Hindawi Publishing Corporation
Mobile Information Systems
Volume 2016, Article ID 3083450, 22 pages
http://dx.doi.org/10.1155/2016/3083450

Research Article

Wise Mobile Icons Organization: Apps Taxonomy Classification Using Functionality Mining to Ease Apps Finding

David Lavid Ben Lulu and Tsvi Kuflik

Information Systems Department, The University of Haifa, Mount Carmel, 31905 Haifa, Israel

Correspondence should be addressed to Tsvi Kuflik; tsvikak@is.haifa.ac.il

Received 12 August 2015; Accepted 24 November 2015

Academic Editor: Salil Kanhere

Nowadays, every user has dozens of Apps on her/his mobile device. As time passes, it becomes increasingly difficult simply to find the desired App among those installed on the mobile device and launch it. In spite of several attempts to address this challenge, no good solution for this growing problem has yet been found. In this paper, we examine the idea of classifying Apps based on their functionality to allow users to find and access them easily. An App's functionality is elicited from the textual description of the App, as retrieved from the App store, and enriched by content from additional online publications. The functional representation is then classified into classes and mapped into personal categories using functional hierarchical compact taxonomies that can be easily presented on the small screen of a mobile device. Experiments and user studies demonstrated the potential of this approach.

1. Introduction

Smartphones have changed (and continue to change) our lives beyond recognition, as they combine the capabilities of cellular phones with advanced computing and communication technologies. By using various Apps, we are able to perform a variety of tasks using our mobile devices more easily than ever before. While the term "mobile App" is not clearly defined, it commonly refers to computer software developed to be executed on smartphones, tablet computers, automobile interfaces, gaming consoles, and other mobile devices, which usually run a specific operating system. Apps are designed to help users perform specific tasks with a specific purpose: reading and writing emails, calendar management, gaming, factory automation, banking, order tracking, and much more [1]. Apple recently reported that their App store contains more than one million Apps [2], downloaded at a rate of about 48.6 million Apps per day. This rate does not appear to be decreasing [3].

According to "The Next Web" magazine [4], the average user has 95 Apps installed on his or her mobile device (in addition to the Apps installed by the manufacturer) and spends more time using them than surfing the Internet: about 94 minutes per day [5]. The median for iPhone users is 108 Apps (88 general Apps + 20 preinstalled Apps) [6].

In 2011, Perez [7] already noted the difficulty of finding a particular installed App. Given that the average user uses only 15 Apps per week, most of the Apps that are installed on the device are rarely used. Thus, if, during a hike, the user comes across a deer and wants to take a picture of it using a filter App, by the time he/she finds the App, the deer has already disappeared (unless the App is in the home screen shortcuts). The studies on finding an installed App focus mainly on context-aware recommendation [8] and predictive App launching [9], but not on Apps organization. All the existing methods attempt to reduce the search space for the user, but the manner in which they achieve this differs.

One of the biggest problems in App search is that, in many cases, the name of the App bears no indication of its functionality (e.g., Instagram, Zedge, and Shazam) and therefore association- or word-based-related searches do not help. Pash [10] described the problem as it applies to iPhone users. When they drag one App onto another in order to make a folder, iOS offers only vague categories such as "Tools," "Reference," or "Productivity." The organization of items using categories is acceptable for libraries, movies, and music, but not for Apps. However, attempting to search for an App

Figure 1: Self-organizer. Functional classification of Apps.

using classes that are too general and abstract can be a lengthy and tedious process, and therefore, Apps organization must be made more intuitive, like their main functions (Actions), for instance (see Figure 1). This type of arrangement may facilitate and speed up the process of finding and launching a desired App. We suggest a method to facilitate the finding of Apps that are not commonly used and do not appear in the home screen shortcuts and whose names the user does not remember (hence, alphabetical order is not very effective), although he/she knows very well what functionality they provide.

This work examines the possibility of using machine learning techniques for the organization of Apps based on their functionality. The main questions we faced were as follows:

(1) Is it possible to perform Apps categorization based on their functionality?

(2) Is it possible to elicit the functionality of Apps from their textual description?

(3) Is it possible to provide personalized categorization of installed Apps?

(4) Is it possible to present the results in an intuitive visualization?

In our previous work [11], we already suggested the initial idea and demonstrated its feasibility with unsupervised algorithms (clustering). In this paper, we considerably extended that study by showing that it is possible to achieve better results using semisupervised machine learning techniques and by applying Web mining techniques for eliciting Apps functionality and for Apps labeling. In addition, we conducted evaluation and user studies to show that the suggested approach provides a solution to the problem that end users like.

The first challenge was already discussed by Böhmer and Krüger [12] in their study on icon arrangement by smartphone users. They studied how people organize Apps they have installed on their devices (using mainly observations). They found five different concepts for arranging icons on smartphone menus, including usage based, relatedness-based, usability-based, aesthetics-based, and external concepts (installation order, alphabetical order, default, etc.). The two most common ones were based on App usage frequency and Apps' functional relatedness. They concluded that users are interested in organizing Apps icons automatically according to the Apps' functionality. This conclusion supports our initial assumption, and therefore, we are not investigating this assumption further. Following their work, we suggest a representation for Apps based on their functional characteristics as elicited from their textual descriptions. Using this representation, we examine the possibility of ordering them hierarchically for presentation on the small mobile device screen.

The rest of this paper is organized as follows. In Section 2, we provide some background and related work. In Section 3, we provide an overview of the suggested multistep approach and an evaluation of the individual steps. In Section 4, we describe the user study and its results. In Sections 5 and 6, we present our discussion and conclusions, respectively.

2. Background and Related Work

Our work deals with resolving the problem of finding Apps installed on a personal mobile device. As such, in this section, solutions for App search are surveyed in general, and we discuss what can be learned from PC desktop organization solutions. Finally, we examine the solutions in the App stores regarding the specific problem.

2.1. App Search Engine Design. Shanahan and Glover [13] discussed the need for a functional search method to address the challenge of App search and how to match Apps with users' needs. They found that the majority of classic search engine queries are functional in nature or task-centric. Thus, they concluded that the App search is a unique search and unlike other searches such as multimedia or Web search. Therefore, the functional representation of Apps requires more research and an essentially different approach. Rudolf [14] described how a perfect App search engine should work according to three stages: (1) classify all Apps into categories; (2) understand the intent of the user; (3) find the best App among the categories. An effective search engine should at least disambiguate the results and guide the user to the appropriate category. It should compute various popularity and quality attributes obtained by analyzing social interactions in the App store and social networks. Gabrilovich [15] introduced the concept of the contextual App search that relies on action needs, that is, not getting stuff read but getting tasks done with Apps (including mobile, desktop, and browser Apps). The author noted that classical information retrieval is centered on the concept of information need, which can be satisfied by reading one or more documents, and that people rarely search Apps on the Web merely to satisfy their curiosity; rather, they search them to get things done. As a result, they suggested search engines that, instead of offering a multitude of Web pages, return Apps that directly help the users accomplish their tasks. A good functional App search is essential and there is progress in the research and development of such tools; however, the results are still unsatisfactory.

2.2. Computer Desktop Organization. The problem we are trying to resolve is somewhat similar to that of desktop file icon arrangement. Hence, before we describe our idea and exploratory study, we examine the long-standing problem of

computer desktop organization. Many desktops are cluttered with icons and the users search for documents or folders among tens of icons. During the last two decades, many studies have focused on this subject. Barreau and Nardi [16] concluded from a user behavior study that users prefer a screen region search for finding files and to use three file categories: ephemeral (temporary files), in process (files on which the user works from time to time), and unimportant (archived files currently not in use). Bergman et al. [17] showed that changing the Microsoft Windows default presentation of directory content from Details to Icons reduces retrieval time by 41%. Voida et al. [18] presented an activity-based desktop interface that uses action as the primary organizing principle. They developed a method for tagging activities and their constituent resources, using semantically meaningful labels. They found that their method allowed individuals to create and switch among activities quickly. Bruce et al. [19] conducted a user study to understand how people organize and structure information collection. They found that users frequently mentioned the need for an intelligent automated helper that would magically solve the challenges they face in the management of personal information collection.

There are several commercial tools that address the desktop icon arrangement problem. Fences [20] arranges cluttered desktop icons into shaded areas, which become movable and sizable containers; that is, it cleans up the chaos into useful groups of shortcuts, such as folders, files, and Websites. Launchy [21] is a search utility that indexes files in specific folders and shortcuts in the start menu to allow quick access to programs and browsing to find relevant files; it can launch programs, documents, and bookmarks with just a few keyboard taps. RocketDock [22] is a file launcher that provides a clean interface on which to drop shortcuts for easy access and organization, with each item completely customizable. Rklauncher and Orbit are similar launcher widgets that provide attractive, quick shortcuts for any folder on the desktop [23, 24].

Most of the solutions proposed thus far (e.g., Fences) are based on the differences between the file types, as indicated by the file extensions, such as ".pdf," ".doc," ".docx," ".ppt," and ".pptx." On a mobile device, however, there is no such option, since all Apps are of the same type (.apk). Hence, it is necessary to understand better what lies behind any App in order to find better and more compact methods of ordering them on the mobile device screen.

2.3. Apps Organizers. There are quite a few Apps developed specifically to support Apps organization, most of which require manual organization of Apps into folders or through label definition (e.g., [25]). These organizer Apps emphasize a quality interface that offers effects, such as smooth flipping and transition, which make Apps organization easy. A few organizer Apps automatically sort the installed Apps into a limited and fixed set of predefined categories (Internet, system, Apps, multimedia, etc.) based on App permissions (phone calls/SMS, network, storage, location, etc.) [26]. These Apps do not provide a proper solution to the problem, as Apps' functionality is not considered at all and there are also possible situations of uneven distribution of Apps across categories. We studied the solutions currently on the market to find the best (or most commonly used) organizer App for searching installed Apps. Most mobile devices are already equipped with the operating system's default solution provided by the device manufacturer, which allows users to search for Apps by scrolling down and up or browsing App screens by moving them to the right or the left. We searched Google Play Marketplace for an organizer App that arranges the Apps automatically and found a small number of popular organizer Apps. "Smart Launcher" [27] catalogues Apps into one of six fixed categories and presents them in a simple interface. It received a high rating (4.6 among 107,192 voters) in Google Play and has been downloaded millions of times (5,000,000–10,000,000 installation instances) and seemed to be the best that currently exists. "Everything Me Launcher" is a home replacement App (rating: 4.5; voters: 29,745; number of installation instances: 1,000,000–5,000,000) that automatically organizes Apps according to user interests [28]. "Yahoo Aviate" automatically rearranges user home screen categories (rating: 4.3; voters: 32,408; number of installation instances: 500,000–1,000,000) and suggests Apps throughout the day using its context awareness capability [29]. Because of its popularity, we used "Smart Launcher" as the reference system for our prototype (described in detail in the User Study).

3. Functionality-Based Apps Organization

Given the above, the research questions that need to be addressed, followed by our hypotheses, are as follows:

(Q1) How can the correct Apps descriptions that describe App functionality be found?

(H1) Textual description of the Apps when elicited from the App store and Apps reviews can provide a precise description of their functionality.

(Q2) How can the functionality be extracted from the App description?

(H2) By extracting verb phrases from Apps descriptions, we can identify Apps functionality.

(Q3) How can Apps be categorized accurately based on their functionality representation?

(H3) Apps functional description can be used to classify them correctly.

(Q4) How can Apps categories be organized and presented given the screen size of the mobile device?

(H4) Semantic relations between Apps' functional descriptions allow hierarchical organization of Apps in scalable abstraction, where the user zooms in and out of relevant functional categories.

We attempted to address the above challenges in a four-step exploratory study, where each step depends on the results of the previous step, as can be seen in Figure 2. These steps are Web mining for identification and enrichment of Apps descriptions; eliciting Apps functionality for

Table 1: Statistics for Apps descriptions from App store and after enrichment.

Measure/words number	App store description			After enrichment		
	Total words per raw description	Bag of words (BoW)	Bag of functionality words (BoFW)	Total words per raw description	Bag of words (BoW)	Bag of functionality words (BoFW)
Max.	1962	1393	239	3179	2019	298
Min.	9	3	1	13	4	1
Average	290.85	122.34	34.21	379.62	153.01	42.43
Median	224	92	25	291	115	31
St. dev.	231.45	103.79	29.94	306.87	132.25	37.58

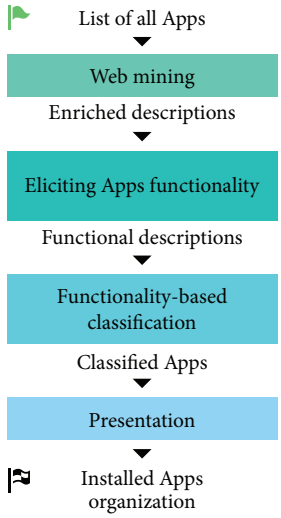


Figure 2: Functionality-based App organization process. Rectangle: steps; arrows: inputs/outputs.

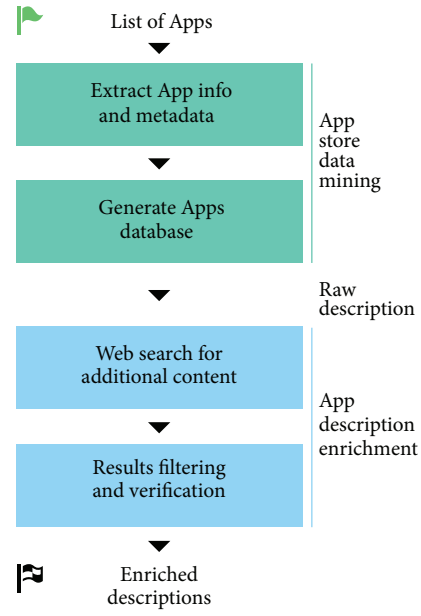


Figure 3: Apps Web mining step.

extracting the right functionality of the App; functionality-based classification for categorizing all the Apps database; and presentation for visualizing the hierarchical categorization of the installed Apps to the user. We examined each hypothesis in an individual study.

3.1. Apps Web Mining. The first step in our study was to explore the possibility of enriching Apps description by Web mining, in order to determine whether we can find useful information about Apps that can be used to identify their functionality (H1). The challenge was to identify and extract relevant information about Apps' functionality from the vast amount of information available on the Web. For this purpose, relevant textual descriptions of Apps were needed. The goal of this step was to provide sufficient raw material (representative features) for the next step. Since Apps descriptions are relatively short (as described below and presented in Table 1), a Web search was performed in order to enrich the textual App descriptions available at Google Play with the content of reviews published in blogs and elsewhere. As illustrated in Figure 3, in the Web mining step, a list of Apps was the input and textual descriptions in the virtual App store and in home pages of the Apps (when available) were searched; enriched Apps descriptions were returned.

3.1.1. App Store Data Mining. For the purpose of this study, a database of Apps was built using the Androidrank Website [30], which provides a list of Apps and their statistics (downloads and rating for each App) and additional information based on data from Google Play. For each App, there are 30 fields of data, including title, description, category, user reviews, descriptions of similar Apps, number of downloads, rating data, growth curve as a function of time, icon, and links to these data. Initially, we collected descriptions and metadata of 50,000 Apps that belong to 30 different Google Play categories. We selected two specific categories in order to focus the study; we retrieved only 6633 Apps from the "*Tools*" and "*Productivity*" categories. These categories contain Apps that function as various assistants to smartphone users. We examined 500 Apps manually and found that there are common functionalities between the two categories and that these vague categories confuse users when they upload their Apps from the App store (meaning that Apps with the same functionality are sometimes assigned to *Tools* and sometimes to *Productivity*). For example, the Apps "QR Droid" and "QR Barcode Scanner" have the same scan functionality but are assigned to different categories, *Tools* and *Productivity*,

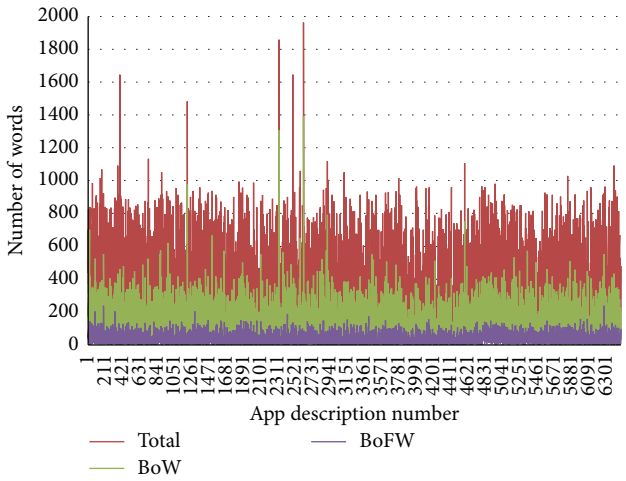


Figure 4: App descriptions word distribution.

Kayak description from Google Play

****

The #1 mobile travel App includes flight and car search, hotel search and booking, flight tracker and my trips, so you have your itinerary at your fingertips and of course, KAYAK for Android is free

Enriched by professional blogs http://intransit.blogs.nytimes.com

****

Kayak has upgraded its iPhone App, allowing users to book flights, hotel rooms and car rentals on their mobile phones

Kayak's new App was partly inspired by travelers who used the mobile App to search for flights but found it so time-consuming to complete additional reservations

Figure 5: App enriched description example (Kayak App). Red rectangles indicate verbs; blue rectangles indicate nouns.

respectively. Out of the 6633 Apps, 4917 belong to *Tools* and 1716 to *Productivity*. As this is an exploratory study, we limited the scope to two categories at this stage in order to be able to examine the idea and provide a proof of concept using a small set of vague categories. It is noteworthy that although this set belongs to only two categories, it demonstrates the App categorization problem and hence helps to demonstrate the advantages of our solution.

3.1.2. App Description Enrichment. Initial analysis of the Apps descriptions revealed a large variation of and relatively short textual descriptions (Table 1, first column from the left, and the red part of Figure 4). In Table 1, we can see that an App description contains between 9 and 1962 words and on average 291 words, out of which 122 are meaningful words (after stop words removal, bag of words, BoW for short) and 34 functionality words (BoW and BoFW columns present the results of *Apps functionality elicitation* described next). Figure 4 illustrates the variation in the data. Given the large variation in the data and the relatively small number of meaningful words, we extended the textual description with App reviews, to obtain larger textual descriptions for the analysis process.

Using the Bing search API, we searched the Web for textual descriptions of Apps and collected about 50 results for each App. The query pattern was "[App Name] + 'App'" such as "Kayak App." The results were filtered and duplicate results, as well as results from Google Play (that were already mined), were removed. For Apps that had less than 50,000 downloads, an additional check was performed to ensure that the search result describes the desired App. We verified that the results contained a link to Google Play with the name of the App that we searched.

Figure 5 illustrates the result of the enrichment process and presents an example of a description of an enriched mobile travelling App description. At the top of the figure, there is the original App description taken from the App store and below it a description taken from a blog discussing the App.

The right side of Table 1 presents the results of the enrichment process. An App description after enrichment contains on average 379 words, a 30.6% increase as compared to the initial App store description. The enrichment helped considerably to represent Apps, as illustrated in Figure 6. There are about 20% fewer Apps with very short descriptions as compared with the original descriptions.

3.2. Eliciting Apps Functionality. The second step of the study examined the possibility of extracting the functionality of the App from its textual description (H2). We obtained the enriched App descriptions as an input from the Web mining step. As can be seen in Figure 7, the second step had two stages: classical text preprocessing and functionality extraction. The goal of this step was to provide a concise functional representation for each App; it yielded two representations for each App.

The text preprocessing stage started with classical information retrieval processing: part of speech tagging was performed, stop words were removed, and then stemming was performed. As part of stop words removal App domain-specific stop words were removed as well as words related to advertising or marketing, directions for operation, technical problems, and versions updates ("free," "service," "device," "note," "best," "fastest," "versions," etc.). The stop word lists were created manually. By building a histogram of words, we could see that there are common words, whose importance in terms of understanding the functional meaning of the text is low. In some cases, mainly in titles, sequences of connected words appeared. These had to be separated (e.g., "MemoryBooster" and "ToDoList" may be split to obtain "Memory Booster" and "To-Do List"). A simple function that separates connected words was developed, based on the identification of a new word by the appearance of a capital letter in the string. Textual descriptions not written in English were automatically translated into English by the Bing translation service. This process created a BoW description for every App.

For functionality elicitation, we used the enriched BoW representation and extracted the words that indicate functionalities, creating a "bag of functional words" (BoFW) that contained verbs such as "manage," "explore," and "track" (see Table 2).

Table 2: Functionality mining (most common) from Apps descriptions. The table highlights the App store category vagueness as compared to the BoFW approach. For example, Apps numbers 4, 5, and 6 get different App categories but almost the same functionalities.

App number	App name	App category	Function 1	Function 2	Function 3
1	Super-Bright LED	Productivity	Light	Flash	
2	Clean Master	Tools	Boost	Clean	
3	TED	Education	Watch	Listen	Learn
4	Period Calendar	Health & Fitness	Track	Backup	Restore
5	My Days-Period & Ovulation	Lifestyle	Track	Forecast/Predict	Backup
6	Menstrual Calendar	Medical	Track	Forecast/Predict	Backup
7	Netflix	Entertainment	Watch	Browse	Rate
8	Calorie Counter-MyFitnessPal	Health & Fitness	Track	Count	
9	MP3 Music Download V6	Entertainment	Manage	Download	Listen
10	Mp3 Search and Download Pro	Music & Audio	Search	Download	
11	File Manager	Business	Manage	Explore	Browse
12	DU Battery Saver	Productivity	Save	Manage	
13	CM Security Antivirus	Tools	Safe	Protect	Lock
14	Draw Something	Games	Draw	Guess	

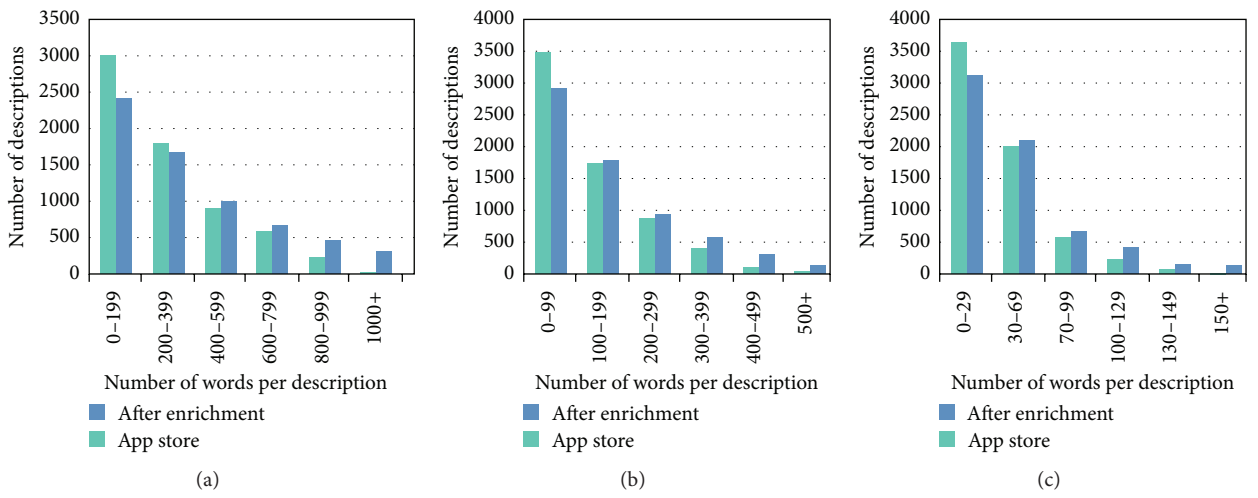


Figure 6: Comparison of App store description (blue bars) and description after enrichment (green bars). The graphs present (a) total words per description, (b) BoW, and (c) BoFW datasets.

Functionality words were elicited using a part of speech tagger (POS) [31] and Verbnet (a hierarchical domain-independent, broad-coverage verb lexicon) [32]. The POS tagger was used to identify nouns and verbs in the text that indicate topics and functionalities. As can be seen in Figure 5, red rectangles and blue rectangles mark different categories of terms, where red rectangles mark verbs such as [search] or [book] and blue rectangles mark nouns, such as [flights] or [reservations]. In addition to the part of speech tagger, we used Verbnet for identifying verbs appearing in the BoW representation. Each term became a query to Verbnet, and the verbs that were identified were added to the BoFW representation of the App. We conducted several preliminary tests with the POS tagger and Verbnet in order to estimate their reliability and found that both missed functionality terms (e.g., the verb "Sync" in the sentence "Syncs the family pill boxes" tagged as a noun by POS, while Verbnet missed "Edit" and "Share"). In addition, we found that both identified words as verbs when in the context of the App description they were not (such as "Screen" and "Schedule"). We considered whether we should simply use the union of the outputs of the two tools (and include irrelevant terms) or their intersection (and risk losing some relevant terms). Therefore, we conducted a functionality elicitation experiment (see results in Table 3).

For each of the 6633 BoW descriptions, we used the POS tagger (second column) and Verbnet (third column) and intersected the tools output (fourth column). As can be seen in the table, the average outputs of the POS tagger and Verbnet are 63 and 115 verbs, respectively. On average, 66% from POS tagger verbs (63) and 36% from Verbnet verbs (115) constitute the intersection (42). Verbnet examines each word individually, in contrast to the POS tagger, which relies on the word context, and therefore it recognizes many words as verbs when they are not. Following the above, we decided to use the intersection of the outputs in order

Table 3: Functionality elicitation.

Measure/verbs number	POS tagger verbs	Verbnet verbs	BoFW-POS tagger and Verbnet intersection
Max.	489	874	298
Min.	1	1	1
Average	63.13	114.68	42.43
Median	51	108	31
St. dev.	46.82	54.67	37.58

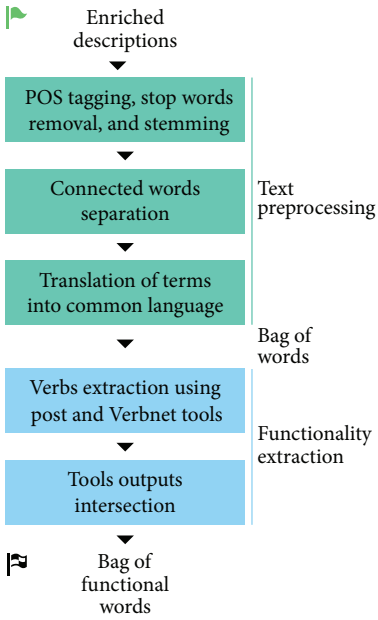


Figure 7: Eliciting Apps functionality step.

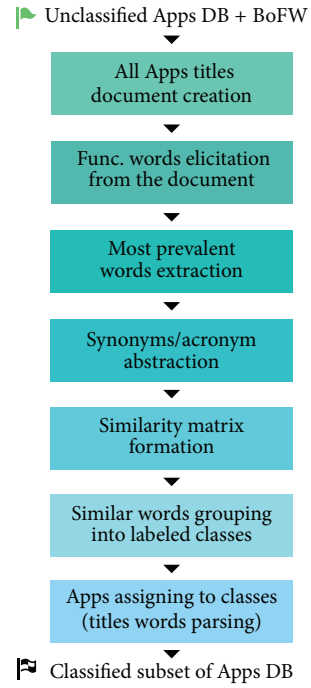


Figure 8: Apps labeling step.

to achieve an accurate BoFW representation; however, this issue needs to be studied further. The BoFW representation after enrichment contained 42 functional words on average (Table 2 shows some examples of the results of the most common functions in each App). Hence, for each App, we finally had two representations, BoW and BoFW, as an output for the next step.

3.3. Functionality-Based Classification. The input for this step was the App description representations for each (unclassified) App in the database from the eliciting Apps functionality step. In this step, we investigated how Apps can be categorized based on their representations, considering both BoFW and BoW (H3). It should be noted that we focused mainly on the BoFW representation, as one of the main ideas of the study was to categorize Apps according to their functionality. In order to categorize Apps and to evaluate the categorization, we needed a subset of labeled Apps; hence, we developed an automatic App labeling function. The functionality-based classification step returned a classified Apps database.

3.3.1. Apps Labeling. An automatic App labeling mechanism may be useful not only for experimentation, but also when thousands and millions of Apps need to be labelled automatically (as this cannot be done manually in reasonable time). Hence, we preferred to suggest such a mechanism rather than to manually label tens or even hundreds of Apps only for experimentation. For labeling, we took advantage of the App titles' unique features, when such existed. As most of the Apps deal with specific functions, but relatively few titles indicate the App functionality explicitly, we relied on the fact that some Apps do include their exact functionality in their title. The App labeling step (see Figure 8) focused on identifying the functionality of the Apps and then created groups of Apps with similar functionality. That is, a subset of classified Apps (training set) from the entire App database was created. Later, we used this subset to classify other unlabeled Apps (see Section 3.3.2).

In order to label the Apps, we extracted the titles of all the Apps in the database and created one document containing all the title words. We retained the pointers from the title terms to the original Apps they represent. Then, we elicited title functionalities by applying the *functionality elicitation* process described previously. As a result, we obtained two representations of the title words collection: titles BoW and titles BoFW. We selected the most prevalent words from each representation to allow later grouping of the original Apps. After examining the words distribution, we decided

to gather all the words that appear at least 20 times in the App titles. This threshold was selected arbitrarily in order to filter out title words that represent many Apps and therefore will require further tuning in the future. Synonyms were abstracted using Bing synonyms [33] to create a standard, and more abstract, representation. Words that were not found in the Synonyms API, such as technological words or acronyms, were combined into groups manually. For example, the words "IMs," "mms," and "sms" were combined into one group and abstracted to the verb "Chat" or to the noun "Message," according to the representation. Hence,

$$\begin{aligned}\text{OverlapsSum}(W_1, W_2) = {} & \text{OverlapCnt}(\text{Gloss}(W_1), \\ & \text{Gloss}(W_2)) + \text{OverlapCnt}(\text{Gloss}(\text{Hype}(W_1)), \\ & \text{Gloss}(\text{Hype}(W_2))) \\ & + \text{OverlapCnt}(\text{Gloss}(\text{Hypo}(W_1)), \\ & \text{Gloss}(\text{Hypo}(W_2))) \\ & + \text{OverlapCnt}(\text{Gloss}(\text{Hype}(W_1)), \text{Gloss}(W_2)) \\ & + \text{OverlapCnt}(\text{Gloss}(W_1), \text{Gloss}(\text{Hype}(W_2))).\end{aligned} \tag{1}$$

In order to group similar words (after synonyms abstraction) into meaningful clusters, we generated a similarity matrix using an integrated similarity function between each pair of prevalent words. The function integrated the extended gloss overlaps method [34] with semantic distance. This method relies on Wordnet [35] and Verbnet. It counts the number of words (OverlapCnt(), (1)) that are shared between two glosses (gloss = explanatory note of a word in the dictionary) (Gloss (word) returns the explanatory note or translation of the word). The larger the overlap between the glosses is, the more related the meanings are. However, the average length of a gloss in a dictionary is only seven words and therefore we extended each gloss. The extended gloss technique expands the glosses of the prevalent words being compared (W_1, W_2) by concatenation glosses of other words that are related to them. This word relatedness was found according to the dictionary hierarchy type-of (word whose semantic field is included within another word) relations, such as hypernyms and hyponyms (Hype(W_1) and Hypo(W_1)). We applied OverlapCnt() function five times to each pair of glosses and obtained the overlap count between each pair of words. Then, we computed the sum of the results (the total overlap words number of the extended glosses between the prevalent words being compared) as represented by OverlapsSum(W_1, W_2) function in (1). Hence,

$$\begin{aligned}\text{If} \quad & (\text{Synonyms}(W_1, W_2) == \text{true}) => \\ & \text{GlossSim}(W_1, W_2) = 1\end{aligned} \tag{2a}$$

$$\begin{aligned}\text{Else} \quad & \text{if OverlapsSum}(W_1, W_2) > 0 => \\ & \text{GlossSim}(W_1, W_2) \\ & = 1 - \frac{1}{\text{OverlapsSum}(W_1, W_2)}\end{aligned} \tag{2b}$$

$$\text{Else} \quad => \text{GlossSim}(W_1, W_2) = 0. \tag{2c}$$

Equations (2a), (2b), and (2c) give a similarity value for each pair of words according to the extended gloss overlaps measure. If the pair of words are synonyms according to Wordnet, GlossSim(W_1, W_2) function returns 1, as can be seen in (2a). In order to normalize the value in the range [0-1], we subtract from one the multiplicative inverse of the glosses overlaps sum, as shown in (2b). If there are no glosses overlaps between a pair of prevalent words, then GlossSim() function returned 0, as can be seen in (2c).

We combined this measure with the semantic distance measure implementation [36] that is based on the hierarchical structure of Wordnet/Verbnet. These hierarchies are treated like undirected graphs and each node consists of a word and its synonyms (synonyms set, synset). This representation allows the length of the path between any two words (or synsets) to be measured using SimDist() (see (3)). This is the number of nodes (synsets) in the hierarchical path between the pair of words (or synsets) using PathDist() (see (3)), that is, going up the hierarchy until the closest mutual ancestor and back down, counting all nodes on the way. The shorter the path from one node to another is, the greater their similarity is. The distance between two synonyms is 1 since they have equal meaning (regardless of the path length). The dictionaries allowed multiple inheritances, and therefore, if more than one path exists between two nodes (words or their synonyms), the shortest path is selected. Hence,

$$\text{SimDist}(W_1, W_2) = \frac{1}{\text{PathDist}(W_1, W_2)}, \tag{3}$$

$$\begin{aligned}&\text{IntSimScore}(W_1, W_2) \\ &\quad = \frac{\text{GlossSim}(W_1, W_2) + \text{SimDist}(W_1, W_2)}{2}.\end{aligned} \tag{4}$$

The extended gloss overlaps measure and the semantic distance measure are averaged (at this stage, we decided to weight them equally), as can be seen as function IntSimScore() (see (4)). This function integrates the measures in order to find accurate similarity between each two prevalent words of App titles. Using the integrated similarity function (range [0-1]), we generated a similarity matrix in order to group words (verbs) under a common word (verb).

Every pair of words from the prevalent words was compared using the integrated similarity function. Table 4 presents the similarity matrix of the extracted prevalent words of Apps titles; because IntSimScore(W_1, W_2) = IntSimScore(W_2, W_1) only half the matrix is filled. Using the similarity matrix, we can find pairs of similar words. In order to find meaningful clusters of words, we group together pairs that contain the same word (at least one). This can be seen as finding connected components in an undirected graph using an adjacency matrix. After examining the matrix, we found that the suitable threshold is IntSimScore$(W_1, W_2) > 0.87$ because it provided the best separation between conceptual clusters in our case. It created granular and balanced groups without overlapping in the word collection; however, this threshold may be refined after future research. The label of

Table 4: Similarity matrix of prevalent words of Apps titles. The letters e and d indicate the generated clusters ("time," "flashlight") according to similar words (IntSimScore(W_1, W_2) > 0.87).

	Clock	Call	Flashlight	Phone	Screen	Alarm	Auto	Light	Timer	Monitor	Time	Flash	Voice	Volume	Recorder	Bright	Sound	Torch	Car
Clock																			
Call	0.57																		
Flashlight	0.64	0.57																	
Phone	0.64	1[a]	0.64																
Screen	0.67	0.59	0.67	0.62															
Alarm	0.96[a,e]	0.73	0.74	0.71	0.78														
Auto	0.55	0.57	0.55	0.6	0.56	0.63													
Light	0.74	0.67	0.9[b,d]	0.71	0.78	0.78	0.63												
Timer	0.91[b,e]	0.57	0.64	0.64	0.67	0.87[b,e]	0.55	0.74											
Monitor	0.67	0.63	0.67	0.89[b]	0.9[b]	0.78	0.7	0.78	0.57										
Time	1[a,e]	0.4	0.13	0.55	0.18	0.5	0.13	0.6	0.9[b,e]	0.18									
Flash	0.67	0.67	0.9[b,d]	0.78	0.7	0.73	0.6	0.9[b,d]	0.67	0.78	0.86[c]								
Voice	0.4	0.6	0.4	0.4	0.5	0.67	0.4	0.67	0.62	0.62	0.5	0.77							
Volume	0.53	0.4	0.53	0.4	0.67	0.44	0.53	0.67	0.53	0.59	0.77	0.44	0.77						
Recorder	0.63	0.67	0.63	0.82[c]	0.67	0.75	0.63	0.75	0.71	0.82[c]	0.18	0.82[c]	0.62	0.62					
Bright	0.12	0.57	0.12	0.33	0.15	0.36	0.12	1[a,d]	0.2	0.15	0.6	0.93[a,d]	0.62	0.67	0.2				
Sound	0.83[c]	0.73	0.25	1[a]	0.33	0.67	0.25	0.67	0.33	0.33	0.83[c]	0.73	1[a]	0.83[c]	0.33	0.67			
Torch	0.67	0.6	1[a,d]	0.74	0.7	0.78	0.57	0.95[a,d]	0.67	0.74	0.2	0.86[c]	0.6	0.56	0.78	0.12	0.27		
Car	0.6	0.63	0.6	0.67	0.62	0.71	1[a]	0.71	0.6	0.76	0.15	0.67	0.44	0.59	0.71	0.13	0.29	0.63	

[a]Very high similarity < 0.90; [b]high similarity < 0.85; [c]medium similarity < 0.80; [d]flashlight cluster; [e]time cluster.

each cluster was the most prevalent word in each cluster. If there was more than one, then the more general word was selected (hypernym). For example, according to Table 4, the pairs (clock, alarm), (clock, timer), (time, clock), (time, timer), and (timer, alarm) pass the threshold and hence constitute a cluster (marked as a thick box border) that includes related terms such as "Clock," "Alarm," "Time," and "Timer." The cluster was labeled "Time," as this is the most common term in the cluster. Following this process, App titles such as "Super Analog Clock," "Smart Alarm Free," and "Kitchen Timer" can be assigned to the "Time" cluster.

In order to evaluate this labeling/clustering method, a "gold standard," a set of manually classified Apps, was built. We took prevalent title words (that had more than 15 occurrences in Apps titles). Each selected word represents at least 15 Apps that include the word in their title and therefore has a high probability to represent the functionality of the Apps. We extracted 143 title words from the BoW collection and 64 title words from the BoFW collection. We took these prevalent title words and manually built 19 BoW and 12 functional clusters. The clusters were created by 5 experts according to the semantic relatedness between the words (discretionary), in order to create large groups (at least 80 Apps in each group) of Apps (descriptions) that have the same or similar functionality. Each expert grouped the words separately. The final cluster set was defined by combining the results of the experts, and in cases where there was no agreement, majority voting was used. These clusters' set served as a "gold standard." Then, we applied the Apps labeling (titles clustering process) described above to the same title words. This process resulted in 21 BoW and 14 BoFW clusters.

Table 5: Similar words (semantic) clustering measures.

Measure	Precision	Recall	F-measure	Rand index	Accuracy
Result	0.752	0.597	0.665	0.512	0.703

In order to evaluate the process results, we compared the machine-generated clusters with the predefined manual clusters. We labeled the automatic clusters based on the majority word members "majority voting" of the manually clustered words within each automatically generated cluster. Table 5 shows the similarity of the clustering. The precision value (0.752) is reasonable (all the clustering measure formulas are according to [37]); however, according to the unsatisfactory results of the F-measure (0.665) and the rand index (0.512), we conclude that although the automatic process creates good clusters, there is room for further research regarding the semantic clustering process.

We noted in some cases that the words did not always represent the common meaning according to Wordnet. For example, the method found that Auto and Car are highly similar (see Table 4); however, in the Apps world, the most common meaning of Auto is "automatic." Therefore, in the future, the process should be further adapted to the specific characteristics of the mobile Apps domain. In addition, Wordnet does not include technological words such as QR and Wi-Fi. In cases such as these, it needs to compute semantic relatedness using external dictionaries such as Wikipedia, whose entries are updated frequently in general and its technology entries in particular [38]. Given the insufficient evaluation results, we used the predefined manual clusters and labels.

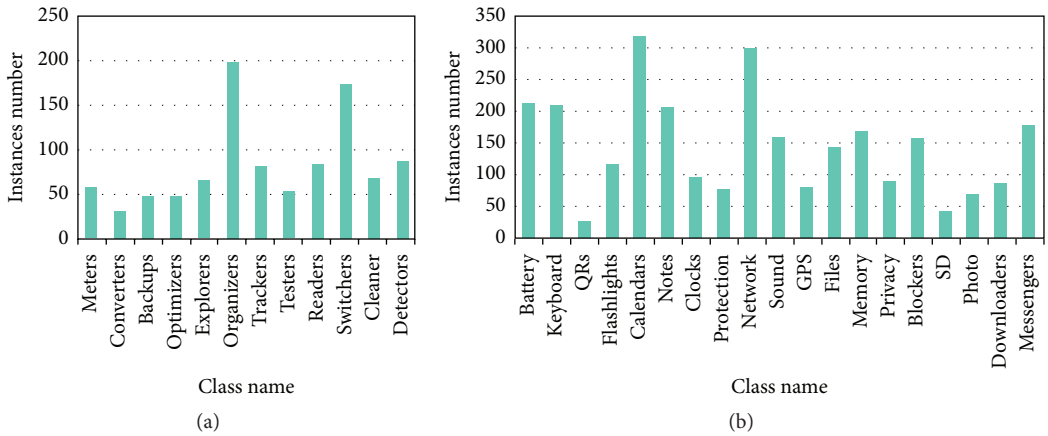


Figure 9: Twelve functional classes (a) and 19 BoW classes (b) instance distributions.

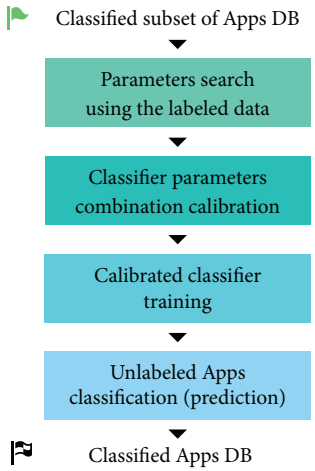


Figure 10: Apps classification step.

Once we had a subset of labeled clusters (classes), some of the original Apps were assigned to the classes based on the words appearing in their titles (recall that pointers were retained from the title words and their abstraction to the original Apps). Out of 6633 Apps, we succeeded in labeling 2747 Apps automatically into 19 BoW classes and 1003 Apps into 12 BoFW classes (see Figures 9(a) and 9(b), resp.). Namely, 41.4% of Apps were successfully labelled (BoW) using the App labeling (title-based clustering approach). At this point, we were ready to explore the possibility of automatic Apps classification (for classifying the remaining 58.6% of Apps into the classes).

3.3.2. Apps Classification. An initial feasibility study explored the possibility of functionality-based clustering using short textual descriptions. The study compared three clustering algorithms and showed some difficulties in cluster overlap, cluster labels, and cluster diversity. The main conclusion was that the descriptions of Apps seem to be sufficient for clustering them, but research is required in order to obtain good accurate results [11]. We extended the clustering experiment and tested four clustering algorithms using both BoFW and BoW representations. Unfortunately, the accuracy results ranged from 12.65% to 20.94%. The overall poor results of the clustering led us to conclude that clustering is not a good solution in this case and therefore we turned to exploring a supervised approach.

The classification step is described in Figure 10. We used the labeled Apps subset BoW and BoFW TF $*$ IDF vector representations (Term Frequency-Inverse Document Frequency [39]) and tested classification algorithms. We used the LibShortText [40] library for Apps description classification. This open source library is designed for short-text classification. It supports effective text preprocessing and fast training/prediction and also provides an interactive tool for error analysis.

The library's optional parameters are specific preprocessing actions, selection of feature representation, classifier, and normalization. Specific preprocessing parameters are tokenization (unigram or bigram) (all the contiguous sequences of two words from given text), stemming, and stop words removal "SPR". The feature representation parameters are binary (if a feature appears at least once in the text, it receives 1; otherwise, it receives 0), word count (the number of times a feature appears in the text), term frequency ("TF," the number of times a feature appears in the text divided by the total number of features), and TF $*$ IDF. There are four classifier options: logistic regression [41], multiclass SVM proposed by Crammer and Singer [42], L2-regularized L1-loss, and L2-regularized L2-loss SVMs [43]. There is an option to apply instance-wise normalization to unit vectors before training/testing.

We examined all the 512 parameters combinations through a "brute-force" parameter search. We applied it to the BoFW and BoW representations of the enriched descriptions. We used a classic cross validation partitioning technique where each partition divided the labeled class instances (see Figure 9 for instance number per class) for the training and prediction test. For each combination of parameters, we examined the prediction test accuracy.

The best classification results are presented in Table 6. For classification by BoFW, the best classifier was L1, which achieved prediction test accuracy of 80.0%; the poorest

Table 6: Functional and BoW classification calibration by test accuracy comparison (best results only).

Dataset	Functional				BoW			
Calibration/classifier	Crammer and Singer	L1	L2	Logistic regression	Crammer and Singer	L1	L2	Logistic regression
Accuracy (correctly classified)	79.16	**80**	78.33	77.5	**88.94**	88.42	**88.94**	88.42
Feature representation	TF/IDF	TF/IDF	TF/IDF	TF/IDF	TF/IDF	Word count or TF	Word count	Word count
Feature preprocess	Stemming, bigram	No SPR, stemming, and bigram	No stemming, unigram	Stemming, bigram	No SPR, no stemming, and unigram	Stemming	No SPR, stemming, and unigram	Unigram
Normalization	With	With	With or without	With	With	With or without	Without	With or without

Table 7: Trained classifier (unlabeled Apps) evaluation.

Measure	Precision	Recall	F-measure	Rand index	Accuracy (correctly classified)
Functionality dataset	0.825	0.847	0.826	0.741	82.5
BoW dataset	0.879	0.8994	0.889	0.798	87.88

classifier was logistic regression. In general, better results were achieved with bigram (the claim is true for three of the four classifiers). TF $*$ IDF feature representation appeared to be the best representation, probably because there are repeated functionality words. The binary representation is less accurate. Normalization improved the results in most cases. Analysis of the BoW-based classification results showed that the best result was achieved by the Crammer and Singer and L2 classifiers with an accuracy of 88.94%, applying TF $*$ IDF or word count representations and with unigram preprocessing. In most of the classifiers accurate preprocessing results were achieved with unigram (the claim is true for three of the four classifiers). The feature representation with higher results is word count (most frequently), while binary representation obtained lower results. Normalization did not improve the results significantly. The results described above are high and satisfying.

After we found good classifiers for each BoW and BoFW dataset with the right parameters combination calibration that generate accurate predictions on the labeled datasets, we classified the unlabeled Apps that remained in the Apps database (3886 Apps) using these trained classifiers. At the end of the classification process, all the App databases (6633 Apps) were tagged and catalogued into the classes. In order to evaluate the effectiveness of this unlabeled Apps classification, we randomly sampled 310 tagged Apps that were not labeled in the *App labeling* phase but tagged only with the trained classifier. We read the original App description and labeled it manually. Then, we compared the automatically assigned classes to the manually assigned classes. Table 7 shows the results of this evaluation. As can be seen, the measures for both datasets (BoFW and BoW) are quite high: rand index (0.741, 0.798), F-measure (0.826, 0.889), and accuracy (82.5, 87.88), respectively.

3.4. Presentation. As we have already seen, for *computer desktop organization*, there are already quite a few solutions. However, one fundamental difference between a desktop and a mobile device is the ratio between the number of icons and the screen size. While on the desktop there is usually enough space and icon arrangement can be flat and one-dimensional, on the mobile, because of its limited screen space, different solutions are needed, which may be hierarchical or multidimensional. Icons should possibly be grouped hierarchically into categories and presented to the users in a way that allows an easy search. The large number of expected Apps and the small size of the screen make such a solution a necessity (see Figure 1). To address this challenge, we suggest arranging the Apps hierarchically in scalable abstraction according to a functionality-based taxonomy (H4). The goal of this step is to provide hierarchical grouping of the Apps installed on the user's device. It starts with Apps grouping and creates a taxonomy of Apps that is used for organizing the Apps installed on the user's device (see Figure 11 for the sequence), thus providing personalized ordering.

3.4.1. Functionality-Based Taxonomy. In order to present the Apps hierarchically and in a logical order, we started with the App classes created in the previous step (12 BoFW labels and 19 BoW labels). However, in the future, the problem will be exacerbated as the number of Apps will increase, unlike our experimental scenario, and the algorithm needs to scale up and address millions of Apps that exist in App stores and probably thousands classes. Note that in the previous step we used a generic Apps grouping mechanism that is now refined before being used for personalized ordering. Using Wordnet and Verbnet, we searched for abstract labels that combine some of the classes in order to form a hierarchy of classes so that the upper level of the hierarchy fits onto the small screen of the mobile device. The semantic relations such as hypernym and Holonymy allowed us to find the closest common ancestor labels (abstract category label) for groups

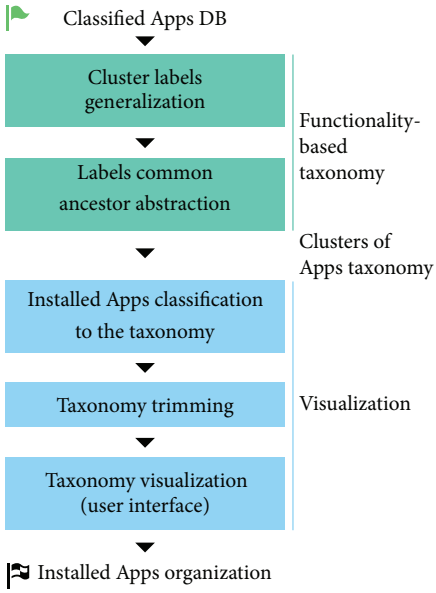


Figure 11: Presentation step.

of class labels. In Verbnet, the verb hierarchy is built-in and there are verbs, classes of verbs, and types of a number of classes (such as Searching => Investigate => Test). These relations enabled us to define a hierarchy of class labels, as illustrated in Figure 12, where the leaves represent Apps that are associated with the original classes and the other tree nodes represent abstractions. Most of the hierarchy deals with part-whole and generalization relationships (meronomy or partonomy) and only a few remain discrete sets. This organization (taxonomy) was used later in the *visualization* phase to present the installed Apps in an intuitive order that makes it easier for the user to search.

To evaluate the taxonomy creation method, we used the clusters created during the *App labeling* phase. Recall that we created 12 and 19 classes, probably too many to be presented on a small screen of a mobile device. We applied the method described above to these clusters labels. We manually evaluated the taxonomy that was generated by this process and found inaccuracies and overgeneralizations (e.g., the lowest common hypernym between "Clock" and "Calendar" is "Device"). The main drawback was that we attempted to create a taxonomy out of the labels, ignoring any additional knowledge that may exist in the App descriptions or external knowledge source such as the Web [44]. Hence, it seems that we need to develop a "universal understanding" mechanism, as Wu et al. [45] proposed, that generates a taxonomy that relies on probabilities, to model ambiguous, inconsistent, and uncertain information. Our conclusion is that more research is required to achieve better results. As the results were not satisfactory, we created the taxonomy manually for the next step. For example, in the lower part of Figure 12, a BoW-based taxonomy can be seen; we can see that the "Time" class has subclasses such as clocks, calendars, and notes. In the BoFW-based taxonomy (diagram (a)), we can see that the "Managers" class has subclasses, such as controllers and organizers.

3.4.2. Visualization. Having created generic hierarchical taxonomies, we now turn our attention to the individual user. Every user (owner) has different Apps installed on his/her mobile. Therefore, the generated functionality-based clusters must be adapted for each user. The suggested solution is to consider the Apps installed on the user's device, classify them according to the generic Apps hierarchical taxonomy, and then trim long and empty branches of the taxonomy such that a balanced tree of categories with Apps is created that can nicely fit onto the small screen of the mobile device. First, only the Apps that are installed on the user's device are mapped to the generic taxonomy (Apps Filtering) on the lowest (leaf) level of the taxonomy (see Figure 13; we move from (1) generic taxonomy to (2) individual taxonomy). Then, considering the numbers of Apps in the individual nodes, empty nodes and paths are pruned in order to provide a compact representation (Empty Branches Pruning). That is, classes and branches without associated Apps are removed from the hierarchy (step (3)). The nodes that remain are merged to present classes that are meaningful to the user (small categories merging). For example, if there are only a few Apps in each sibling class (see Figure 13, steps (3), (4), and (5), e.g.; to-do lists and Notepad classes from the BoW taxonomy both contain tree Apps), these small classes (smaller than the defined threshold, e.g., four Apps per class in our case) are merged and their Apps become part of the common ancestor class (step (4)). Furthermore, in cases where there were three classes ordered hierarchically on the same branch (grandfather => father => grandson classes path), the grandson class can be moved up one level in the hierarchy by removing the parent (father) class and connecting directly to the grandfather class (grandfather => grandson classes path, such as the transition between steps (4) and (5), branches collapsing). The goal of this process is to achieve a trimmed taxonomy with balanced distribution of Apps over Apps categories that are present on the user's mobile device, for easy presentation.

As a result, the Apps are organized into the minimal levels of hierarchy that are needed for compact presentation (depending on the number of installed Apps). Within the top layer, there are a few functional classes (that can fit onto the screen of a mobile device), while the layers below it contain related classes based on the taxonomy. The lower (leaf) level contains the Apps. The results can be presented to the user as follows, according to the default setting of the smartphone: classes are presented in functional folders, and BoW classes appear as frames within each cluster, as illustrated, for example, in Figure 14. If there are too many folders (say, over 16, the threshold of device folders per screen in some mobile phones), similar folders are grouped into fewer, more abstract, folders.

4. User Study

Thus far, we have examined the individual steps of our suggested approach and showed that we succeeded, to a certain extent, in addressing each of the individual research challenges. We note that although we showed the feasibility of the idea, further research remains to be done to improve

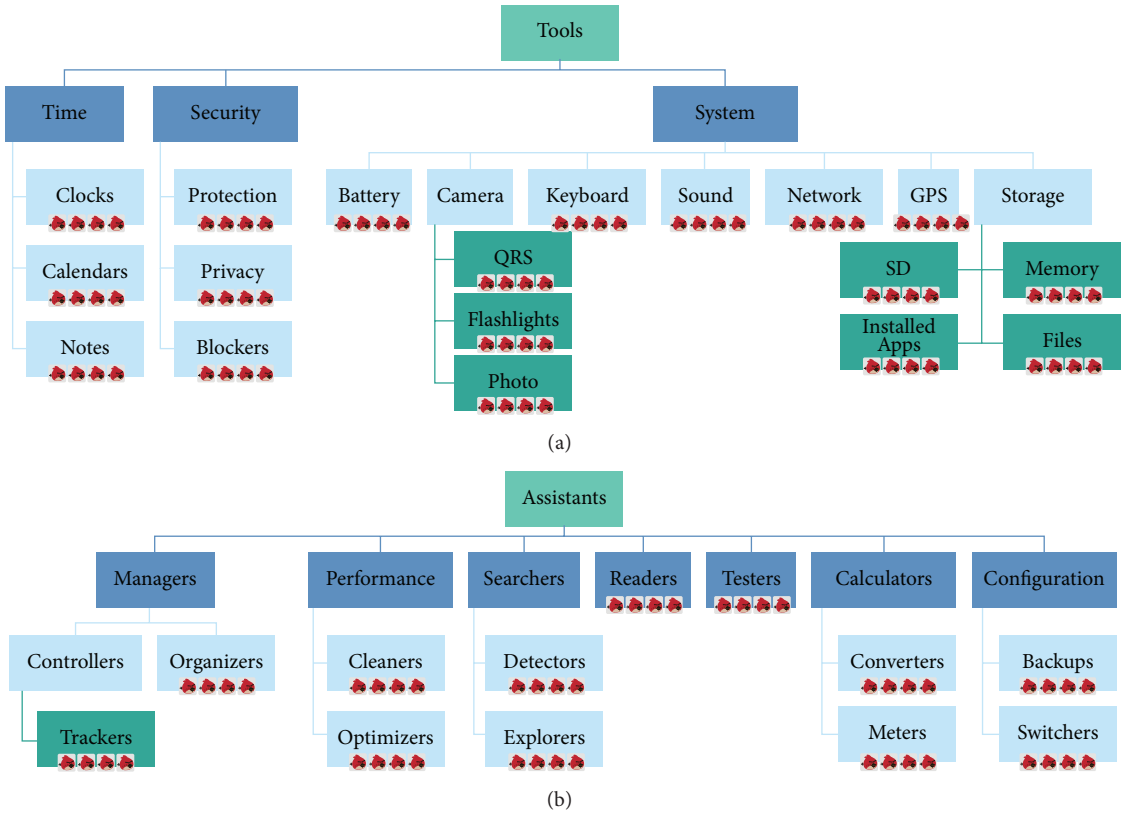
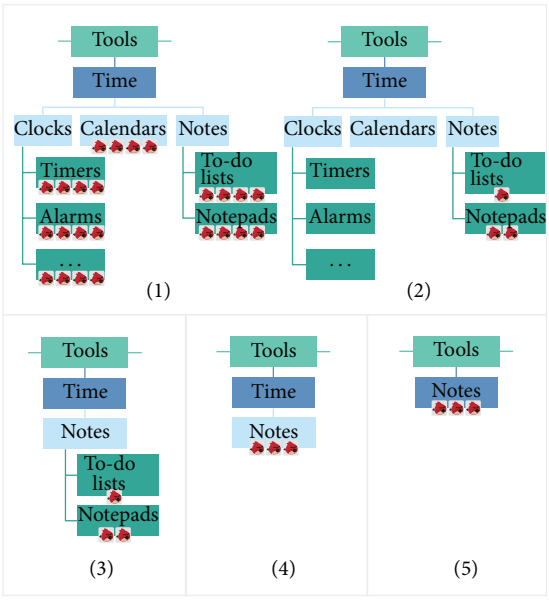


FIGURE 12: BoFW-based (diagram (a)) and topic BoW-based (diagram (b)) taxonomies for Tools and Productivity categories. Each color illustrates the depth level of the class tree. The leaves illustrate class associated Apps.

the individual steps. Following the individual experiments described in Section 3, we built a prototype system that simulates the end result of the entire process. We used these results in a user study exploring how functionality-based classification is perceived by the end users. We compared the personalized hierarchical functional categorization, "Our Approach," with a fixed categorization referred to as a "Reference Approach." We simulated the "Smart Launcher" App, described in the Apps Organizers, as a reference (six predefined and flat categories), as it is currently the most popular automatic App organizer.

4.1. Design. For the study, we developed two user interfaces, one for Reference Approach (Figure 15(b)) with six fixed categories (following the Smart Launcher interface) and another for Our Approach (Figure 15(a)) with dynamic hierarchical categories (our solution). The interface design is intended to simulate a smartphone screen, as well as a category tree, in terms of the number of icons presented on the screen and their relative size. Hence, the number of icons on the screen is limited to that which may be presented on a mobile device screen (we selected 16 icons per screen (configurable), as a representative number, the number of icons presented on the Nexus 4 screen). The user study focused on examining the icon organization technique. We used a desktop application that presents the two options in a similar way, to emulate the characteristics of the small mobile device screen. Tapping on the category tree, "category click," resulted in the presentation of the relevant Apps (icon and name for each App) on the screen. The user was presented with the name of the App (Figure 15(a), "Timer") he/she needed to find, a short description (Figure 15(a), "Stop Watch"), and an icon (as can be seen in the upper right part of Figure 15). Using these data, the user could navigate the category tree searching for the desired App among the Apps on the screen. When the user found the right App he/she clicked on it "Screen Clicks". The interfaces were developed using CodeProject open source software [46].

4.2. Data and Procedure. We conducted a counterbalanced within-subject experiment comprising four stages. Each stage simulated a device with a specific number of installed Apps; the users needed to find a specific App using the device interface. In the first experimental stage, 40 Apps were installed, then 90 Apps, 140 Apps, and finally 240 Apps. Each stage included Apps that were used in the previous stage as well as new ones. In order to balance the learnability in the transition between the approaches, half of the users started from Reference Approach and transitioned to Our Approach and half performed the transition between the approaches in the opposite order. To decrease the learning effect between approaches, we used two test sets, one for each approach across the stages (480 Apps in total). It should be noted that although there were different Apps in each set, the functionality and the order in which they were presented to the user were identical. The categories



(1) Initial state and parameters:
TaxTree-generic taxonomy tree (GTT) with all Apps assigned as category nodes (*CatNode*)
AppsPerFolderThreshold_Upper = 16;
AppsPerFolderThreshold_Lower = 4;
(2) Creating personalized Apps tree:
Input: *TaxTree* and list of installed Apps on the user's device
Procedure: assign only the installed Apps to the relevant lowest *CatNode*
Output: *PersonalTaxTree*-*TaxTree* with installed Apps as leaves (*AppLeaf*) that associate with GTT *CatNodes;*
(3) Empty subtrees pruning:
Input: *PersonalTaxTree*
Procedure: remove subtrees having no Apps as leaves (applying depth-first approach recursively)
(3.1.) If *CatNode* has no *AppLeaf* in its hierarchy:
(3.1.1.) Remove *CatNode* from *TaxTree*
Output: *TrimmedPersonalTaxTree*:
(4) Small categories merging:
Input: *TrimmedPersonalTaxTree*
Procedure: for each *CatNode* that has at least *AppLeaf* in *TrimmedPersonalTaxTree* add it to a list of *AppLeavs*:
(4.1.) If *CatNode* has less *AppLeavs* than *AppsPerFolderThreshold_Lower*
(4.1.1.) If CatNode's ancestor AppLeavs + CatNode's AppLeavs is less than *AppsPerFolderThreshold_Upper*
(4.1.1.1.) Connect the *AppLeavs* to the *CatNode ancestor*
(4.1.1.2.) Remove *CatNode* from the tree and from the list of category nodes
(4.1.1.3.) Add the ancestor to the list of leaf category nodes
(4.1.2.) Else remove *CatNode* from the list of category nodes
Output: *CompactPersonalTaxTree*:
(5) Branches collapsing:
Input: *CompactPersonalTaxTree*
Procedure: remove long, empty paths (applying depth-first approach recursively)
(5.1.) If *CatNode* has a single child (and it is not an App)
(5.1.1.) Link the single child to CatNode's ancestor
(5.1.2.) Remove *CatNode* from *TaxTree*
Output: *ReducedPersonalTaxTree*

Figure 13: Taxonomy trimming process. (1) Initial state of the taxonomy (subtree). (2) Creating personalized Apps tree. (3) Empty subtrees pruning. (4) Small categories merging. (5) Branches collapsing.

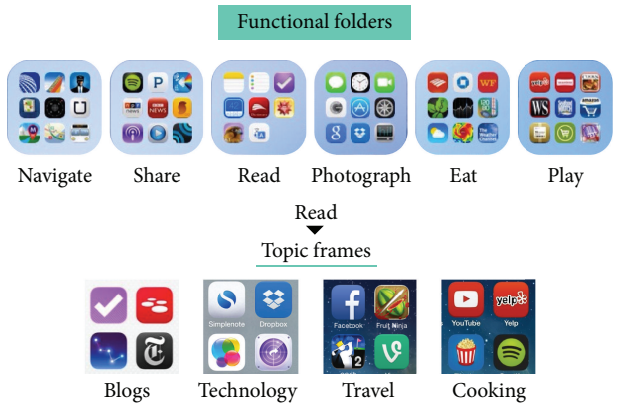


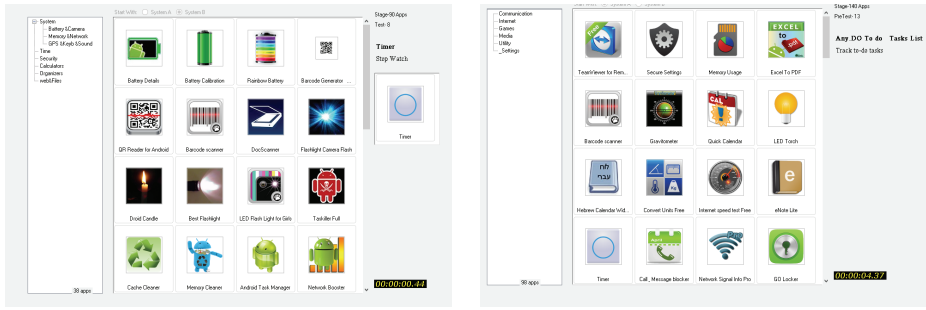
Figure 14: Process output. Upper slides indicate functional folders (BoFW); lower slides indicate "Read" folder Apps clustered by (BoW) topic (the icons are for demonstration only).

generated by the prototype were the same for all participants but depended on the configuration. This design created four different configurations in the experimental platform (Reference Approach [A] with set 1 and then Our Approach [B] with set 2 = A1B2, A2B1, B1A2, and B2A1). It should be noted that while these configurations were dynamically generated, the App categories were the same for each configuration in order to simulate the correlation between approaches and Apps set to the specific categories derived from them.

In each one of the four stages (40, 90, 140, and 240), there were six tests in each of which the participants were asked to find a specific App. During each test, data were collected, including time from the start of the test until the App was found and clicks on the category tree. Only after all the stages in one approach were completed did the participant move to the next approach.

Prior to each stage, the participants were asked to explore the category tree and the Apps associated with each category carefully, giving them an opportunity to become familiar with the Apps, because smartphone users are familiar only with the Apps installed on their own device (the same opportunity was given for the two applications, to avoid any bias).

In each test, the participants saw the App's icon for half a second in order to simulate a real situation where the user knows the desired functionality but only vaguely remembers the icon. During a search, when the user encounters an icon



(a) (b)

Figure 15: Experimental environment. Interfaces differ only in the category tree. Reference Approach with six fixed categories (image (b)) and Our Approach with hierarchical categories (image (a)).

he/she immediately remembers that this is the icon for which he/she was searching.

4.3. Participants. Forty students of the University of Haifa, mainly from the Information System Department, who were paid for their time, participated in the study. All had been smartphone users for at least a year, having tens of Apps on their devices. 55% were males and 45% females, aged between 20 and 30, while the mean age was 25.38 (SD = 2.26). 37.5% were iOS users, and the rest (62.5%) used Android-based phones. 67.5% of the participants reported they had fewer than 60 Apps installed on their mobile; 22.5% had 60–100 Apps, 5% had 100–150 Apps, and 5% had more than 150 Apps. 62.5% of the participants reported that their App search function is App paging/scrolling (operating system default), 10% used a dedicated launcher, and the rest (27.5) reported another method, such as manually created folders and manual home screen icon arrangement. Figure 16 presents the participants' characteristics.

4.4. Hypotheses

(A) We hypothesized that our suggested method for Apps categorization would be more effective than the alternative method. We assumed the following:

(1) It will take less time to find a specific App with Our Approach than with Reference Approach.

(2) It will take fewer category clicks with Our Approach than with Reference Approach.

(B) We hypothesized that our suggested method would be preferred by the users to the alternative methods. We assumed the following:

(1) Users will consider Our Approach a better search method than Reference Approach.

(2) Users will want to adopt Our Approach more than other App search methods (Reference Approach, manual folders, home screen icon arrangement, and OS default search method).

4.5. Results. To evaluate the efficiency of the approaches, we compared them in terms of the time it took the participants to find the right App and the number of clicks on the category tree. We conducted multivariate analysis applying the independent variables approach and number of Apps using SAS statistical software [47]. There are two degrees of freedom, approach type (our solution, competitor) and stage, namely, the number of Apps (40, 90, 140, and 240). We collected 320 observations (40 participants $*$ 2 approaches $*$ 4 stages). To determine the appropriate analysis, the measures were tested, since normal distribution of the data was not found (by a Kolmogorov-Smirnov test) within the intersection of every stage and type of approach. Generalized Linear Mixed Models (Glimmix) were applied after log transformation.

4.5.1. Interaction Time Comparison. Our hypothesis was that our proposed solution would be more efficient than the reference system, and indeed, in all four stages, Our Approach outperformed Reference Approach, as it enabled the users to find the Apps faster, as shown in Table 8 and Figure 17. Figure 17 visualizes the differences in terms of search time medians in ms until the user found the right App using Our Approach and the Reference Approach. In general, Our Approach enabled users to find the Apps faster than Reference Approach. The median of search time gradually increased with the number of Apps presented to the participants in both cases. The differences between the two approaches were found to be statistically significant according to the Glimmix with Bonferroni correction ($P < 0.05$), except for the case of 240 Apps. The increase in time was also found to be statistically significant ($P < 0.05$ for both). In addition, we can see that the green boxes (=quartile 2 + quartile 3) are smaller than the blue boxes of Reference Approach, indicating that the distribution of users' interaction time is denser. Only with 240 Apps was the max. result (upper whisker) higher than Reference Approach.

4.5.2. Category Clicks Comparison of Approaches. We also compared the number of user clicks in the two cases and found that Our Approach outperformed Reference Approach in all stages, as it enabled the users to find the Apps with fewer category clicks, as illustrated in Table 9 and Figure 18.

Table 8: Comparison of approaches according to the time (ms) in each stage until the user finds the right App by various measures.

Measure/stage	Reference Approach				Our Approach			
	40	90	140	240	40	90	140	240
Mean	11674.1	19131.51	24258.4	27189.9	6997.71	11084	15967.1	20329.5
Median	9359.3	16543.1	23371.7	25098.1	5727.1	10244	13787.9	18160.5
St. dev.	6471.32	8458.89	9917.89	10830.1	5410.49	6372.32	8739.02	10958.2

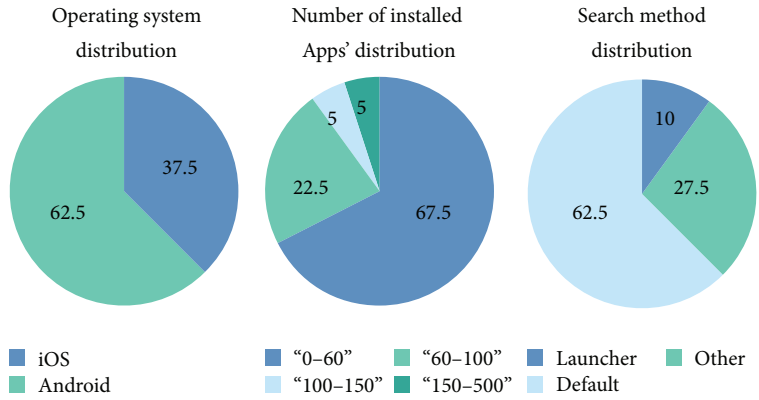


Figure 16: Participant's characteristics.

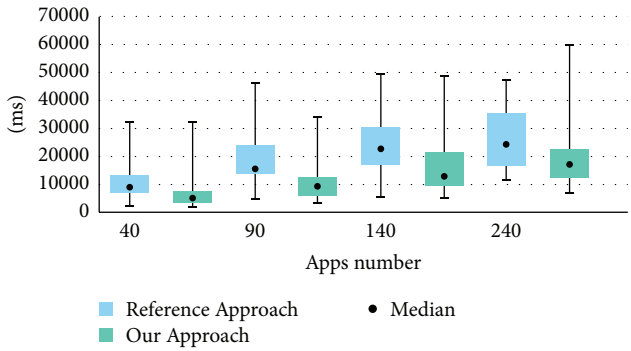


Figure 17: Differences between approaches in terms of median of time (ms) until the user finds the right App.

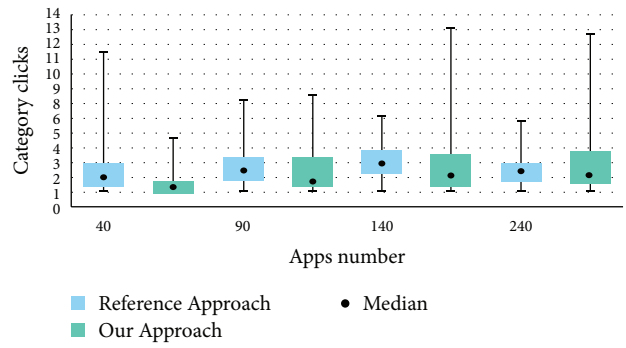


Figure 18: Differences between approaches in terms of median of category clicks number until the user finds the right App.

When working with Our Approach, participants clicked less on the categories tree than when working with Reference Approach, from 40 Apps (to a larger extent) to 240 Apps (to a lesser extent). Figure 18 presents the main differences between Reference Approach and Our Approach in terms of the number of clicks on the category tree until the right App is found.

In every stage, as compared with the Reference Approach, the number of category clicks with Our Approach increased, as the hierarchy grew (as Our Approach uses the hierarchical categories approach while Reference Approach has fixed categories). Still, the median number of category clicks is less for Our Approach than for Reference Approach. In each approach, at all stages there is at least one user who found the right App using only one category click (lower whisker). In addition, we can see that the green boxes (quartile 2 + quartile 3) for 140 and 240 Apps are comparatively long for Reference Approach, which indicates that the distribution of users' category clicks is wider. Further, the fact that the max. number of clicks is higher than that for Reference Approach may be attributed to learning difficulties of some of the users. To test whether these differences are statistically significant, Glimmix analysis was conducted. The results indicate that the differences between Reference Approach and Our Approach in the 40 Apps and 240 Apps stages are significant ($P < 0.05$); for example, the mean number of clicks on the tree was significantly lower with 40 Apps. On the other hand, in the last stage with 240 Apps, when working with Our Approach, participants clicked more times on the tree than when working with Reference Approach ($P < 0.05$). No significant differences were found between the two approaches with 90 or 140 Apps.

4.6. System Usability Scale and Preferences Questionnaires. Regarding the hypothesis that categorization of Apps based

Table 9: Comparison of approaches according to the category clicks and stages by various measures.

Measure/stage	Reference Approach				Our Approach			
	40	90	140	240	40	90	140	240
Mean	2.65	2.82	3.09	2.49	1.66	2.36	3.11	3.46
Median	2.1	2.6	3	2.5	1.4	1.8	2.2	2.2
St. dev.	1.92	1.53	1.33	1.04	1.01	1.6	2.69	2.93

on their functionality will be preferred by the users to current popular approach, as part of the study, the participants filled the SUS questionnaire [48]. The results of the questionnaire indicated that Our Approach ($M = 69.98$, SD = 21.55) scored significantly higher (paired t-test, $P < 0.001$) than Reference Approach ($M = 44.7$, SD = 19.26). Our Approach received a median SUS score of 75 and Reference Approach received 42.5. According to Bangor et al. [49], the interpretation of these scores is that Reference Approach's score is almost poor and Our Approach's is between good and excellent. Moreover, at the end of the experiment, the users were asked to express their opinions and comment on the approaches and compare them with other methods they know. It is noteworthy that, given that the two approaches have the same design, poor design may have impacted both approaches equally, and hence, the large difference really matters.

Figure 19 presents the partial preference questionnaire distributions. Among the sample, 72.5% preferred Our Approach to Reference Approach. When the participants were asked which search method they found most suitable, 42.5% preferred manual folder, 37.5% preferred Our Approach, 15% preferred Reference Approach, and 5% preferred the default search method of the operating system. 62.5% wanted to adopt Our Approach as compared with methods such as Reference Approach, manual folders, home screen icon arrangement, and default and other search methods (open-ended question). The participants mentioned in their comments that they prefer manual folders (pie chart (b)) for reasons of accuracy, learnability, and recall; however, they wanted to adopt our solution (pie chart (c)) because of the automatic mechanism and the time saved using it as compared to the manual folders (part of other methods) and Reference Approach. This is a very interesting and unexpected finding that will be discussed further.

4.7. Open-Ended Questions Analyses. We also analyzed the participants' comments regarding their preference (open-ended questions, e.g., what are the pros and cons of each approach?) in order to elicit more insight into the pros and cons of each approach. The participants mentioned that the main drawbacks of Our Approach are the multiple subcategories and the need to adapt quickly to the new Apps categories configuration. In addition, they noted the advantages of Our Approach, such as significant time saving while searching Apps and the notion that the category tree is specific, easy to navigate, and organized better than that of the other search methods they know. Regarding Reference Approach, the participants mentioned that it is less convenient, mainly because of the general folders that are inconsistent and confusing.

Figure 20 presents a word cloud of the participants' comments. The font size of each item is proportional to the number of times the word appeared in the comments. As can be seen, frequently appearing words can be divided into two categories: "mechanism" and "sentiments." The words used to describe Our Approach that appeared in the "mechanism" category were categorization, hierarchy, search, dynamic, section, and more. The "sentiment" words were easy (and easily), logical, clear, organized, better, and more. For Reference Approach, the frequent words in the "mechanism" category were folder, search, general, order, scroll, and more. The "sentiment" words were cumbersome, hard, inconsistencies, problem, confused, and more (clear and convenient appeared mainly with less/not preceding them). Generally speaking, given the sentiments expressed in the text, Our Approach received more approval and generated more positive feeling than Reference Approach.

5. Discussion

5.1. Overall Approach. In this paper, we suggested a novel approach for addressing the problem of finding installed Apps. We suggested a hierarchical functionality-based Apps ordering approach to help users organize their Apps and ease the process of finding infrequently used installed Apps. We addressed the problem in a three-stage process, illustrated in Figure 21, that includes Web mining for enriching Apps descriptions, Apps functionality elicitation and representation, functionality-based classification, and finally a taxonomy-based presentation of Apps clusters. We conducted an empirical study that tested the feasibility and effectiveness of the new approach that included a quantitative analysis of the individual steps, as well as a user study that evaluated the integrated solution. We found that, at most stages, the results were satisfactory as a proof of concept, but more work may be needed to improve the stages and the overall solution, as in some cases the existing tools and techniques we used did not provide high quality results and we had to manually improve the results in order to begin the next stage from a better start point.

In the user study, we focused specifically on evaluating how the idea is perceived by users according to the classification results (included Apps misclassifications that may occur with automatic tools). The user study demonstrated that Our Approach outperformed the Reference Approach with respect to the time it took users to find Apps (Hypothesis (A.1) confirmed) and the number of category clicks (Hypothesis (A.2) confirmed). The participants preferred Our Approach to the Reference Approach (Hypothesis (B.1) confirmed). They mentioned advantages such as intuitive

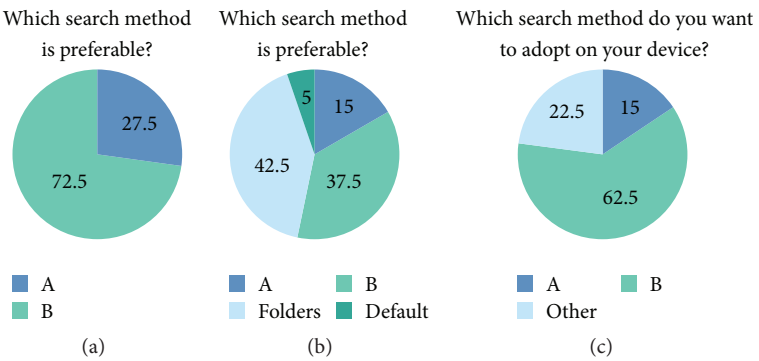


Figure 19: Preference questionnaire distributions (A: Reference Approach; B: Our Approach).

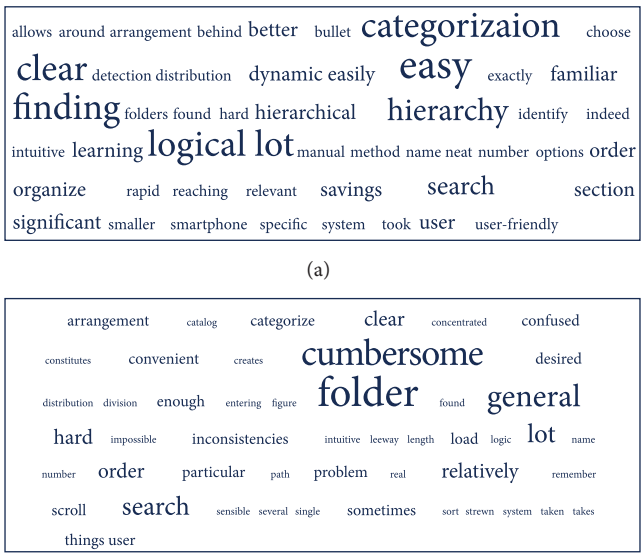


Figure 20: Word cloud of open-ended questions. The larger the font size is and the greater the contrast is, the more frequently the participants used the word in their comments about Our Approach (cloud (a)) and the Reference Approach (cloud (b)).

Figure 21: Functionality-based-taxonomy Apps organization process steps.

Apps organization (although the Apps categorization was not perfect), effective time saving while searching Apps, and efficient category tree navigation. The users wanted to adopt Our Approach more than other App search methods (Hypothesis (B.2) confirmed). It is noteworthy that the users preferred our suggested approaches, although their structure was more complex than that of the Reference Approach (hierarchy of App clusters versus list of App clusters) because of the categorization granularity with levels of abstraction. However, although the users preferred Our Approach to the Reference Approach, they preferred manual folders to all other approaches. This is an interesting finding that may be attributed to the fact that most users nowadays are used to organizing items in folders and that finding an installed App is not yet a major problem. The suggested approach will prove beneficial with the increase in the number of installed Apps, where maintaining manual folders will become a challenge.

As a result, we can conclude that a more detailed layered taxonomy is more beneficial than a flat and approximate list; however, the automation of Apps labeling and taxonomy building needs to be improved in order to achieve a better performance, in particular when we aim at automating the process completely, as the number of installed Apps will increase. It seems that there is room for further research on all the individual stages. Still, it seems that this approach provides an effective solution to the problem. Moreover, when new Apps are downloaded and installed, they can automatically be placed in the correct location according to the categorization. Apps may also be rearranged periodically

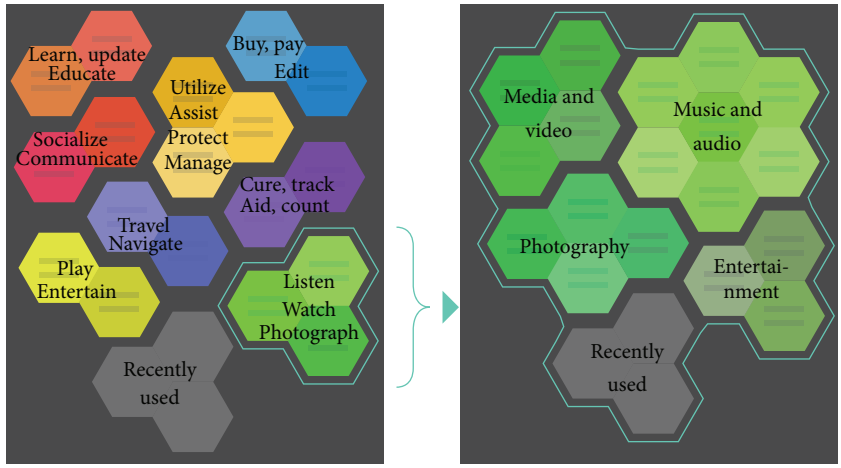


Figure 22: Suggested mobile UI solution.

as the number of Apps grows and some balancing is needed. In addition, this method allows the classic search of Apps to be enriched by using their name together with a search by functionality. For example, suppose Yelp has a label "travel"; it is very likely that a user can find this app simply by typing "T" in an app search box and obtaining a list of functional categories starting with "T," including travel, and then can look at the list of Apps under travel, rather than searching the tree from its root.

5.2. Suggestion for a Mobile Graphic User Interface. The category tree on the prototype interface renders the task quite remote from what would actually be happening in terms of interaction on a mobile device. Therefore and following the participants' comments about the simplistic interface of the prototype, we illustrate a possible visual solution for Apps ordering on a mobile device. Our challenge consists of presenting the App groups to mobile users in a limited screen area, where quick App identification is needed. We illustrate the desired user interface (Figure 22):

(1) Scalable hierarchical level of details: the desired state of the user interface is a hierarchical reflection with several levels (three as an example) containing between 100 and 150 Apps. The number of Apps directly affects the number of levels in the hierarchy. The interface is able to include about 20 icons at every level, and hence, with two levels we may handle up to 400 Apps and with three levels up to 8000 Apps, if all categories are fully populated and evenly distributed.

(2) Personalized: the interface presents the categorization that is specific to the user. The categories are created according to the Apps installed on the user's device.

(3) Ease of use: the interface allows quick and easy access to the Apps. A child function would be visible when visiting a parent function in order to enhance Apps searching. By tapping a particular function, rather than simply transferring to another window, the next level is presented over the whole screen, while the other functions may be shrunk and moved down to the edges of the screen (not shown in the figure). The fluid behavior of the screen allows simple switching between functions within the screen.

(4) Overlapping: there are Apps having characteristics that match several categories and are therefore assigned to all of them, allowing more than a single way to find these Apps.

5.3. Research Limitations and Implications. While suggesting a novel Apps ordering approach and demonstrating its feasibility, this work has a number of limitations regarding the suggested process for the *functionality-based Apps organization* and the *user study*. The implications of these limitations must be handled so that the suggested process will be a success in practice:

(1) Process limitations: the approach developed is far from being complete. The individual parts need to be refined further. It was tested and evaluated on an Apps database containing 6633 Apps extracted from 2 specific categories (Tools and Productivity). There are App stores with over a million Apps. These Apps are divided into approximately thirty general store categories. A larger scale research study may be required for examining the ability of the system to handle a real-life huge amount of Apps across a wide variety of functionalities and topics.

(2) The study is limited to two major categories of Apps: "Productivity" and "Tools." While it is true that these two categories can be the major source of confusion in App search because of their functional ambiguity, it might be interesting to explore how the proposed method performs for other categories.

(3) There appear to be other methods of App functionality categorization using methods such as LDA [50], and the performance of the proposed technique is not compared against such methods.

(4) User study limitations: the user studies took place in a university laboratory and the users tested Our

Approach using a PC with a WinForm interface, although we made an effort to simulate everyday situations. A field research study using our suggested approach should be performed to examine it in a real-life situation.

6. Conclusions and Future Work

In this paper, a solution to the need for an installed App finding method was proposed. We suggested and evaluated a method that automatically arranges Apps according to their functionality in order to help users organize their Apps. The approach was found to be technically feasible, although its individual stages need further improvement. The approach was found to be useful by the users and was perceived to be better than current simple approaches. Moreover, it seems that, with the growing numbers of Apps, the need for such effective Apps ordering solutions will increase and more sophisticated solutions, like the one suggested in this paper, are needed. However, as noted, during the study, we encountered quite a few challenges that need to be further investigated and these include the following:

(1) Generalization: we intend to evaluate this solution for larger numbers of diverse Apps in the virtual store, from different categories (instead of thousands of Apps, millions of Apps, from more categories). It appears that a general solution must rely on an automatic classification process. In future work, we intend to examine the use of predefined taxonomies, such as the Yahoo Open Directory project.

(2) Comparison: comparison with other approaches for functionality-based categorization.

(3) The proposed approach: the proposed approach combines flat clustering with semantic-based abstraction for creating hierarchical clustering. It may be interesting to experiment with using hierarchical clustering from the beginning and evaluate the contribution of the semantics.

(4) Field study using mobile App: in order to evaluate the idea in a real-life environment, a mobile App should be developed that demonstrates the idea and allows experimentation in a real-life setting and with the users' personal devices.

(5) Attractive and interactive user interface: in accordance with the *guiding principles of user interfaces* described in this paper, we will create a more intuitive user interface.

In addition to the future research suggested based on our work, there are additional related research directions that may further enhance the use of Apps as the number of Apps on personal devices grows:

(1) Personalized recommendation and prefetch based on context awareness: currently, clusters of installed Apps are presented as a taxonomy that suits the user. We can expand the personalization of the technology such that the clusters will appear in context with the user's behavior. For example, a location-based service will take a user's physical location next to a shopping center into account and the shopping cluster of Apps will pop up as a recommendation.

(2) Developers' decision-supporting system: when a developer uploads his/her created App and its description to the virtual store and cannot decide which category to assign it to, applying Our Approach, a system can suggest the correct classification of an App.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors would like to thank Dr. Joel Lanir and Dr. Nitsa Barkan of the University of Haifa for helping in the design of the user study and in the analysis of the results.

References

[1] Wikipedia, "Mobile app," 2012, http://en.wikipedia.org/wiki/Mobile_app.

[2] S. Costello, *How Many Apps Are in the iPhone App Store*, About.com Guide, 2013, http://ipod.about.com/od/iphonesoftwareterms/qt/apps-in-app-store.htm.

[3] D. Horace, "App developers receive $12 for each iOS device sold," Asymco, 2012, http://www.asymco.com/2012/02/19/app-developers-get-12-for-each-ios-device-sold/.

[4] S. Paul, Android users have an average of 95 apps installed on their phones, according to Yahoo Aviate data, TheNextWeb, TWN blog, 2014.

[5] Charles Newark-French, Mobile App Usage Further Dominates Web. Spurred by Facebook, 2012.

[6] Appsfire 2012, http://blog.appsfire.com/infographic-ios-apps-vs-web-apps/.

[7] S. Perez, *App-ocalypse*, TechCrunch, 2011, http://techcrunch.com/2011/12/18/app-ocalypse/.

[8] A. Karatzoglou, L. Baltrunas, K. Church, and M. Böhmer, "Climbing the app wall: enabling mobile app discovery through context-aware recommendations," in *Proceedings of the 21st ACM International Conference On Information And Knowledge Management (CIKM '12)*, pp. 2527–2530, ACM, November 2012.

[9] T. Yan, D. Chu, D. Ganesan, A. Kansal, and J. Liu, "Fast app launching for mobile devices using predictive user context," in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys '12)*, pp. 113–126, ACM, Lake District, UK, June 2012.

[10] A. Pash, "Organize your Apps by action instead of category for a more intuitive find-and-launch system," Lifehacker, 2012, http://lifehacker.com/5882749/organize-your-apps-by-action-instead-of-category-for-a-more-intuitive-find+and+launch-system.

[11] D. L. B. Lulu and T. Kuflik, "Functionality-based clustering using short textual description: Helping users to find apps installed on their mobile device," in *Proceedings of the International Conference on Intelligent User Interfaces (IUI '13)*, pp. 297–305, ACM, New York, NY, USA, March 2013.

[12] M. Böhmer and A. Krüger, "A study on icon arrangement by smartphone users," in *Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*, pp. 2137–2146, ACM, Paris, France, April-May 2013.

[13] J. G. Shanahan and E. J. Glover, "Bridging the app gap: from web search to app search via functional search," in *Proceedings of the 1st Workshop on Appification of the Web*, Lyon, France, April 2012.

[14] M. Rudolf, "In pursuit of a perfect App search engine," Workshop on the Appification of the Web, 2012.

[15] E. Gabrilovich, "From information needs to action needs: towards contextual app search and recommendation," in *Proceedings of the Workshop on the Appification of the Web (AppWeb '12)*, Lyon, France, April 2012.

[16] D. Barreau and B. A. Nardi, "Finding and reminding: file organization from the desktop," *ACM SIGCHI Bulletin*, vol. 27, no. 3, pp. 39–43, 1995.

[17] O. Bergman, S. Whittaker, M. Sanderson, R. Nachmias, and A. Ramamoorthy, "How do we find personal files? The effect of OS, presentation & depth on file navigation," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI '12)*, Austin, Tex, USA, May 2012.

[18] S. Voida, E. D. Mynatt, and W. K. Edwards, "Re-framing the desktop interface around the activities of knowledge work," in *Proceedings of the 21st ACM Symposium on User Interface Software and Technology (UIST '08)*, pp. 211–220, ACM, October 2008.

[19] H. Bruce, A. Wenning, E. Jones, J. Vinson, and W. Jones, "Seeking an ideal solution to the management of personal information collections," in *Proceedings of the Information Seeking in Context Conference (ISIC '10)*, Murcia, Spain, 2010.

[20] Stardock, Fences, 2012, http://www.stardock.com/products/fences/.

[21] J. Karlin, "Launchy," 2010, http://launchy.net/.

[22] Punk Labs, RocketDock, 2008, http://rocketdock.com/.

[23] Cogeco, RK Launcher, 2005, http://www.askvg.com/rk-launcher-one-of-the-best-dock-utility-for-windows/.

[24] Ecocardio, Orbit, October 2012, http://www.ghacks.net/2008/06/02/orbit-provides-a-circle-based-menu-for-windows/.

[25] E. X. Launcher, GO Launcher Dev Team, 2013, https://play.google.com/store/apps/details?id=com.gau.go.launcherex.

[26] ZeroTouchSystems, "Auto App Organizer," 2012, https://play.google.com/store/apps/details?id=com.utility.autoapporganizer&feature=related_apps#?t=W251bGwsMSwxLDEwOSwiY29tLnV0aWxpdHkuYXV0b2FwcG9yZ2FuaXplciJd.

[27] GinLemon, Smart launcher, 2011, https://play.google.com/store/apps/details?id=ginlemon.flowerfree.

[28] EverythingMe Launcher, March 2014, http://www.androidauthority.com/everythingme-launcher-what-you-need-to-know-342218/.

[29] Aviate Beta, Yahoo, March 2014, https://play.google.com/store/apps/details?id=com.tul.aviate&hl=en_GB.

[30] Androidrank, October 2012, http://www.androidrank.org.

[31] SharpNLP—open source natural language processing tools 2006, version 1.0.2529 Beta, April 2013, http://sharpnlp.codeplex.com/.

[32] K. K. Schuler, *VerbNet: a broad-coverage, comprehensive verb lexicon [Ph.D. thesis]*, University of Pennsylvania, Philadelphia, Pa, USA, 2005.

[33] Bing Synonyms API, October 2012, http://datamarket.azure.com/dataset/bing/synonyms.

[34] S. Banerjee and T. Pedersen, "Extended gloss overlaps as a measure of semantic relatedness," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, vol. 3, pp. 805–810, APA, Acapulco, Mexico, August 2003.

[35] G. A. Miller, "WordNet: a lexical database for English," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[36] T. Dao and T. Simpson, "Measuring similarity between sentences," April 2013, http://wordnetdotnet.googlecode.com/svn/trunk/Projects/Thanh/Paper/.

[37] C. D. Manning, R. Prabhakar, and H. Schütze, *Introduction to Information Retrieval*, vol. 1, Cambridge University Press, Cambridge, UK, 2008.

[38] E. Gabrilovich and S. Markovitch, "Computing semantic relatedness using wikipedia-based explicit semantic analysis," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI '07)*, pp. 1606–1611, Hyderabad, India, January 2007.

[39] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing & Management*, vol. 24, no. 5, pp. 513–523, 1988.

[40] H.-F. Yu, C.-H. Ho, Y.-C. Juan, and C.-J. Lin, "LibShortText: a library for short-text classification and analysis," Tech. Rep., 2013, http://www.csie.ntu.edu.tw/~cjlin/papers/libshorttext.pdf.

[41] C.-J. Lin, R. C. Weng, and S. S. Keerthi, "Trust region Newton method for logistic regression," *Journal of Machine Learning Research*, vol. 9, pp. 627–650, 2008.

[42] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of Machine Learning Research*, vol. 2, pp. 265–292, 2002.

[43] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "Training algorithm for optimal margin classifiers," in *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory (COLT '92)*, pp. 144–152, July 1992.

[44] Z. Kozareva and E. Hovy, "A semi-supervised method to learn and construct taxonomies using the web," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '10)*, pp. 1110–1118, Cambridge, Mass, USA, October 2010.

[45] W. Wu, H. Li, H. Wang, and K. Q. Zhu, "Probase: a probabilistic taxonomy for text understanding," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 481–492, ACM, Scottsdale, Ariz, USA, May 2012.

[46] Ozgur Ozcitak, ImageListView, 2010, http://www.codeproject.com/Articles/43265/ImageListView.

[47] SAS, Statistical software, version 9.4, SAS Institute Inc., Cary, NC, USA.

[48] J. Brooke, "SUS: a 'quick and dirty' usability scale," in *Usability Evaluation in Industry*, P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, Eds., pp. 189–194, Taylor and Francis, London, UK, 1996.

[49] A. Bangor, P. Kortum, and J. Miller, "Determining what individual SUS scores mean: adding an adjective rating scale," *Journal of Usability Studies*, vol. 4, pp. 114–123, 2009.

[50] A. Gorla, I. Tavecchia, F. Gross, and A. Zeller, "Checking app behavior against app descriptions," in *Proceedings of the 36th International Conference on Software Engineering (ICSE '14)*, pp. 1025–1035, ACM, Hyderabad, India, May 2014.