online E-Services:** An application to manage all steps of a service to their

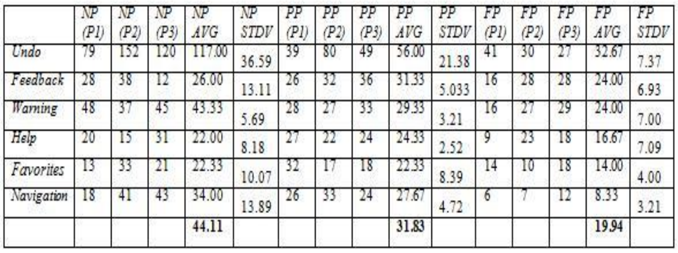
   customers online with a specific transaction number for each application.

Table- 4: Time (in min) spent by subject in analyzing the usability-related functionalities of their project



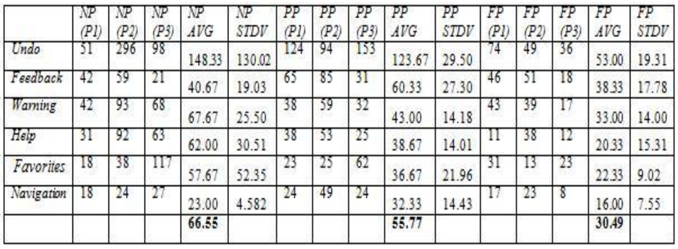
	NP (P1)	*NP (P2)*	*NP (P3)*	*NP AVG*	*NP STDV*	*PP (P1)*	*PP (P2)*	*PP (P3)*	*PP AVG*	*PP STDV*	*FP (P1)*	*FP (P2)*	*FP (P3)*	*FP AVG*	*FP STDV*
Undo	17	53	41	37.00	18.33	33	24	26	27.67	4.72	92	26	19	45.67	40.28
Feedback	19	18	18	18.33	0.58	20	29	22	23.67	4.72	18	19	22	19.67	2.08
Warning	17	22	45	28.00	14.93	12	21	25	19.33	6.66	33	19	22	24.67	7.37
Help	19	16	33	22.67	9.07	7	15	13	11.67	4.16	22	19	14	18.33	4.04
Favorites	17	28	26	23.67	5.86	18	22	21	20.33	2.08	33	9	17	19.67	12.22
Navigation	22	27	24	24.33	2.52	22	28	24	24.67	3.05	21	9	24	18.00	7.94
				25.67					**21.22**					**24.33**	

Table- 5: Time (in min) spent by subject in designing the usability-related functionalities of their project



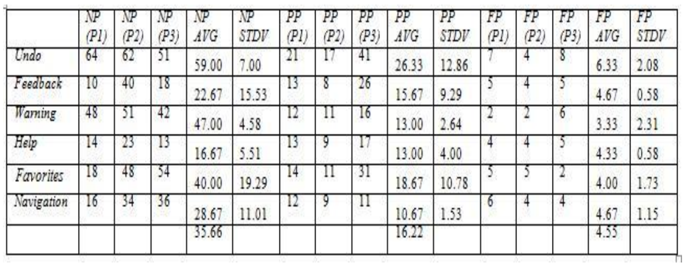
	NP (P1)	*NP (P2)*	*NP (P3)*	*NP AVG*	*NP STDV*	*PP (P1)*	*PP (P2)*	*PP (P3)*	*PP AVG*	*PP STDV*	*FP (P1)*	*FP (P2)*	*FP (P3)*	*FP AVG*	*FP STDV*
Undo	79	152	120	117.00	36.59	39	80	49	56.00	21.38	41	30	27	32.67	7.37
Feedback	28	38	12	26.00	13.11	26	32	36	31.33	5.033	16	28	28	24.00	6.93
Warning	48	37	45	43.33	5.69	28	27	33	29.33	3.21	16	27	29	24.00	7.00
Help	20	15	31	22.00	8.18	27	22	24	24.33	2.52	9	23	18	16.67	7.09
Favorites	13	33	21	22.33	10.07	32	17	18	22.33	8.39	14	10	18	14.00	4.00
Navigation	18	41	43	34.00	13.89	26	33	24	27.67	4.72	6	7	12	8.33	3.21
				44.11					**31.83**					**19.94**	

Table- 6: Time (in min) spent by subject in implementing the usability-related functionalities of their project



	NP (P1)	*NP (P2)*	*NP (P3)*	*NP AVG*	*NP STDV*	*PP (P1)*	*PP (P2)*	*PP (P3)*	*PP AVG*	*PP STDV*	*FP (P1)*	*FP (P2)*	*FP (P3)*	*FP AVG*	*FP STDV*
Undo	51	296	98	148.33	130.02	124	94	153	123.67	29.50	74	49	36	53.00	19.31
Feedback	42	59	21	40.67	19.03	65	85	31	60.33	27.30	46	51	18	38.33	17.78
Warning	42	93	68	67.67	25.50	38	59	32	43.00	14.18	43	39	17	33.00	14.00
Help	31	92	63	62.00	30.51	38	53	25	38.67	14.01	11	38	12	20.33	15.31
Favorites	18	38	117	57.67	52.35	23	25	62	36.67	21.96	31	13	23	22.33	9.02
Navigation	18	24	27	23.00	4.582	24	49	24	32.33	14.43	17	23	8	16.00	7.55
				66.55					**55.77**					**30.49**	

Table- 7: Time (in min) spent by subject in testing the usability-related functionalities of their project



	NP (P1)	*NP (P2)*	*NP (P3)*	*NP AVG*	*NP STDV*	*PP (P1)*	*PP (P2)*	*PP (P3)*	*PP AVG*	*PP STDV*	*FP (P1)*	*FP (P2)*	*FP (P3)*	*FP AVG*	*FP STDV*
Undo	64	62	51	59.00	7.00	21	17	41	26.33	12.86	7	4	8	6.33	2.08
Feedback	10	40	18	22.67	15.53	13	8	26	15.67	9.29	5	4	5	4.67	0.58
Warning	48	51	42	47.00	4.58	12	11	16	13.00	2.64	2	2	6	3.33	2.31
Help	14	23	13	16.67	5.51	13	9	17	13.00	4.00	4	4	5	4.33	0.58
Favorites	18	48	54	40.00	19.29	14	11	31	18.67	10.78	5	5	2	4.00	1.73
Navigation	16	34	36	28.67	11.01	12	9	11	10.67	1.53	6	4	4	4.67	1.15
				35.66					16.22					4.55	

Table- 8: Average total time (In Minute) to develop all usability related functionality

	NP	*PP*	*FP*
Requirement Analysis	25.67	21.22	24.33
System Design	44.11	31.83	19.94
Implementation	66.55	55.77	30.49
Testing	35.66	16.22	4.55
Total	171.99	125.04	79.31

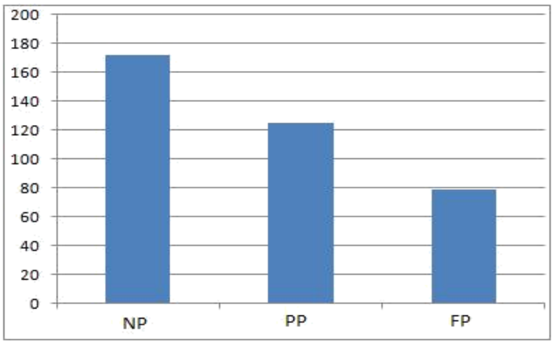


Figure-12: Average time to develop usability of the project

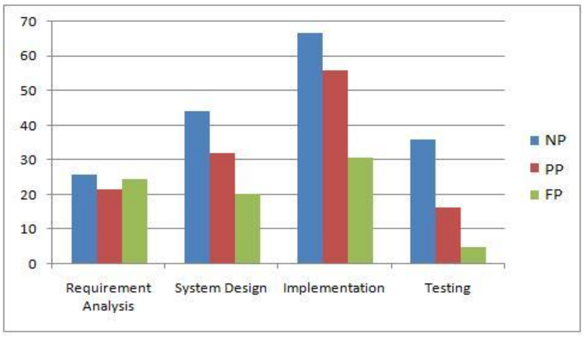


Figure-13: Average time to develop usability of the project (in different phases)

Kruskal-Wallis Test: One way ANOVA (Analysis Of Variance) test is performed on the data samples to prove that the samples are from different distributions.

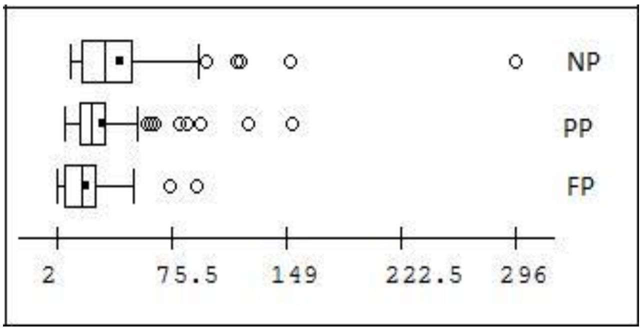


Figure-14: Independent Samples (Kruskal-Wallis test)

Ho: Samples came from the same distribution.
Ha: Samples came from different distributions.
(significance level = 0.05)

DF = 2

H = 12 / (216 * (216 + 1)) * 2673501.131944 - 3 * (216 + 1) = 33.460095

Critical Value: Chi-Square(2, 0.05) = 5.991465 P(>H) = 5.423E-008

Reject the null hypothesis at the 0.05 significance level. Samples came from different distributions.

Conclusion

In this paper we propose a solution to guide the Android Apps development teams in including the above six usability features into their applications, which will make their applications widely acceptable by the users. The proposed process can be either used manually or by process automation tool. As the proposed guidelines help including the usability features at the early stage of development, the cost of development and total time is reduced significantly.

References

[1] Bass, L. and John, B. E., 2002. Linking Usability to Software Architecture Patterns Through General Scenarios. **Journal of Systems and Software**, Volume 66, Issue 3, pp 188-197.

[2] Bass, L. and John, B. E, Kates, J. Achieving Usability Through Software Architecture. Technical Report. CMU/SEI-2001-TR-005, March 2001.

[3] Juristo, N., Moreno, A. and Sanchez-Segura, M.I., 2007. Guidelines for eliciting usability functionalities. IEEE Transactions on Software Engineering Volume 33, Issue 11, pp 744-758.

[4] Nielsen, J., 1993.Usability Engineering. Morgan Kaufmann Publishers, Boston, USA.

[5] Y-H. Cheng, W-K. Kuang, S-L. Su, (2010). An Android System Design and Implementation for Telematics Service. IEEE page 209.

[6] Android Developer Guide, available online, http://developer.android.com/

[7] Design Patterns: Elements of Reusable Object Oriented Software, Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides

[8] iOS Human Interface Guidelines by Apple

Author Profile

Swagatika Kar received the B.Tech. (Computer Science & Engineering) degree from Synergy Institute of Engineering and Technology and M.Tech. (Computer Science) degree from Berhampur University in the years 2010 and 2014, respectively. Her area of research involves Software Engineering and Data mining.