Gregor David Sitter, BSc

Recommendations Based on Tags and Extensions for Catrobat's Community Site

Master's Thesis

to achieve the university degree of

Master of Science

Master's degree programme: Software Development and Business Management

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl-Ing. Dr.techn. Wolfgang Slany

Institute for Softwaretechnology
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Wolfgang Slany

Keutschach, June 2018

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Date

Signature

Abstract

Due to the strong increase of mobile devices such as smartphones and tablets, as well as the spread of social software and the use of social networks like Facebook, Google+ and Twitter, the data sent via the World Wide Web (WWW) are also increasing. Moreover, the data is not only sent but also stored, which results in a massive data overload. Thus, users are often overwhelmed when they are trying to find relevant information or other materials that they may be interested in. To tackle this problem various mechanisms are used for sorting, searching and managing this data overload. Recommender Systems (RSs) and tagging systems are two of these mechanisms, which are also in the focus of this Masters's thesis. RS is a well-studied area which offers many approaches to rely on. Therefore, only the popular systems and approaches such as collaborative, content-based and knowledge-based, are discussed in this thesis. Further, tagging systems have also a good potential to filter and search for relevant content. Tags have their origin in simple annotations, which were used for extending data in the web with additional information. In social media, tags are used to mark pictures, books, music or other digital content for sorting and sharing it with other users. To understand the concepts of annotations and tags and the resulting system a short overview over Web 2.0 and social networks is given in this thesis.

Another focus in this thesis is the implementation of a tagging systems and a RS which is based on the implemented tagging system for the community site of Catrobat. Data gained from the implementation of these systems, are evaluated and analyzed to verify if the activity on the community site and the downloads of programs can be increased thereby. Afterwards, the results are discussed followed by a conclusion. Finally, this thesis ends with a summary of relevant findings and possible features with which the implemented system can be extended and improved.

Keywords
Catrobat, Pocketcode, Recommendations, Recommendation Systems, Tags, Tagging Systems, Collaborative, Content-based, Knowledge-based

Contents

List of Figures

List of Tables

Listings

1 Introduction

With the constant advancement of online devices, such as the evolution from single-purpose to multi-purpose devices like smartphones and tablets, the mobile ecosystem is steadily growing (Böhmer et al., 2013). One key purpose of these online devices is to produce and transfer information – resulting in a huge overload of information. To overcome this problem, the concept of information filtering was introduced (Morita et al., 1994). According to Belkin et al. (1992), information filtering describes a set of processes like bringing information only to users who have a need for this particular information. There are several approaches to instigate and achieve information filtering. For example, if an individual is exploring a new city and is trying to find a fancy place for lunch, it might happen that there are too many options and that the individual might not be able to decide on which place to choose. One possible solution could be that this individual either decides to ask a local friend to recommend a place, or the person decides to go with a recommendation out of a traveler guide. Based on this scenario, two approaches can be identified. The latter example (conducting the travel guide) is called the "search method", where users can define the base for getting information, which is also the most common method. On the other hand, consulting a local friend would be called the "expert recommendation". However, in this case the information from the expert is only suited for the user, if both parties have related and similar preferences (Davidsson, 2010).

The two approaches stated above can be problematic, because not all possible information is considered. That means a system is required which can extract and deliver relevant information to a user. This extraction system is then called a *Recommender System (RS)*. Such systems can filter unseen information (so called items[1]) and try to predict whether the item is liked by

[1]Items can be digital (eBooks, online magazines) or physical (books, dvd's).

a user[2] or not. There are many types of RSs, such as collaborative, content-based or hybrid systems. The collaborative type tries to recommend items that are liked from users which are similar to the target user, while Content-based systems recommend similar items which are liked by the target user and Hybrids are combinations from two or more different RS (Ghazanfar et al., 2010).

Emerging from Web 2.0, social tagging systems are providing additional information which can be used in RS. More precisely, tags are used to describe items with keywords such as “Sport” or “Game”. With this information, existing traditional algorithms can be enhanced, or new RSs can be built from scratch (Costa-Montenegro et al., 2012).

1.1 Motivation

The motivation of this thesis is to capture and analyze the concept of constantly improving website usability by implementing RSs based on tags. Therefore, the community website of Catrobat is used. Catrobat is a FOSS project and also the name of a visual programming language at the Technical University of Graz (TUGraz). On the community website, programs which are created with Catrobat, can be downloaded, uploaded and shared with other users.

The main point of this thesis is to investigate if the implemented tagging and recommendation system benefits user activity and raises download numbers of programs. Therefore, collected data will be analyzed in order to prove or to argue the suitability of those systems. Two research questions were formulated:

Question 1

Can the activity on the community site be raised by adding a tagging system which helps users to sort and find programs?

[2]The user who gets the recommendation is called the target user in this thesis.

Question 2

Can the download numbers on the community site be increased by implementing a RS based on tags?

1.2 Thesis overview

This thesis is structured as follows. Chapter 2 contains a more detailed explanation about the project Catrobat, including background information and an overview of the project structure. Chapter 3 contains an overview of the topic RS and explains several underlying approaches. In Chapter 4, a short summary of Web 2.0, the Social Web and an explanation of tagging systems will be given. The practical part of this thesis can be found in Chapter 5, including a short overview of the existing implementations and the requirements of the tagging system and the RS. Chapter 6 analyzes and discusses the results which are provided by the implemented systems from Chapter 5. Finally, a summary of the most essential findings and ideas for future work in this field is provided in the last Chapter 7.

2 Catrobat

Catrobat[1] (Figure 2.1a illustrates the logo) is the name of a FOSS project founded by Wolfgang Slany, head of Institute of Software Technology (IST) and professor at the TUGraz.

Furthermore, Catrobat (former called Catroid) is also the name of the visual programming language which is being developed during the course of the FOSS project. The basis for the visual language is Scratch[2], a programming system developed by the Lifelong Kindergarten Group at the MIT Media Lab (Slany, 2014). Still, there is a difference between Scratch and Catrobat: Catrobat is running on smartphones and tablets, so no PC is necessary to program as this can be easily done via apps. Scratch on the other hand depends on hardware such as keyboard, mouse and a large screen whereas Catrobat targets mobile devices with multi-touch sensitive screens. These differences are leading to opposing challenges in user interaction and usability (Slany, 2012).

(a) Catrobat Logo (b) Pocketcode Logo

Figure 2.1: Logos of Catrobat and Pocketcode

[1] Catrobat, 2018b.
[2] Scratch, 2018b.

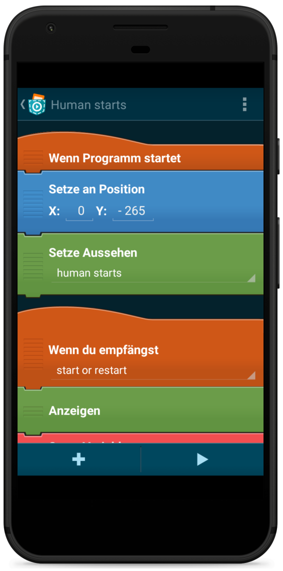


Figure 2.2: Pocket Code - Program

2.1 Functionality

Catrobat uses Pocket Code (PCo) (see Section 2.3) as an Integrated Development Environment (IDE). This IDE enables writing, editing and interpreting in one single tool.

Figure 2.2 shows the use of the visual programming language in the PCo app. Users only need to bring the pre-built blocks (bricks) together to create a program. Ultimately, individuals do not need to gain coding skills, because with the help of the bricks and the very easy and user-friendly handling, no programming knowledge is required to develop programs. With this set-up it is also simple to make games, stories and more. Another advantage is, that the user learns more about loops, variables and conditions, which are the basics of programming and developing software.

2.2 Overview of the Catrobat FOSS project

Catrobat as a project has many external contributers. However, the main developers are students[3]. Students can participate in the project when writing their thesis for the university degree Bachelor of Science in Engineering (BSc) or Master of Science (MSc). The FOSS project has the following structure:

1. *Project head*. Leader of the Catrobat project and the community. Wolfgang Slany, founder of Catrobat, is also the head of the project.
2. *Project management*. Organizational and administrative work such as handling account permissions or managing student participations in the project.
3. *Coordinators*. Leading the sub-teams and coordination between the teams.
4. *Developers*. They implement features and continue to develop the project.

Table 2.1 shows an overview of sub-teams and sub-projects of Catrobat. Students can choose to join one of these groups.

Name	Description
Catroweb	Responsible for homepage of pocketcode
Chromecast	Responsible for Google Chromecast support in Pocket Code
Core	Core team of Pocket Code
Design	Responsible for the design
Drone	Responsible for controlling the AR Drone via Pocket Code
HTML5	Responsible for HTML5 implementation
iOS	Responsible for implementation of Pocket Code for smart phones with iOS
Jenkins	Responsible for the test-server environment
Lego Robot	Responsible for controlling the Lego Mindstorm robots via Pocket Code
Musicdroid	Responsible for playing music on your mobile phone
NFC	Responsible for Near Field Communication (NFC) support Pocket Code
NPO Marketing	Responsible for the marketing
Paintroid	Responsible for drawing pictures on your mobile phone
Phiro	Responsible for controlling the Phiro robot via Pocket Code
Physics Engine	Responsible for physic engine support in Pocket Code
RasperIno	Responsible for controlling the RasperyPi and Arduino with Pocket Code

Table 2.1: Catrobat Team Overview. Adopted from (Schnedlitz, 2016)

[3]Catrobat, 2018a.

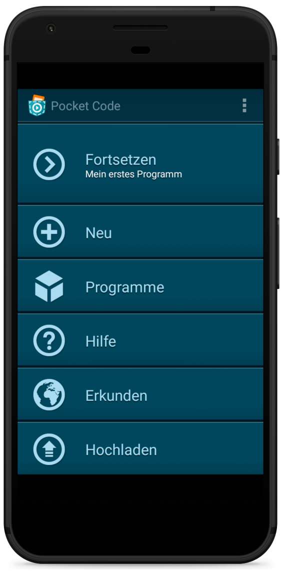


Figure 2.3: Pocket Code - Application

2.3 Pocket Code

PCo (Figure 2.1b illustrates the logo) is developed for more than one platform such as the most widely used Operating Systems (OSs) of Google[4] and Apple[5]. At the time of writing this thesis, the application is only available for android devices and can be downloaded[6] via Google Play Store[7].

Figure 2.3 shows PCo on an android device. The Graphical User Interface (GUI) is well structured and intuitive. The user can access programs fast via the GUI on the mobile device also share these and download new ones.

There is a community site for sharing and downloading shown in Figure 2.4. Not only the mobile app but also the community site has gone through many changes and several features were implemented. The upgrade of the web framework is one example of a fundamental change, which provided an

[4]Google, 2018a.
[5]Apple, 2018.
[6]Catrobat, 2018d.
[7]Google, 2018d.

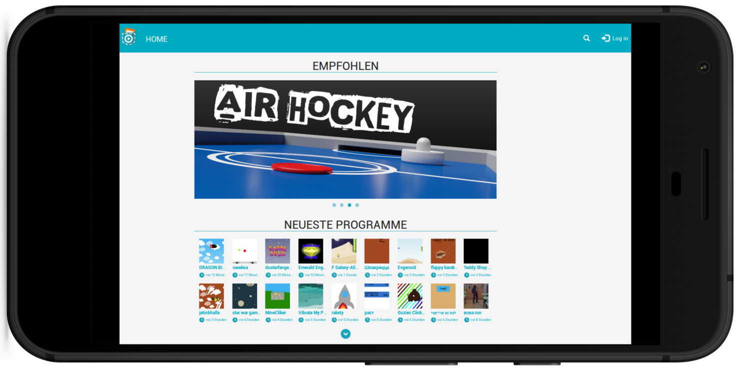


Figure 2.4: Homepage for sharing programs

easy way of further development. The web framework is called Symfony[8], the tool with which the comment section was integrated. The comment section is a space on the page, where users can comment on and discuss programs with others. In addition, social media channels, such as Facebook[9], Google+[10] or Twitter[11] are integrated in the community site.

[8]Symfony, 2018.
[9]Facebook, 2018.
[10]Google, 2018b.
[11]Twitter, 2018.

3 Recommendation System

This chapter gives an overview of RS and their approaches and techniques such as collaborative, content-based, knowledge-based. Further, the knowledge sources where RS acquire their data for recommendations are explained in more detail. How important the research of these systems are, shows the Netflix price at the end of this chapter.

3.1 Background

In this digital age people are using the internet for communication and services. Products, such as books and movies are used online by millions of people. The number of these so called items is growing every day. As a result, it is getting more difficult for the users to get through this large amount of available online data (Dareddy, 2017).

If people are looking for advice or recommendations, they usually tend to ask friends or experts. They talk about movies, gossip about latest fashion, ask a librarian for a good book or talk with a seller at the news stand. However, this form of recommendation has a weakness, because not all relevant information might be included. There is maybe a new movie or a new book released which might be of interest, but none of the involved parties can recall or access these “bits of information”. Therefore, in order to acquire proper information, a computer-based system can be used, which is called RS (Ekstrand et al., 2011). RSs are a part of information filtering systems. Information filtering provides individual or sets of data from a larger assortment, which might be of interest for the user (Van Meteren et al., 2000).

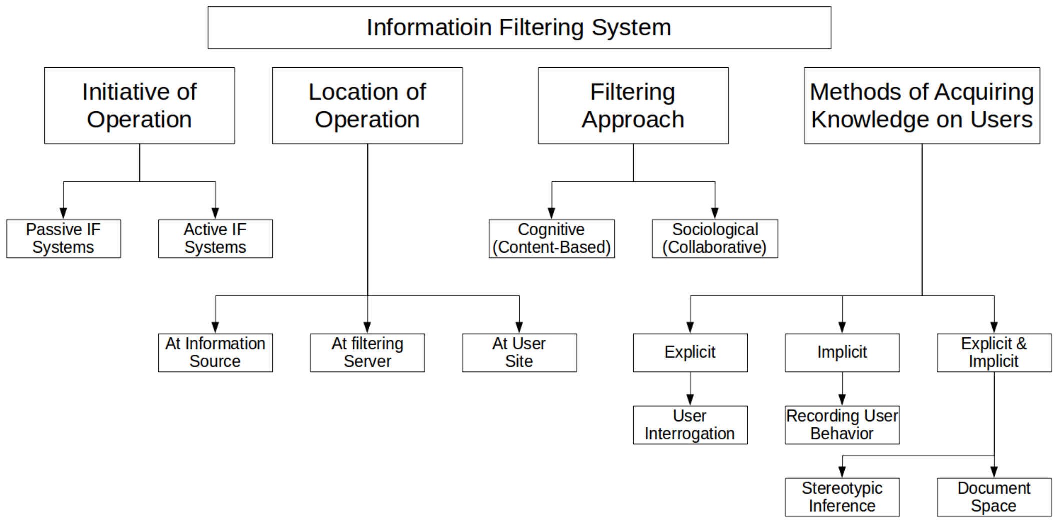


Figure 3.1: Classification of Information Filtering. Adopted from (Hanani et al., 2001)

Therefore, these systems can be used for answering questions such as "what should I buy next?" or "which movie should I see next?" (Felfernig, Jeran, et al., 2014).

As can be seen in Figure 3.1, Hanani et al. (2001) describes four parameters to classify information filtering systems.

- *Initiative of operation*: Information filtering systems can be active or passive and describe *who* initiates the filtering process. Active systems use the profiles of the users and search, collect and send the relevant information to the user. Passive systems look over an incoming data stream and omit irrelevant information.
- *Location of operation*: A system operates at the information source, at the server site or at the user site. When if operating at the information source, the user can send their profile to the information provider and the wanted information is send back to the user. On the other hand, if the system runs on a server, the user sends the profile to the server and the server processes the data and requests the information from the information provider. The server receives the data and sends it back to the user. The most popular location is on the user site. An

incoming stream is evaluated and filtered locally. This variant is a passive filtering system.

- *Filtering approach*: System can use either cognitive or sociological filtering. The cognitive approach uses the content of the data for the filtering process, therefore it is also called content-based filtering. The sociological approach or also called collaborative filtering, is a recommendation process, which is used to recommend items according to the habits of the user. Systems based on the collaborative approach are now being referred to recommendation systems which are explained in more detail in Section 3.2.1.
- *Methods for acquiring knowledge about users*: Methods for acquiring knowledge on users can be explicit, implicit or a combination of both. In the explicit approach, the users usually fill out a form, which describes their areas of interest whereas, the implicit approach creates the knowledge automatically by recording the user's behavior.

These days, the collaborative approach is the most commonly used filtering method in terms of RSs. Generally, this approach uses the similarity of users and their ratings to compute a recommendation for the target user (Gong et al., 2008). Filtering approaches and methods are explained in more detail in Section 3.2.

According to Resnick et al. (1997), the term Collaborative Filtering (CF) was coined by the developer of *Tapestry*, one of the first RSs.

Tapestry was an experimental mail system in which users were being included in the filtering process by using annotations. For example, if user A wants to acquire specific information annotated by user B with "Must Read", user A can filter all available data by this annotation (Goldberg et al., 1992).

Resnick et al. (1997) preferred the term *recommender system* (synonym for Recommendation System) instead of CF. The term recommender systems widely gained acceptance and the definition of this term was expanded gradually. Therefore, this term includes all recommendations of items that can be of any interest to the user, no matter how these recommendations are generated (Burke, Felfernig, et al., 2011).

3.2 Techniques

Traces of RSs can be found in various areas, such as information retrieval, forecasting theories and management science. However, the beginning of independent research on RSs starts not until the mid-1990s, when researchers started to concentrate on recommendation problems that occur by using rating structures (Adomavicius et al., 2005).

As a result, many techniques and approaches such as collaborative and content-based filtering have been developed over the years (Park et al., 2012). These filtering methods and concepts such as knowledge-based and hybrid methods, which are commonly known as the traditional techniques, will be explained in more detail. Other methods such as social network-based, context awareness-based and group recommendation approaches are more advanced and more recently developed (Lu et al., 2015).

3.2.1 Collaborative Recommendations

Pure collaborative recommendations are based on the rating information history of other users which have a similar rating profile as the target user to recommend unseen and not yet evaluated items. There is no need to know specific details about the items (for example genre or author of books) for recommendation since only the rating is important (Balabanović et al., 1997).

The collecting of ratings or user opinions can be *implicit* or *explicit*. *Implicit* ratings are based on the behavior of the user, such as browsing data or purchasing information. If a user purchases a product, this can be interpreted as a positive rating. There is no need for the user to do something personally. On the other hand, *explicit* ratings are actively submitted by the users themselves, for example by rating products on a scale from 1 (very bad) to 5 (very good) (Breese et al., 1998).

Recommendations can be provided to the users in two ways. The first one is *prediction*. The system predicts if the user likes the item or not and based on this prediction the recommendation is generated. The second one is *ranking*, where the user is provided with a ranked list of recommended items. Generally, the latter one is more widely used (Lam, 2005).

In order to generate these recommendations, the algorithm used for filtering in collaborative systems can be split into two methods: *memory-based (neighborhood-based)* and *model-based*. The *memory-based* uses data from databases, where all previous user-item ratings are stored. Then, the nearest neighbors are calculated with algorithms such as k-nearest-neighbors (kNN). On the other hand, the *model-based* method uses the data to develop a predictive model. In practice, many approaches in the *model-based* method such as Bayesian Clustering, Latent Semantic Analysis or Support Vector Machines exist (Desrosiers et al., 2011). Combining algorithms from both methods create a hybrid filtering approach. These hybrid algorithms usually have a better performance than a pure *memory-based* or *model-based* approach (Su et al., 2009).

User-based Collaborative Filtering

User-based CF uses the similarity of users to predict the interest of the target user on a specific item. Users are matched as similar, if the rating history of same items is very much alike. To limit the number of similar users, a threshold can be defined or only the top-N users are selected. In order to measure the similarity of users, the *Cosine similarity* and *Pearson correlation* are commonly used (Wang et al., 2006).

Item-based Collaborative Filtering

In contrast to the already mentioned user-based CF, item-based CF is not using the similarity of users but rather the similarity of items and their ratings. Therefore, *Cosine similarity* or *Pearson correlation* are used to measure whether items are similar or not. However, there is one problem with the measurement, namely the difference of the rating behavior of the users. In order to alleviate this problem, an *adjusted Cosine measure* is applied where the user's average rating is subtracted from the ratings. Similar to the user-based CF, not all similar items are taken into account for the prediction. Only the top-N items are used (Wang et al., 2006).

Problems

There are two main challenges in collaborative systems: the *data sparsity* and *cold-start* problem. The *data sparsity* problem occurs, when users only rate or buy specific items. Therefore, users are quickly measured similar. Additionally, with no additional information about these users, the RS has problems to recommend items which the target user might like. One solution might be a hybrid approach, where additional information about the users, such as age, gender or education, is taken into account. Furthermore, the assignment of default values in databases can be used. Items get predefined ratings in order to fill in the blanks (Jannach et al., 2010, pp. 22–25).

The *cold-start* problem also occurs because of *data sparsity* (Jannach et al., 2010, p. 25). It can be divided into three groups, such as the *new-user*, *new-item* and *new-community* problem. The *new-community* problem describes the start-up difficulties, when a new system goes online. After initialization of the system, there is no data for the RS. One solution is to encourage the users to rate items. Another is to wait until enough data is collected to show recommendations. The *new-item* problem occurs when new items are being added to the system. Usually, these items have no ratings. One solution is to have a group of users which are responsible for evaluating each new item in order to get some ratings. However, one of the greatest challenges is the *new-user* problem. In contrast to the *new-item* problem, new users joining the system have no ratings or have not bought anything yet. To tackle this problem, the use of hybrid systems is common (see Section 3.2.4) (Bobadilla et al., 2013).

3.2.2 Content-based Recommendations

This approach tries to recommend items that are similar to items which the user liked in the past. The main difference to the collaborative approach, especially the Item-based filtering (see Section 3.2.1) is, that no rating is needed for the similarity measurement, only the content of the items is used (Van Meteren et al., 2000).

The content of an item is also called *attribute*, *characteristic*, *feature* or *variable* and can be structured or unstructured. Table 3.1 shows a structured movie database. The values of the content are well-defined, for example the genre can be action, fantasy, romance or drama, or a mix of them. Unstructured data has no well-defined values and often comes in form of free-text like in newspapers. Besides the author, the text of the newspaper is used as content. To deal with this free-text, the usual procedure is to create structured data out of the unstructured data, for example with calculating the *tf*idf* weight. *tf* is the term frequency and *idf* is the inverse document frequency. A high calculated weight of a term for a specific document means that the term provides the most information about this document (Pazzani et al., 2007).

ID	Title	Producer	Genre	Type
1	Title A	Producer 1	Action, Fantasy	DVD, Blue-ray
2	Title B	Producer 2	Romance	Streaming Media
3	Title C	Producer 3	Drama	DVD

Table 3.1: Item Representation

Furthermore, a user profile is needed for generating the recommendations. As already explained in collaborative recommendations (Section 3.2.1), the preferences of the user can be given *explicit*, for example by selecting checkboxes to say if the items is liked, or *implicit*, like purchasing an item. With the preferences of the users, algorithms can learn a user model for recommendation. Such algorithms are for example *decision trees*, *rule inductions*, *nearest neighbor methods*, *relevance feedback* or *probabilistic methods* (Pazzani et al., 2007).

Problems

Contend-based recommendations have also known limitations. One of these limitations is the *shallow content analysis*. For example, the users get recommendations based on the keywords used in an article. However, the RS does not differentiate in their quality, e.g. if an article is well-written or not. Another example is the recommendation of jokes. Most jokes are short, which means that there is not much content to extract. Therefore, a solution is to use annotations like tags, such as “very funny” (see Chapter **??**). Another problem is *overspecialization*. After learning the preferences of a user, the system tends to recommend items that are too similar. One example would be the recommendation of news articles, where the users receive other articles which they might have already read. One solution for this problem is to define a threshold. With this threshold, the system filters not only the items which are too different, it also filters items that are too similar. Another method is to insert random items to get “randomness” (Jannach et al., 2010, pp. 75–76).

Similarly to collaborative systems, contend-based methods have also the *cold-start* problem (Section 3.2.1), more specifically the *new-user* problem. In order to acquire data about the preferences of the new users, the user can provide some keywords or can choose from a list of keywords, after entering the system (Jannach et al., 2010, pp. 76–77).

3.2.3 Knowledge-based Recommendations

RSs with collaborative or content-based approaches are well suited for items which are being bought very frequently, such as books, dvds or newspapers. On the other hand, items such as houses, cars or computers are difficult to recommend since these items are not being bought on a regular basis. As a consequence, there are not many ratings which would be needed for the collaborative approach. Furthermore, old recommendations such as a rating of a five year old computer is misplaced for content-based recommendations. To fill this gap, knowledge-based techniques were introduced. This approach uses domain knowledge and the user requirement for recommendations. Section 3.4 explains the knowledge sources which are used by RSs. Knowledge-based systems can be distinguished into *case-based* and *constraint-based* recommender (Felfernig, Friedrich, et al., 2011).

Case-based Recommendation

Case-based approaches are using *similarity measures* to recommend items. For example, a user enters particular requirements into the system. The system searches for items which are similar to the requirements and presents an item as a solution. If the user is unsatisfied with the solution, the requirements can be changed and the system starts a new search (Lorenzi et al., 2005).

Constraint-based Recommendation

Similar to the case-based approaches, the constraint-based recommender works with the particular requirements entered. The difference is, that constraint-based techniques additionally take explicitly defined constraints for the recommendations. Based on that, only items are recommended which are not violating the constraints. If there are no items with this requirement available, the user must redefine the requirements (Jannach et al., 2010, pp. 81–87).

Problems

The problem of knowledge-based recommender lies in the knowledge acquisition, namely to gather the knowledge about items and to convert it to usable data. It has no *cold-start* problem since this approach needs no rated items or similarity between users, only the similarity between items and user requirements (case-based) or items which do not match the constraints (constraint-based) (Felfernig, Friedrich, et al., 2011).

3.2.4 Hybrid Recommendations

Hybrid recommender consist of more than one RS. Combining multiple systems is used to mitigate their disadvantages and to increase performance (Sharma et al., 2013). In the following sections, a variety of hybrid systems are discussed.

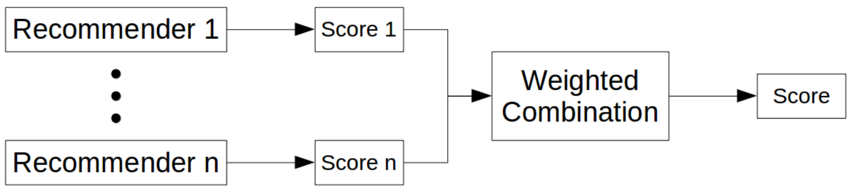


Figure 3.2: Weighted Hybrid System. Adopted from (Le et al., 2017)

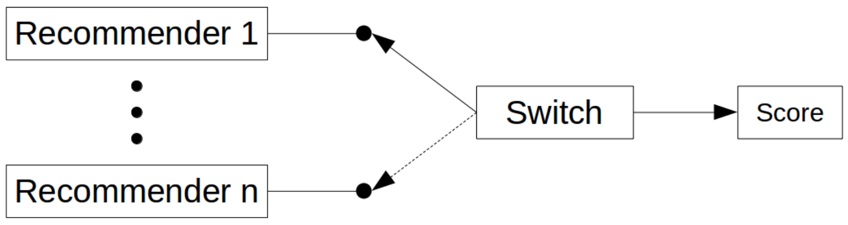


Figure 3.3: Switching Hybrid System. Adopted from (Le et al., 2017)

In weighted hybrid systems, the scores of items from all available recommender in the system are used to compute the final score of the item, which is illustrated in Figure 3.2. The simplest implementation would be a linear combination of these scores. The weight can be adjusted, because collaborative recommender are not good with items that are not rated often by users (Burke, 2002).

A switching hybrid can switch between the available recommender in the system, which can be seen in Figure 3.3. Based on the switching criteria, the system decides which recommender should be used to present its result. Therefore, it is possible to use different RSs for different user profiles with this hybrid (Burke, 2007).

In mixed hybrid systems, the recommendations of all available recommender

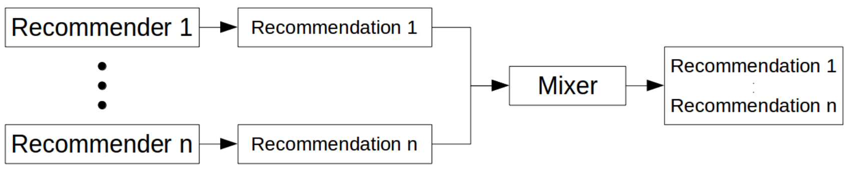


Figure 3.4: Mixed Hybrid System. Adopted from (Le et al., 2017)

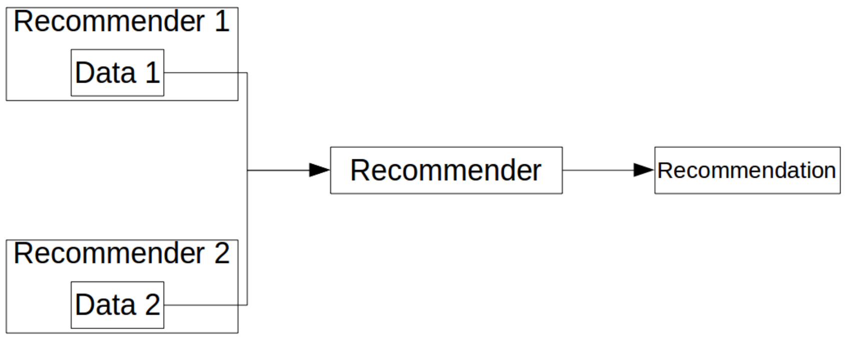


Figure 3.5: Features Combination Hybrid System. Adopted from (Le et al., 2017)

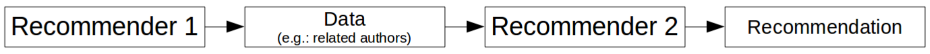


Figure 3.6: Features Augmentation Hybrid System. Adopted from (Le et al., 2017)

in the system are combined to one single recommendation and is presented to the user (Burke, 2002). This system is shown in Figure 3.4.

Hybrid system, which use a feature combination approach, consist only of one component with a recommendation algorithm. The inputs for the algorithm are the features of different data sources from other recommender (see Figure 3.5). Although there is only one component used for recommendation, feature combination is still a hybrid system. The reason is the underlying knowledge sources (Burke, 2007).

In feature augmentation, one recommender creates features which are used as an input for the next recommender (see Figure 3.6). The difference to feature combination is that feature augmentation is not using only raw data from different knowledge sources; it also uses additional functionality from the recommender before (Burke, 2002).

In cascade hybrid systems, the first recommender creates a rough ranking of the recommended items. The next recommender gets the ranking as an input (see Figure 3.7). This input can only be refined, not overturned (Burke, 2007).

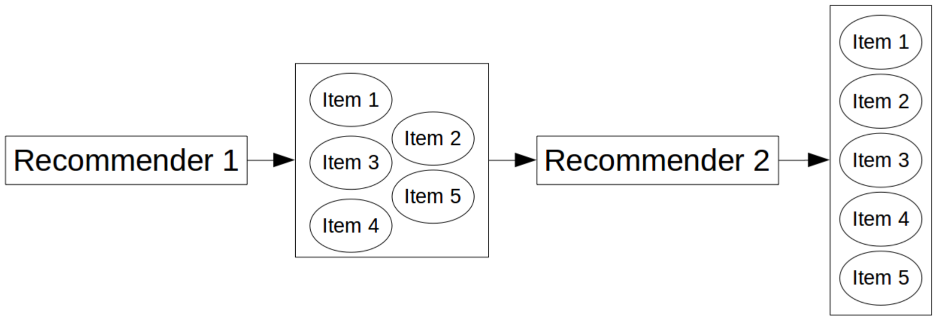


Figure 3.7: Cascade Hybrid System. Adopted from (Le et al., 2017)

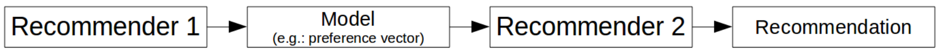


Figure 3.8: Meta-Level Hybrid System. Adopted from (Le et al., 2017)

In the meta-level approach, a model learned by the previous recommender is used as an input for the next recommender (see Figure 3.8). It must be ensured, that the produced model can be used by the next recommender logic (Burke, 2007).

3.2.5 Non-personalized Recommendations

Non-personalized recommender use the average rating or votes over a product from all users who have rated or voted for the item. Thus, the rating of the user is not necessarily the same as the average rating. Furthermore, this has the effect that all users are getting the same recommendation. Recommender which are built on non-personalized recommendations are *automatic* (no user effort) and *ephemeral* (no user recognition) (Schafer et al., 1999).

Another option is to recommend the top-N items with the most ratings regardless of the ratings itself. Here, the user gets only a ranked list of items which are popular (Cremonesi et al., 2010).

3.3 Purpose

According to Ricci et al. (2011), the function of RSs differs for the user and the service that provides the recommendations. They identified five functions or rather reasons, why the service providers are interested in this field.

- *Increase sales*: Selling more items to the users is the primary reason for most providers of RSs. In addition, providers which only offer online content, such as newspapers, journals or magazines, are not especially interested in increasing sales but in increasing the number of items read.
- *Increase sales of diverse items*: Similar to the first point, increasing sales of diverse items is also an important reason. The provider not only wants to sell the most popular item, but also items which are hard to find without using a RS.
- *Increase satisfaction*: An effective, in other words accurate RS can increase the user satisfaction, which concludes that the user enjoys the system and this results in a longer system usage.
- *Increase fidelity*: The more often the user visits the system or homepage, the more accurate the RS becomes. The loyalty to the system or the site increases, if the RS recognizes the user by each visit and gives the feeling of being important.
- *Increase user understanding*: Another important reason is the better understanding of the user. If the provider knows the preferences of the user, they can use this knowledge, for example to adjust the stock of items.

On the other side, the user has also reasons to use the recommendations. Herlocker et al. (2004) mentioned ten of them:

- *Context annotation*: A RS annotates existing context, such as a list of books, in order to show the user which books are worth reading.
- *Discover good items*: Users desire not only items, they want good items. Therefore, RS presents a ranked list. This list has a prediction of each item, which forecasts how much they would like them.

- *Discover all good items*: In a few cases the users not only want to see some good items, they wish to see all possible good items available. For example, in the medical sector it might be vital to see all possible items due to the fragmented and broad area.
- *Sequences of recommendations*: Some RSs benefit from presenting more than just one item. For example, after listening to one song, the RS displays a sequence of music tracks.
- *Just browsing*: There are users who are using the RS just for browsing. They have no intentions to buy something.
- *Find trustworthy recommenders*: Primary, the RS helps the user to find items, but some users have no trust in the recommendation and therefore they "play" with the system to see what recommendation they get.
- *Better profile*: Many users improve their profile by rating more items, in order to get more precise recommendations.
- *Express themselves*: The RS is not important for all users. They just want to rate items, for the rating's sake.
- *Help other users*: There are users who rate items for others, because they want to help the community.
- *Influence other users*: Some users use the RS not to improve their own recommendations, but rather to influence others. Therefore, they promote items that then other users get as recommendations.

3.4 Knowledge Sources

Every intelligent system needs knowledge to work properly. This is also the case in RSs. The algorithm used in recommender can be classified by this knowledge sources (Burke, Felfernig, et al., 2011). According to Felfernig and Burke (2008) there are four sources where RSs can gain knowledge. These four knowledge sources are listed below:

1. The target user itself (Individual)
2. Other users (Social)
3. Data of the recommended items (Content)
4. The domain of the recommendation (Content)

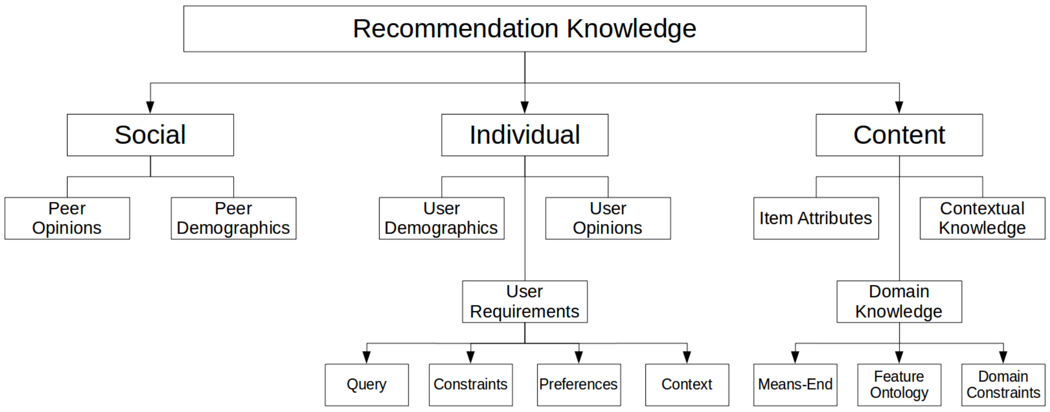


Figure 3.9: Knowledge Sources. Adopted from (Felfernig and Burke, 2008)

In Figure 3.9, the authors present a taxonomy of the knowledge sources and they explain that "Content" describes all elements, which are not generated from users.

3.4.1 Social

The social knowledge uses all users and their profiles of the system. This knowledge is a major part of the collaborative algorithm. The opinions of the users are often represented as a $m \times n$ matrix, where m stands for the users and n for the items. The entries in this matrix reflect a rating of an item from a user (Table 3.2 shows this matrix). Other opinion-options, besides ratings, are for example tags or reviews in form of text. RSs can also use demographic data for predictions (Burke and Ramezani, 2011).

	Item 1	Item 2	Item 3
User X	1	4	3
User Y	4	2	1
User Z	1	5	5

Table 3.2: Item-User Matrix

3.4.2 Individual

The individual knowledge of the user is important for the personalization of recommendations. Whether the opinion or demographic data is social or individual is a pure viewpoint. For example, it is individual if a specific user is getting recommendations based on their opinion or demographic data. Otherwise, if other users receive recommendations and use the data from the specific user, it is social. Additional to opinion and demographic data, users can have special requirements. The system must process these requirements that come in different forms, such as in queries, constraints, preferences or as context. Queries are user interface specific, for example to tick a box to specify a particular requirement for the recommendation. Constraints cannot be violated, such as giving no vegan food restaurant recommendations to users which are only eating vegan. On the other hand, preferences can be violated to some degree. For example, a user wants to buy a product below a certain price, e.g. € 10 but would also buy it for € 15 (Felfernig and Burke, 2008).

3.4.3 Content

The content knowledge covers a large area. In some systems, there is only knowledge about the items and their attributes. Such knowledge is considered to be basic and often taken from a database, where the items are stored. Context can be vital in order to determine or match individual requirements, such as finding a restaurant for a business meeting as opposed to find a restaurant for a birthday party. More complex is the domain knowledge such as *means-end*, where the user has a specific item ("means") for a special purpose in mind ("end"), as well as *feature ontology*, which links items to each other, or *constraints*, because items can have constraints that cannot be violated (Burke and Ramezani, 2011).

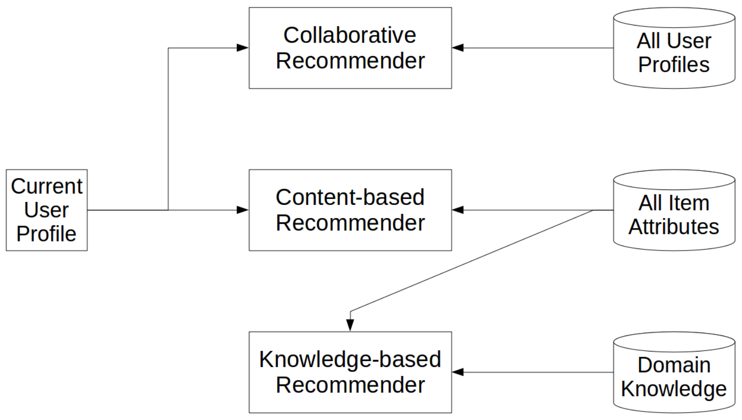

Collaborative Recommender
All User Profiles
Current User Profile
Content-based Recommender
All Item Attributes
Knowledge-based Recommender
Domain Knowledge

Figure 3.10: Recommender and their Knowledge Sources. Adopted from (Burke, 2005)

3.4.4 Knowledge Sources Example

Three approaches of RSs were selected, to give an insight into where they are acquiring knowledge. These three are collaborative, content-based and knowledge-based recommender.

Collaborative recommendations compare the current user profile (individual knowledge 3.4.2) with the user profiles of other users (social knowledge 3.4.1) to find similarities for recommendations. Instead of the user profiles, the content-based recommendation needs the item list with the attributes (content knowledge 3.4.3) and the current user profile. Items are recommended similar to items which the user rated high. Like content-based recommendations, knowledge-based recommendations use the item list as well but in addition it also uses domain knowledge (content knowledge 3.4.3), the connection between the item and the benefit to the user. Figure 3.10 illustrates the three recommender and their knowledge sources graphically (Burke, 2005).

3.5 Netflix Prize

Netflix[1] is an online platform with a subscription service. This service offers streaming movies and TV shows on a variety of devices. The devices must be connected to the internet but the service can be used any time (Gomez-Uribe et al., 2015).

How important it is to research and improve RSs, shows Netflix with its competition: *The Neflix Prize*[2].

In October 2006, Netflix has given access to a large amount of data. This data contains over 100 million anonymous ratings and come from over 480 thousand anonymous subscribers which are randomly chosen. The ratings are collected between the years 1998 and 2005 and consist of a scale from 1 to 5 (Bennett et al., 2007).

The challenge was to develop a RS that has at least a 10% reduction of the Root Mean Square Error (RMSE) compared to the RMSE of the system *Cinematch*, the recommender of Netflix. The team who first reached the 10% mark won $ 1 million (Bell et al., 2007).

The challenge had more than 50, 000 participants from 186 countries. After three years, on September 21th, 2009, the team *BellKor's Pragmatic Chaos (BPC)* won the challenge. They submitted 20 minutes before the team *The Ensemble*, who also managed to get above the 10% mark (Hallinan et al., 2016).

[1]Netflix, 2018b.
[2]Netflix, 2018a.

4 Annotations and Tags

This chapter contains an overview of annotations and tagging and their usage. For understanding the annotations and tags a short detour through the Semantic Web as well through Web 2.0 is given. Further, it is shown how tagging can be used to support or improve RSs.

4.1 Semantic Web

The Semantic Web extends the World Wide Web (WWW) and its data with additional information. This information is called *semantic annotations*. With these annotations, the web is made understandable not only for humans but also for machines to make the automatic processing easier (Chirita et al., 2007).

4.1.1 Semantic Annotation

As already mentioned above, annotations extend the WWW, more precisely semantic annotations are generating metadata that can be used for new and also existing information access methods (Kiryakov et al., 2004).

Semantic annotations are not simple textual annotations which are primarily used by the creators of the data, it identifies relations between concepts. For example, the annotation of “Paris” would connect it to the concept “City” and “Country”. That is because Paris is a city in France which is a country. Therefore, there is no more uncertainty of meaning of the word “Paris” (Uren et al., 2006).

```
...
<MultimediaContent xsi:type="ImageType">
  <Image id="IMG">

    <Region id="1">
      <Semantic>
        <Label><Name> Roosevelt </Name></Label>
      </Semantic>
    </Region>

    <Region id="2">
      <TextAnnotation>
        <KeywordAnnotation><Keyword> Churchill </Keyword></KeywordAnnotation>
      </TextAnnotation>
    </Region>

    <Region id="3">
      <Semantic>
        <Label><Name> Stalin </Name></Label>
      </Semantic>
    </Region>
...
```

Figure 4.1: Annotation Example. Adopted from (Arndt et al., 2007)

Imagine a picture of three well-known personalities like Franklin D. Roosevelt, Winston Churchill and Joseph Stalin on it. Figure 4.1 shows an example of a metadata file for this picture, with semantic and textual annotations. Regions with id 1 to 3 marks sections in the picture annotated with the names of the persons. Only regions with the id 1 and 3 are semantic annotations which means that an authoring tool can connect to other concepts such as the biography of the persons. Region with the id 2 is only a textual annotation and is ignored by these tools (Arndt et al., 2007).

4.2 Web 2.0

The term "Web 2.0" emerged at a conference for the further development of the WWW in the year 2004. One point of the Web 2.0 is, that the users are no longer limited to consuming; they can now participate actively in designing the web through providing content. Other parts of Web 2.0 are new techniques such as Asynchronous JavaScript and XML (Ajax) and Rich Site Summary (RSS) or applications like Wikis and Weblogs. More specifically, Web 2.0 is not just one technique or an application, it is a combination of both (Richter et al., 2007, pp. 4–6).

O'Reilly (2007) shows the difference between Web 1.0 and Web 2.0 based on examples. Some of these examples are shown in Table 4.1.

Web 1.0	Web 2.0
DoubleClick	Google AdSense
Ofoto	Flickr
Akamai	BitTorrent
mp3.com	Napster
Britannica Online	Wikipedia
personal websites	blogging
publishing	participation
directories	tagging

Table 4.1: Examples of differences between Web 1.0 and Web 2.0. Adopted from (O'Reilly, 2007)

4.2.1 Social Web and Social Software

The social web is a part of Web 2.0, which focuses on the support and interaction of social structures (Ebersbach et al., 2016). It stands for a category of websites and applications, where the contributions of the users are in the center (Gruber, 2008).

Applications or dynamic websites which are using the web as a channel for communication are called social software. The primary function of this software is not to connect servers or send data between them, it is the connection and communication between users, users and a community or among multiple communities. The difference to the social web is, that the social web also includes the data of users, the used application and data of the social bond between users (Ebersbach et al., 2016).

Schmidt (2006) presents three basic functions of social software. These functions are listed below, including a short description:

- *Information management*. Software for searching, managing and sharing information.
- *Identity management*. Software for the presentation of its own.

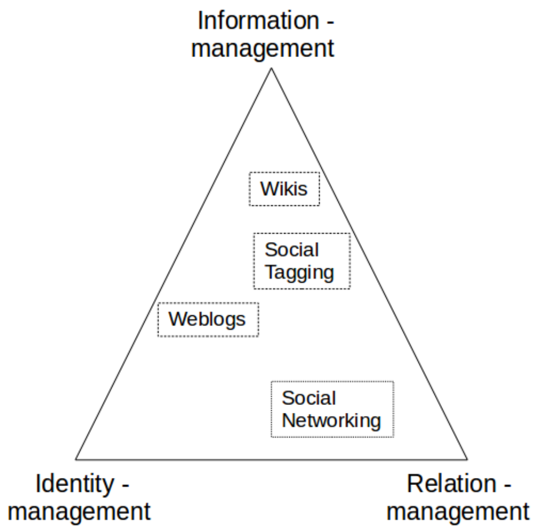


Figure 4.2: Social Software Triangle. Adopted from (Richter et al., 2007)

- *Relation management*. Software for managing or finding new relationships.

Not every social software covers all functions equally. In the next Section 4.2.2, different types of social software are explained in more detail.

4.2.2 Types of Social Software

Richter et al. (2007, pp. 11–12) distinguishes between four types of social software, such as Weblogs, Wikis, Social Networking and Social Tagging.

Richter et al. (2007, p. 12) defines a “Social Software Triangle” that assigns the four types to the basic functions, which are explained in Section 4.2.1. Figure 4.2 shows the assignment of the four types graphically. Weblogs, wikis and social networks are explained in more detail in the following subsections. Social tagging is explained in Section 4.3.

Weblogs

A “Blog”, the short form for Weblog, is a website in which users, called “blogger”, can publish or “post” almost everything. The published material, called “posts”, are mostly textual but can also contain photos or multimedia content. The posts are typically in reversed order, which means the newest post is on the top of the website. Many blogs contain hyperlinks to external websites and also a comment section, where the users can leave a comment to a post. Blogs in the form as today started around 1997 with Dave Winer’s *Scripting News*[1] blog. Like the Scripting News blog, most blogs are online diaries or personal journals (Nardi et al., 2004).

Creating and maintaining blogs is not very difficult. Users can benefit from certain software, so called “blogware”. *WordPress*[2] and *Blogger*[3] are two blogwares which are supporting the user in creating and managing blogs. The user does not need technical knowledge for this, as opposed to other platforms such as *Movable Type*[4], where the user should know how to install and manage the software on a webserver (Murugesan, 2007).

Wikis

The word wiki derived from the Hawaiian word *wiki-wiki*, which means quick. A wiki allows users not only to view but also to create and edit the content, which makes a wiki a collaborative website. It simplifies the process of creating a Hypertext Markup Language (HTML) page with information. An additional system that records changes of the pages in the wiki can be used to reset the page to an older version if needed. It is possible that some pages can only be edited by registered users or a group of members although everyone can view the page. There are also wikis without a restriction for editing and viewing. Wikis are a popular form to spread information and knowledge among people, because they can share information and discuss it in groups (Parker et al., 2007).

[1]Winer, 2018.
[2]WordPress, 2018.
[3]Blogger, 2018.
[4]Type, 2018.

Wikis gained popularity through the encyclopedia *Wikipedia*[5], the biggest wiki online. Furthermore, wikis are used for a collection of websites like *wikiindex*[6], which contains a collection of other wikis (Richter et al., 2007, pp. 19–24).

Social Networking

Social networks such as *Facebook*[7], *myspace*[8] or business networks like *XING*[9] are community sites, where users can connect with other people. It usually starts with creating a profile. In this profile, information such as birthday, address, hometown, interests and more, can be included. Instead of using their real names, users can also use a pseudonym. In order to get in contact with other people, a friend request must be sent to the other user. This request must be accepted to establish a link between the two profiles. Mostly, users whose friend requests are accepted, are managed in friend lists. The primary usage for this community sites is to stay in contact with others or meet new people. In addition, popular uses are sharing photos and information or get updates on the activity of friends (Dwyer et al., 2007).

4.3 Tagging

4.3.1 Tags

Providing online content with keywords can be helpful for later filtering, searching and managing. These keywords are called “tags”. The collaborative form of adding tags to content is called “collaborative tagging” or “folksonomy”, which is made up of two words, “folk” and “taxonomy”. Traditionally, categorizing was performed by the author of the document or an authority, such as the librarian of digital libraries. In collaborative

[5]Wikipedia, 2018.
[6]WikiIndex, 2018.
[7]Facebook, 2018.
[8]Myspace, 2018.
[9]XING, 2018.

tagging, anyone can add tags to content, which is most useful when there is no authority or there is too much content (Golder et al., 2006). An application of collaborative tagging is the social bookmarking system Pinboard[10]. User can tag their bookmarks and share them among each other (Ovadia, 2012).

Golder et al. (2006) identified several functions on how tags can describe content of bookmarks:

- *Identifying what (or who) it is about*. Describes the topic of the bookmark, for example "Sport".
- *Identifying what it is*. Describes the kind of bookmark, for example "Blog".
- *Identifying who owns it*. Identifying who is the creator or the owner of the bookmark content, for example "John Doe".
- *Refining Categories*. Instead of creating new categories, refine old ones, best with round numbers, for example "Categorie100".
- *Identifying qualities or characteristics*. Describes the opinion of the tagger, for example "Good".
- *Self reference*. Describes the relation to the tagger, for example "myBlog".
- *Task organizing*. Describes the task which is performed on the content, for example "readLater".

4.3.2 Social Tagging Systems

Social tagging systems such as *Flickr*[11], *CiteULike*[12] or *Last.fm*[13] allows users to share, tag and manage their resources. Tags in these systems are used as links to resources, tagged by the owner and other people. Tagging systems also struggle with problems such as *ambiguity*, *synonyms* and *discrepancies in granularity*. An example for the ambiguity problem is the tag "bat". It can refer to a flying little animal or a wooden stick for playing baseball.

[10]Pinboard, 2018.
[11]Flickr, 2018.
[12]CiteULike, 2018.
[13]Last.fm, 2018.

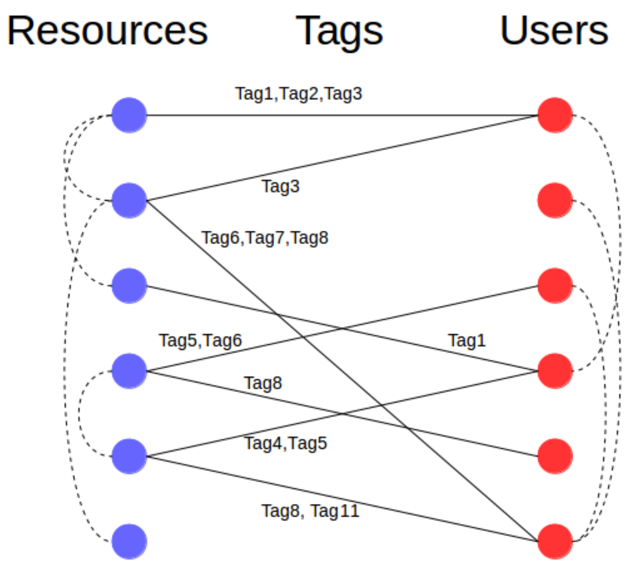


Figure 4.3: Model of a Tagging System. Adopted from (Marlow et al., 2006)

Synonyms are making it hard to retrieve all desired resources, for example the user wants all resources relevant to "tasty". Some resources may be tagged "delicious", so they are ignored and not retrieved. Hence, in order to get all resources, the users would have to know all possible synonyms of a tag. The problem with the level of granularity is, that tags may be too specific for some users or the other way round. An example for the level of granularity is the tag "JavaScript". It may be too specific, but the tag "programming" would not be sufficient (Specia et al., 2007).

In Figure 4.3, a conceptual model of a tagging system is presented. This model consists of three elements: *users*, *tags* and *resources*. Users can assign tags to resources and also can be connected to other users via social software. Furthermore, the resources can be connected to each other, for example through links between web pages (Marlow et al., 2006).

Marlow et al. (2006) developed two tagging taxonomies which analyze how the resultant tags are influenced in tagging systems. This comprises the characteristics of system designs on the one hand and the user incentives and motivations on the other hand. The user incentives and the key dimensions of the systems design are listed below with a short description.

System Design

- *Tagging rights*. Tagging systems can restrict the allocation of tags. Is the restriction set to *free-for-all tagging*, then any user can tag any resource. Otherwise, if it is set to *self-tagging*, user can only give tags to resources which they created. The restriction can also vary between these two levels, e.g. the system can decide which resources can be tagged by which user. Restriction can also be set for deleting tags, for example not every user can delete any tag of a resource.
- *Tagging support*. Tagging systems can have different levels of support for the user during the tagging process. The supporting level can be *blind tagging*, *viewable tagging* or *suggestive tagging*. Blind tagging means that the user cannot see assigned tags from other users while tagging the resources. Viewable tagging allows users to look at the tags already assigned to the resources. If tags are offered to the user while tagging, then the system uses suggestive tagging. The suggested tags can be generated from already existing tags which the user used or from tags of other users which assigned tags to the resources.
- *Aggregation*. There are two approaches, namely the *bag-model* and the *set-model*. The first approach allows a multiplicity of tags for one resource, which can result in duplicated tags. In the second approach the system requests a group of users to collectively tag a resource. This prevents the duplication of tags.
- *Type of object*. An important consideration is the type of the resource. Well-known types are web pages, blog posts, images, videos or audio objects. In fact, any object can be tagged if the object can be virtually represented.
- *Source of material*. The resources provided can come from users or from the systems itself. The system can also be an open system, which allows to tag any web resource, such as links.
- *Resource connectivity*. The resources in the system can be linked to each other. There are three categories: *linked*, *grouped* or *none*.
- *Social connectivity*. Users can also be linked to each other via social software. Similarly to resources, the connection between users can be categorized in *linked*, *grouped* or *none*.

User Incentives

- *Future retrieval*. A resource or a collection of resources are tagged for personal retrieval. For example, user tag a collection of books with "to read" for reading at another time or tag songs with "power-walk" and create a playlist for sport activities.
- *Contribution and sharing*. Resources can be shared with friends, family or any other user with the use of tags, e.g. tag vacation websites for the family.
- *Attract attention*. Own resources can be tagged with popular tags to get other users to view them.
- *Play and competition*. Users create tags based on a set of rules. In some systems, the systems itself create this rules, like in tag games where users are stimulated to tag a resource which others might also tag. In other systems, a group of users develops the rules, e.g. to seek all resources with a particular feature.
- *Self presentation*. Users can use the tagging to write their own identity into the system or rather leave a personal mark on a particular resource. For instance, the tag "seen live" marks an identity or personal relation to the resource.
- *Opinion expression*. Tags can express a personal opinion of a resource. As a result, users convey an opinion to other users.

4.3.3 Tags and Recommendation Systems

There are two aspects of the relationship between tags and RSs which are described below in more detail. The first is the explanation of how recommendation technologies can be used for the recommendation of tags. The second is how tags and their information about the resources can be integrated in the recommendation process to suggest useful items to the user.

Tag Recommendations

As described in Section 4.3.1, tags can be assigned to resources for organizational purpose, filtering or sharing. However, not all resources have tags, because tagging is a time-consuming task and not every user is going through the process of tagging. In order to support and to encourage users to tag their resources, tag recommendation systems are used. There are two main types of these systems: *personalized* and *non-personalized*. The personalized systems take the users interests and preferences into account, whereas the non-personalized systems do not (Nguyen et al., 2017).

Song et al. (2011) distinguish between two approaches for tag recommendation systems: the *user-centered* and the *document-centered* approaches. The user-centered approach recommends tags based on the historical tagging data and similar users, similar to CF techniques. On the other hand, the recommendations of the document-centered approach are based on the data from the document itself, for example author or title. This approach also applies to other object types, such as videos or images and not only to documents. Thus, it is better to call it a content-based technique.

In addition to content-based, graph-based is one of the most popular techniques for tagging recommendations. A graph-based algorithm is *FolkRank*[14] which is based on *PageRank*[15]. The basic idea is that resources, which are marked as important by users that are important or influential, become important itself. Furthermore, combinations of the two techniques content-based and graph-based are possible (Shu et al., 2010).

Recommendations using Tag Information

RSs which use tags as additional resources for creating more effective recommendation algorithms are called *tag-based* or *tag-aware* recommender systems. The tags can reflect on the user's opinion and preferences because the user can freely choose and assign tags. Further, tags can be used to evaluate the qualities of the resources because they show the semantic

[14]Jäschke et al., 2007.
[15]Brin et al., 2012.

relations between them. In addition, the co-occurring properties of tags make it easier to find similar resources (clustering items) or users (building user communities). However, even if they are useful for managing and searching resources, not all tags are suitable for using in RSs. Tags have their limitations such as *ambiguity* or *synonyms* (see Section 4.3.2). There are many researches concerned with solving the problems with tags, such as the *clustering-based* methods from Shepitsen et al. (2008) and Capocci et al. (2008), who are trying to alleviate the word reduction problem. Other researchers, like Mika (2007) and Kim et al. (2008), use *ontology-based* algorithms to reveal the semantic relations among tags. Another example is the research from Zhang et al. (2010) and Liu et al. (2010), which uses *graph-based* methods to solve the sparsity problem (Zhang et al., 2011).

For the RSs, there are two approaches to use tags and their information in the recommendation process. One of them is using tag as additional information about the content of the resource. This information can be used by content-based RSs for suggesting items. The other approach is to extend the user-item matrix with a third tag dimension. This approach is suitable for collaborative techniques (Jannach et al., 2010, p. 262).

An example for a content-based approach is provided by Szomszor et al. (2007), who base recommendations on *tag-clouds*. They are using two datasets for generating the tag cloud, which they call the *rating tag-cloud*. They use the ratings of movies from the Netflix rating database and the movie tagging data from Internet Movie Database (IMDb)[16]. These two datasets can be combined through an easy string matching with the titles of the movies. The rating scale lies between 1 to 5, where 1 is bad and 5 is very good. Based on these ratings and the tagging data, this method tries to estimate the future ratings of not rated movies by the user. Thus, a rating tag-cloud for the user is created. Figure 4.4 shows an example of a rating tag-cloud for a rating of 5 for a user. It is possible to generate different tag-clouds for the user for every rating. The size of tags results from the frequency of occurrence, for example in Figure 4.4, most of the movies with the tag “Fantasy” were rated with 5. Few movies with a rating of 5 had the tag “Horror”.

[16]IMDb, 2018.

Figure 4.4: Example of a Tag-cloud. Adopted from (Szomszor et al., 2007)

Based on this, the system can build a list of recommendations with movies that are not rated but have the same tags as movies with a high rating because the user might also like these movies. Szomszor et al. (2007) have also shown that a RS can be built purely on tags that are assigned by collaborative tagging.

As already mentioned, a collaborative approach usually extends the user-item matrix (see Section 3.4.1) by adding a third dimension. An example for this is provided by Tso-Sutter et al. (2008). They handle the three dimensionality (<*user, item, tag*>) by projecting it as a three two-dimensional problem (<*user, item*> and <*item, tag*> and <*user, tag*>). This can be done by expanding the user-item matrix horizontally and vertically with *user tags* and *item tags*, which is shown in Figure 4.5. *User tags* are viewed as items and *item tags* are viewed as users in the user-item matrix. The *user tags* are tags from a certain user who tagged an item, where the *item tags* describe a particular item. In addition, it is possible to use clustering methods to group similar tags together, instead of viewing each single tag as a user or an item. Tso-Sutter et al. (2008) then recompute and fuse user- and item-based CF with the new user-item matrix. They show that this approach is more effective as standard baseline models without the use of tag information.

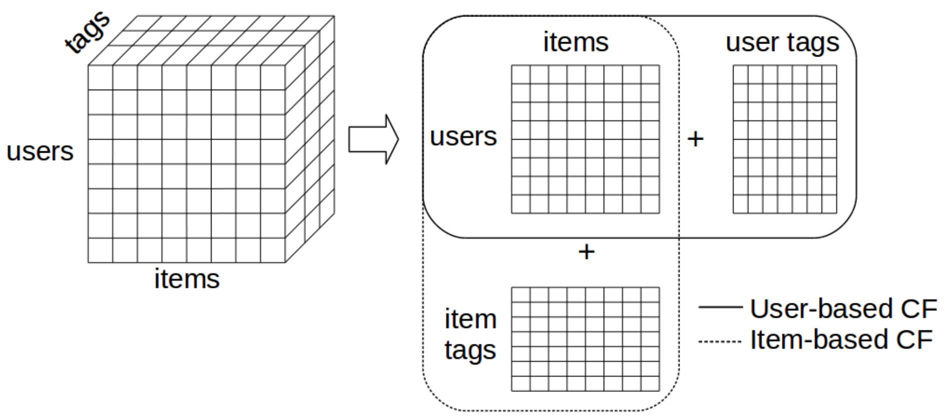


Figure 4.5: Extended user-item matrix. Adopted from (Tso-Sutter et al., 2008)

5 Practical Part

The practical part of this thesis contains the implementation of the two previously discussed systems: the tagging system and the recommendation system. Furthermore, this chapter includes a more detailed explanation of the aforementioned problem and how the research questions were developed. After this, there is a description of the already existing implementations. Last but not least, it is explained and shown how the tagging and the recommendation system are implemented. The analysis of the data which are obtained through the implementation, is discussed in Chapter 6.

5.1 Methodology

5.1.1 Research Questions

Online systems need to be constantly developed and extended with new features for the users. Catrobat is no exception.

Since the system consists of an app (client side) and a server, there are two possibilities to enhance the system and offer users new features. First, in the app itself, where for example new bricks for programming can be added or the design can be changed with new themes. Second, and that is what this thesis is about, to enhance the server and thus the community site. Since it is a site where people interact with each other, it is only obvious that the focus should be on user features. More precisely, on how to gain more users, raise the activity and download numbers of the community site. To achieve these goals, a tagging and recommendation system was implemented in order to find answers to the following research questions:

1. *Can the activity on the community site be raised by adding a tagging system, which helps users to sort and find programs?*
2. *Can the download numbers on the community site be increased by implementing a RS based on tags?*

5.1.2 Activity and Download Numbers

To increase the activity and raise the download numbers, several options were considered, such as to try to increase the activity by changing the design or to organize more events, like the *Galaxy Game Jam*[1] where users can participate and compete against each other. However, these options are not effective enough, as the design changes may not appeal to all users and the events are not permanent, but only available for some time.

As already mentioned, the main focus of the community site is to interact with each other. Thus, possible interaction points were examined more closely in order to increase the activity. One of these points, which is already implemented, is the comment section, where users can communicate directly with one another. However, not every user is interested in commenting and discussing. Since, the main interaction point is uploading your own created programs and download programs from other users, the implementation of a tagging system would be the best choice. For the requirements of the tagging system see further below in Section 5.2.1 and the implementation can be found in Section 5.3.3.

Additionally, further considerations had to be made to increase the download numbers. For this, a closer look at the download process was necessary. Users can download the program directly to the app or create a standalone program and install it on their mobile device. If the program it is tried out and liked by the user, they will probably download more of this kind of program. Therefore, a RS is a good choice in order to increase downloads. The RS can use the tag information to generate a list of similar programs. This system is also called a tag-based recommendation system (see Section 4.3.3 for more details). The requirements of the RS is discussed in Section 5.2.2 and the implementation can be found in Section 5.3.4.

[1] Catrobat, 2018c.

5.2 Requirements and Evaluation

5.2.1 Tagging System

The tagging system helps the user to categorize and group programs. Thereby, programs which have the same tag can be found easier. By clicking on the tag, other programs with this tag are displayed. Furthermore, users can directly search with the search bars after programs by simply typing the tag name in.

In order to get the users acquainted with the new system, only six tags were added and users are restricted to these tags. The six tags are: *Game*, *Animation*, *Story*, *Music*, *Art* and *Experimental*. To overcome the problem with "tag spam", which means to tag a program with irrelevant tags, a user can only use up to three tags and of course giving none is also an option. This concept is based on the tagging of Scratch[2].

To make the tags available to several users from different countries, multi-language support is necessary. For starters, the tags were only translated into three different languages, such as *German*, *Italian* and *French*.

As additional information to the tags, extensions of the programs should be used. Users can not give the extension tag to the program, because extensions in Catrobat are special bricks. These bricks are used to control external devices such as a drone or a Lego Mindstorm robot. Thus, these bricks are automatically recognized by the system and the extension tag is added after the upload. The extensions do not need to be translated because they are proper names, such as *Arduino*, *Drone*, *Lego*, *Phiro* and *Raspberry Pi*.

The implementation of the tagging system is created in a way that it is easily expandable. An overview of the implementation can be found in Section 5.3.3.

[2]Scratch, 2018a.

5.2.2 Recommendation System

A RS generates a ranked list of items that may be of interest to the user. Many algorithms and approaches were investigated and some are explained in more detail in Chapter 3. Since users are not always active on a regular basis, too little data may be available to use a collaborative system (Section 3.2.1). The RS should be available to all users, whether the user is regularly active on the page or uses it only now and then. Therefore, the system for the community site of Catrobat should use tag information for generating the recommendations. This tag-based recommender system shows similar programs on the detail site of the currently watched program. More information about tag-based recommender system can be found in Section 4.3.3.

The ranking of the recommended programs is based on their similarity. If two or more programs are ranked the same, the order among them is random. An overview of the implementation can be found in Section 5.3.4.

5.2.3 Evaluation Methods

In order to evaluate the two systems and to answer the research questions, two different evaluation approaches are used. First, the evaluation is being done via a tracking system. The data is collected via the clicks on tags as well as the clicks on the recommendation system. In addition, Google Analytics[3] tracks the activity on the community site.

The second approach is rather a testing approach than a tracking, as it measures the impact of the RS on the downloads via a *A/B test*. This test, also called *split-run testing*, compares two versions (A and B) with each other (Kohavi et al., 2015). In this case, there are two versions of the community site. One without the RS, version A, and one with it, version B. Version A is only visible to Russia. Version B is visible to all other countries, whereby Germany is being used for the evaluation, as it is a comparable country based on the fact that it has a similar user base as Russia. Figure 5.1 shows an example of a user flow to the program details page, first visiting the

[3]Google, 2018c.

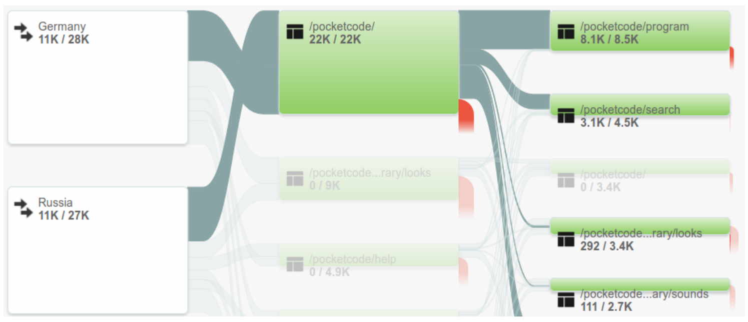


Figure 5.1: User Flow Chart (Source: Googly Analytics)

index page[4], then clicking on a program and going to detail site[5]. In total Russia had 11,000 sessions while Germany had 11,000 sessions accessing the index page. From these sessions, 4,800 from Russia and 3,700 from Germany were browsing further to the details page of a program.

5.3 Implementations

This section provides an overview of the existing implementation methods, including the implementation of the tagging and recommendation systems. Since giving a detailed explanation of the implementations would be beyond of the scope of this thesis, only the core functions are explained. The complete implementation can be viewed on **Github** - `https://github.com/Catrobat/Catroweb-Symfony`.

5.3.1 General Information of the System

The complete system is build up on the web framework Symfony (version 2.7) as already mentioned in Chapter 2. Symfony uses Doctrine[6] (version 2.5) for communication with the Database (DB). Unlike Structured Query Language (SQL) Doctrine uses an object-oriented dialect to write queries.

[4]https://share.catrob.at/pocketcode/
[5]https://share.catrob.at/pocketcode/program/XXX
[6]Doctrine, 2018.

Listing 5.1 shows a shortened version of the tag entity. The columns of the tables in a DB are linked to a php object class with annotations.

```php
/**
 * @ORM\Entity
 * @ORM\HasLifecycleCallbacks
 * @ORM\Table(name="tags")
 * @ORM\Entity(repositoryClass="Catrobat\AppBundle\Entity\
    TagRepository")
 */
class Tag
{
  /**
    * @ORM\Id
    * @ORM\Column(type="integer")
    * @ORM\GeneratedValue(strategy="AUTO")
    */
  protected $id;

  /**
    * @ORM\Column(type="string", nullable=true)
    */
  protected $en;

  /**
    * @ORM\Column(type="string", nullable=true)
    */
  protected $de;

  /**
    * @var \Doctrine\Common\Collections\Collection|Program[]
    *
    * @ORM\ManyToMany(targetEntity="\Catrobat\AppBundle\Entity\
      Program", mappedBy="tags")
    */
  protected $programs;

  //...
}
```

Listing 5.1: Tag Entity

This is more comfortable to work with than the standard SQL queries. New objects can be easily created (`$tag1 = new Tag()`), filled with data (`$tag->setEN`

`('Tag')`) and saved in the DB (`$entityManager->persist($tag1) $entityManager ->flush()`). Querying objects from the DB is possible with the entity manager (`$tag = $repository->find($tagId)`) or with more complex queries with the *Query Builder*[7](`$QueryBuilder->select('t')->from('Tag', 't')->where('t.id = ?1)`). It is also possible to write queries with Doctrine's native SQL-like language called Doctrine Query Language (DQL)[8](`$tag = $entityManager-> createQuery('SELECT t FROM Tag t WHERE t.id=:id)->setParameter('id',1)->getResult ()')`). Most queries in the project are built with the *Query Builder*. Special functions, such as the search function, are written with DQL.

5.3.2 Existing Implementation

Some features were already implemented by the work of Stefan Jaindl during his master thesis[9]. Thus, the download of all programs can already be logged and there is an own section in the admin area with the logged data.

For this master thesis, this feature is used as a basis for further logging for the tagging and recommendation systems. The data consists of information of all downloads. For analysis and evaluation, only the download behavior of the two countries (Germany and Russia) are important. Furthermore, the data must be extended with the name of the program from which the downloaded program has been recommended and for the activity logging with information of the tags.

5.3.3 Implementation of Tagging System

Users can choose tags during the upload process, which are then shown on the community site. Therefore, the implementation of the tagging system is split into two tasks: implementation on the client side and on the server side. The client side is uploading and chooses the tags, whereas the server side is getting the uploaded program and manages the tags.

[7]Builder, 2018.
[8]Language, 2018.
[9]Jaindl, 2016.

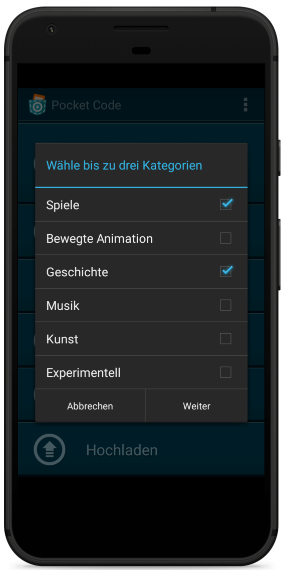
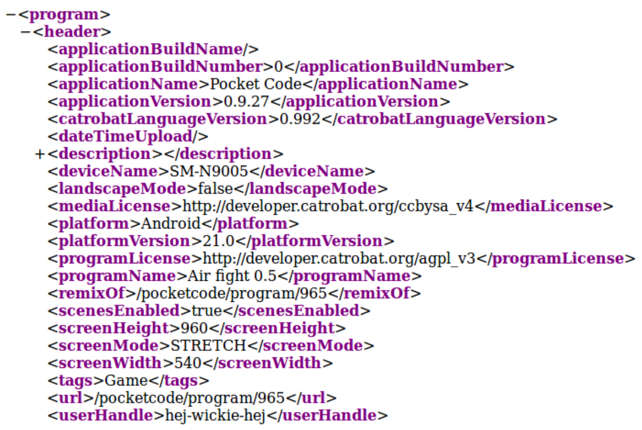


Figure 5.2: Additional Tag Layer

Client Side

The implementation on the client side is done by the Catroid team. They added an additional layer in the uploading process, which is shown in Figure 5.2. To get the tags which are currently supported, the client makes a Hypertext Transfer Protocol (HTTP) request to the server. In the appendix a reference to the implemented web Application Programming Interface (API) functions is given. From there, up to three tags can be chosen, while choosing no tag is also an option.

Afterwards, the tags are added to the program's Extensible Markup Language (XML) files. In these files, all relevant information is stored, such as the Catrobat language version, creators name and program name. Figure 5.3 shows a part of an XML file. The server can extract all information from the XML after the program is uploaded.



```
-<program>
 -<header>
    <applicationBuildName/>
    <applicationBuildNumber>0</applicationBuildNumber>
    <applicationName>Pocket Code</applicationName>
    <applicationVersion>0.9.27</applicationVersion>
    <catrobatLanguageVersion>0.992</catrobatLanguageVersion>
    <dateTimeUpload/>
   +<description></description>
    <deviceName>SM-N9005</deviceName>
    <landscapeMode>false</landscapeMode>
    <mediaLicense>http://developer.catrobat.org/ccbysa_v4</mediaLicense>
    <platform>Android</platform>
    <platformVersion>21.0</platformVersion>
    <programLicense>http://developer.catrobat.org/agpl_v3</programLicense>
    <programName>Air fight 0.5</programName>
    <remixOf>/pocketcode/program/965</remixOf>
    <scenesEnabled>true</scenesEnabled>
    <screenHeight>960</screenHeight>
    <screenMode>STRETCH</screenMode>
    <screenWidth>540</screenWidth>
    <tags>Game</tags>
    <url>/pocketcode/program/965</url>
    <userHandle>hej-wickie-hej</userHandle>
```

Figure 5.3: XML Snippet

Server Side

The implementation on the server side is structured in various steps, starting with the uploading process up to managing and displaying tags and extensions on the community site. Below is a list with all implemented steps. After that, the steps are explained in more detail with code for better understanding.

- *Create relations between tags, extensions and the uploaded program in the upload process*. This step includes the extraction of the tags from the XML, creation of extensions and also the creation of relations between tags, extensions and programs in the DB.
- *Displaying tags and extensions*. Display the tags and extensions which are connected to the program on the details page of the program.
- *Managing extensions*. In the admin area, extensions can be added and updated. Tags are translated in the supported languages. Therefore, tags can only be added if they are translated and saved in the DB.
- *Searching programs with tags and extensions*. Tags and extension names can be used to look for programs with these tags and extensions or list programs by clicking on them.

The uploading process starts on the server side with the upload of the program via the API (see appendix for reference). Listing 5.2 shows how tags are extracted and added to the DB during the uploading process. The code is shortened and in-line commentary was added for better understanding. Important to mention is that the code is written in such way that it is not possible to add tags which are not in the DB and it is not possible to add more than three tags. The *addTag* call in line 21 creates the relation between the program and the tag, for example for the program “HupiFlupi” the tag “Game” is added, and also for the tag “Game” the program is added.

```
 1  //...
 2  public function addTags($program, $extracted_file, $language)
 3  {
 4    //Gets all supported languages from the database. If the
          language is not supported, English is set as default.
 5    $metadata = $this->entity_manager->getClassMetadata('Catrobat
          \AppBundle\Entity\Tag')->getFieldNames();
 6    if (!in_array($language, $metadata))
 7      $language = 'en';
 8
 9    //Get the tags from the extracted program.
10    $tags = $extracted_file->getTags();
11    if (!empty($tags))
12    {
13      $i = 0;
14      foreach ($tags as $tag)
15      {
16        //Lookup, if the tag is in the database.
17        $db_tag = $this->tag_repository->findOneBy([$language =>
              $tag]);
18        if ($db_tag != null)
19        {
20          //If the tag is in the database, the tag is added to
                the program. A relation is created.
21          $program->addTag($db_tag);
22          $i++;
23        }
24        //Not more than three tags can be added to a program.
25        if ($i == 3)
26          break;
27      }
28    }
29  }
30  //...
```

Listing 5.2: Adding Tags

Unlike adding tags to a program, extensions and the relation to programs is created automatically. Therefore, the adding mechanism is implemented as a listener. After the successful upload but before the program is added to the DB, an event is dispatched. The event is called *catrobat.program.before.persist*. The extension listener is called and the program's XML is parsed for special bricks, like Lego or drone bricks. These bricks have an attribute which is called *category*. This attribute has a value like *LEGO_EV3_SPEED*. The listener gets all nodes with this attribute, extracts the value and shortens it to the first underscore, the prefix. After that, the listener browses the DB whether this extension exists or not. If it exists, the relation between the extension and the program is established. The code of the parsing for the extensions is shown with some in-line commentary in shortened form in Listing 5.3.

```
public function checkExtension($extracted_file, $program)
{
  $xml = $extracted_file->getProgramXmlProperties();

  //Get all nodes with the attribute category.
  $xpath = '//@category';
  $nodes = $xml->xpath($xpath);

  $program->removeAllExtensions();
  if (empty($nodes))
    return;

  //Extaction of the prefix.
  $prefixes = array_map(function ($element) { return explode("_
     ", $element['category'], 2)[0]; }, $nodes);
  $prefixes = array_unique($prefixes);

  //Get all saved extensions from the database.
  $extensions = $this->extension_repository->findAll();

  //The extension is added to the program.
  foreach ($extensions as $extension) {
    if (in_array($extension->getPrefix(), $prefixes )) {
      $program->addExtension($extension);
    }
  }
}
```

Listing 5.3: Adding Extensions

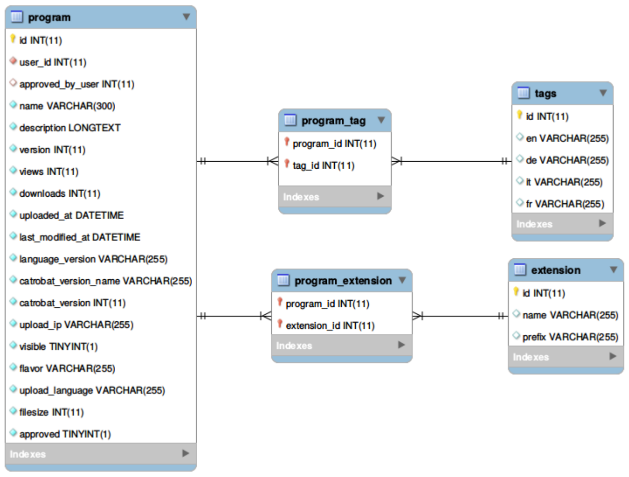
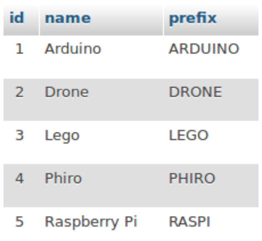


Figure 5.4: Database Schema of Programs, Tags and Extensions

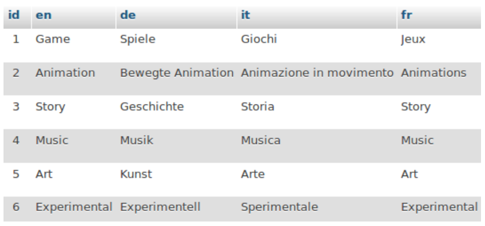
Tags and extensions have a so called *many-to-many* relationship which means that one program can have many tags and one tag can be assigned to many programs. This is also true for extensions. Figure 5.4 shows the DB schema with the *many-to-many* relationship between programs and tags, also programs and extensions. The columns of programs are shortened for a better overview.

After building the relationships between the programs via extensions and tags, they can be a useful tool to search for other programs which are also tagged with the same searched tag or extension. On the details page of the programs, the tags and extensions are designed as buttons. With a click on those buttons, the search site is called and an API request is made. The server makes a query and gets all programs with the searched tag or extension. The programs are sorted in descending order by the upload time, which means the newer programs are at the top of the list. Furthermore, the search was improved to include tags and extensions. For this to work, the already existing search query was extended with an additional query. The new query also searches in all supported languages. Therefore, there is no difference in searching for "Game" or the German word "Spiele". Figure 5.5 shows the structure of the tags and extensions table in the DB. For now, four languages, six tags and six extensions are supported.



id	name	prefix
1	Arduino	ARDUINO
2	Drone	DRONE
3	Lego	LEGO
4	Phiro	PHIRO
5	Raspberry Pi	RASPI

(a) Extensions



id	en	de	it	fr
1	Game	Spiele	Giochi	Jeux
2	Animation	Bewegte Animation	Animazione in movimento	Animations
3	Story	Geschichte	Storia	Story
4	Music	Musik	Musica	Music
5	Art	Kunst	Arte	Art
6	Experimental	Experimentell	Sperimentale	Experimental

(b) Tags

Figure 5.5: Structure of Tags and Extensions

The translations are made by contributors all over the world and managed within *crowdin*[10]. After translating, the translations can be downloaded as *yml* files and imported into the project. Each supported language has its own file. The file is structured in identifiers and translations. An identifier can have many sub-identifiers, for example the identifier "tags" has a sub-identifier "constant" which also has a sub-identifier "tag1". This would be "tags.constant.tag1". This makes it possible to create groups. Currently, there are five tags, "tag1" to "tag5" in "tags.constant". In the German file, the first identifier "tag1" has an associated translation "Spiele", the second has "Bewegte Animation" and so on as can be seen in Figure 5.5b. The template engine *twig*[11], which is used for front-end programming, calls the translations for the tags from the *yml* files. The call has the form "("tags.constant.tag" tag.id)—trans(, "catroweb")". The id in the DB is also the id in the translations (tag1, tag2, ...). Extensions, such as "Lego" are not translated because the names are proper names.

For managing the extensions, new sections are added in the admin area. There it is possible to add and change existing extensions. To add a new extension, only a name must be assigned and the prefix should be known. After an upload of a program, the server automatically searches for the newly added extension. If the prefix changes of one or more extensions, it can be updated in the admin area by clicking on "edit". When an extension is added, it is possible that programs were already uploaded with this extension. To create a relationship between the program and extensions,

[10]Crowdin, 2018.
[11]Twig, 2018.

a function was implemented which deletes all existing relationships and scans all uploaded programs anew to search for the added extensions in the system.

5.3.4 Implementation of Recommendation System

The implementation of the RS is based on the implementation of the tagging system described in the section above. More precisely, the tags of the programs are used to generate a list of similar programs. After accessing the details site of a program, the RS is initialized via an API request. The server gets the id of the program for which the recommendation should be made. Generating the recommendation or - more specific - an array of programs is the core of this RS, which consists of a DQL query. The function with this query is shown in Listing 5.4. The code is shortened for better reading.

```
public function getRecommendedProgramsById($id, $flavor = '
   pocketcode', $limit, $offset)
{
  //Getting tag and extension ids
  $tag_ids = (...);
  $extensions_id = (...);

  //Building the dql string for the query
  $dql = "
      SELECT COUNT(e.id) cnt, e.id
      FROM Catrobat\AppBundle\Entity\Program e
      LEFT JOIN e.tags t
      LEFT JOIN e.extensions x
      WHERE (
        t.id IN (:tag_ids)
        OR
        x.id IN (:extension_ids)
      )
      AND e.flavor = :flavor
      AND e.id != :pid
      AND e.visible = TRUE
      GROUP BY e.id
      ORDER BY cnt DESC
      ";

  //Creating the query and setting the parameters
  $qb_program = $this->createQueryBuilder('e');
  $q2 = $qb_program->getEntityManager()->createQuery($dql);
  $q2->setParameters(
  [
    'pid' => $id,
    'tag_ids' => $tag_ids,
    'extension_ids' => $extensions_id,
    'flavor' => $flavor
  ]);
  $q2->setFirstResult($offset);
  $q2->setMaxResults($limit);

  //Getting only the ids of the recommended programs
  $id_list = array_map(function ($value)
  {
    return $value['id'];
  }, $q2->getResult());

  //Fetch the program object and returning the array with the
     recommendations
  $programs = [];
  foreach ($id_list as $id)
    array_push($programs, $this->find($id));

  return $programs;
}
```

Listing 5.4: Core of the Recommender

First, tag and extension ids of the program are requested with a simple query from the DB. After that, the DQL is constructed which should get only an array of ids of similar programs because the query is faster with only fetching numbers instead of whole objects. Simply put, the DQL fetches an array of ids of programs and the more the program has the same tags and extensions, the more often its id occurs in the array. Afterwards, the same ids are grouped and their frequency is counted. The array is ranked according to this frequency in descending order. Finally, this array of program ids is iterated over and the program objects for the recommendations are fetched and returned via the API. This array of programs objects is displayed on the detail site of the program shown in Figure 5.6.

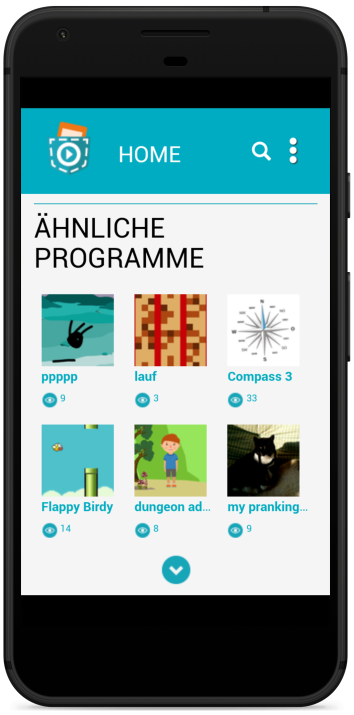


Figure 5.6: Recommender Section on Detail Site

6 Results

This chapter contains program statistics and some general information about the tags and extensions used on the community site. After this, the evaluation of the results from the implementation of the tagging and recommendation system is presented followed by a discussion of these results. Because of the small usage of extensions, they are not included in the results.

6.1 General Statistics and Information

The data used for this thesis evaluation is provided by DB and some parts are provided by activity measurement from Google Analytics. The tagging system was implemented into the main system on June 1st, 2016 and the active logging of the clicks on January 21th, 2017. The RS is online since January 1st, 2017. Six days later, the cross validation between German and Russia was added. The observation period for the evaluation started on February 1st, 2017 and ended on October 31th, 2017 - enough time for the user to get used to the system.

The following subsections will show some statistics of the community site, such as program statistics and the distribution of the tags and extensions. Furthermore, only programs that are visible and not marked as inappropriate are considered. Uploaded programs include newly uploaded and re-uploaded programs unless it is explicitly mentioned. Re-uploaded programs are programs which were already online and were then uploaded again. The purpose of re-uploading lies in changing things, for example change the code or the description. Programs can also be re-uploaded in order to add tags or extensions.

6.1.1 Program Statistics

Table 6.1 provides an overview of the amount of users and programs which are extracted from the DB. Additionally, the amount of programs with and without tags and extensions are listed. A total number of 35,794 programs were uploaded during the observation period. 23,605 of these uploaded programs were programs with tags, which is equal to 65.95%. Only 339 uploaded programs had extensions and only 308 programs had combination of tags and extensions, which equals about 0.86% of the total uploaded programs. The user base can be considered as relatively high with 38,970 users. However, it must be considered, that not all users are regularly active.

Users	38,970
Total programs	35,794
Programs with tags	23,605
Programs with extensions	339
Programs with tags and extensions	308

Table 6.1: General Statistics

6.1.2 Tag and Extension Distribution

As mentioned in the previous section, 23,605 programs (Table 6.1) were uploaded with tags. Figure 6.1 shows how the tags are distributed over the uploaded programs. Obviously, the tag "Game" is the most popular and the most used tag. This is probably because the target group are young people. On the other site, as mentioned in the section above, extensions have barley been used (e.g. Drone was not used at all). The most used extension is *Phiro* which is used 282 times in programs, which could be due to the fact that the majority of the users use the app only on smartphone or tablet and do not buy the hardware which is needed for the use of extensions.

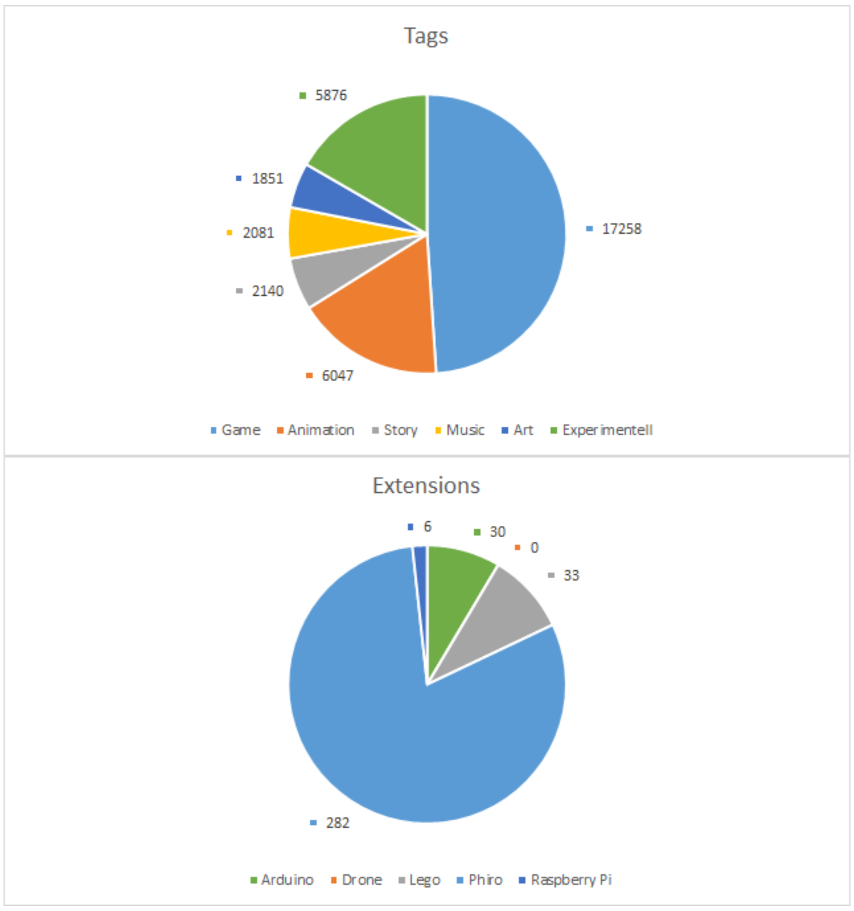


Figure 6.1: Distribution of Tags and Extensions

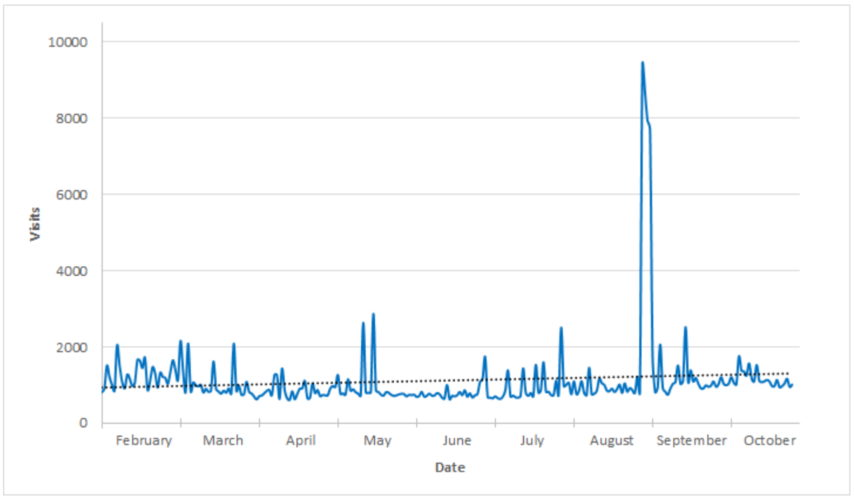


Figure 6.2: Activity on the Community Site

6.2 Evaluation of Results

6.2.1 Results of the Tagging System

In this evaluation the activity is defined as a visit of the user. A visit is the period where the user interacts actively with the community site. The visits statistics are provided from Google Analytics and the clicks and download statistics coming directly from the DB. Figure 6.2 shows the visits on the y-axis of the community site and the date on the x-axis. The inserted trend-line is represented as a dot line and shows the long-term movement, which slightly increases during the evaluation period. Generally, the course of th line is constant. However, there are some peaks, especially during the end of August and the beginning of September.

During the evaluation period, a total of 34, 588 programs were uploaded, from which 23, 070 programs had tags assigned. Figure 6.3 represents the number of downloads and uploads of programs with a particular tag. The tag “Game” is uploaded 16, 868 and downloaded 135, 142 times. All other tags are far behind, for example the second most downloaded tag is “Experimentell” with 49, 696 downloads and the second most uploaded tag is “Animation” with 5, 886 uploads.

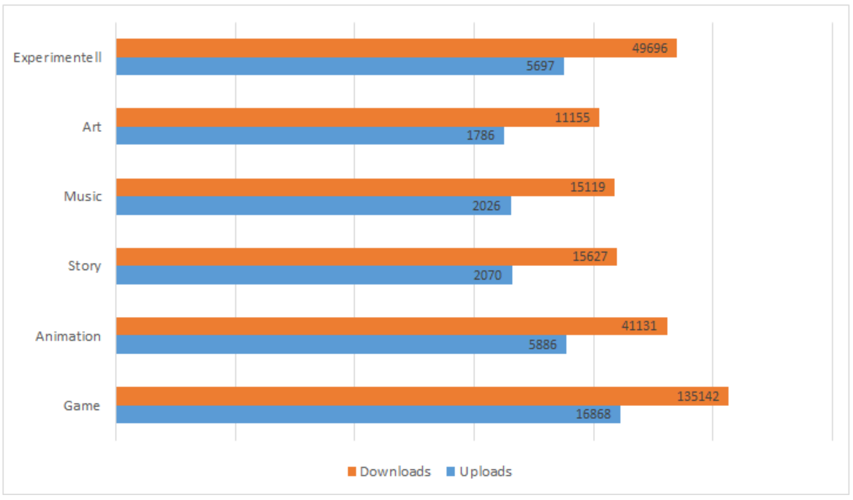


Figure 6.3: Download Uploads

Additionally, the click-through rate is calculated. Table 6.2 shows in detail how often the tags are clicked on in relation to the visits where the program detail site is included. The click-through rate of the tags is overall low and the highest rate is provided by the tag "Game" with 6.66%. On the other side, the lowest rate is provided by the tag "Art" and has just 0.08%. In sum the total click-through rate equals 9.20%. The evaluation and analysis of the extensions can be neglected due of their small usage.

Measure	Game	Animation	Story	Music	Art	Experimentell
Total visits of the community site	302,211					
Visits including the detail site	58,218					
Number of clicks	3,880	691	173	101	46	463
Click-through rate (in %)	6.66	1.19	0.30	0.17	0.08	0.80

Table 6.2: Click Statistics of Tags

6.2.2 Results of the Recommendation System

For the A/B test, Germany and Russia were chosen because of their similar user base. An example of user flow on the community site can be found in the previous Chapter 5 in Section 5.2.3.

All further analyzes take all visits into account, where the program detail site has been viewed. Table 6.3 summarizes all data of version A (Russia) and version B (Germany). The total visits of the community site and also the visits which include the detail site are pretty much equal. A had a visit rate for the detail site of 5.98%, whereas B had a visit rate for the detail site of 7.35%. The download rate of A is slightly higher than the download rate of B. Recommended programs had 700 clicks and led to 217 downloads in B, which equals 31.00%. The total download rate achieved in both versions is over 50.00%.

Measure	Version A	Version B
Total visits of the community site	28, 155	27, 694
Visits including the detail site	1, 684	2, 035
Total downloads	18, 694	14, 313
Downloads via RS	-	217
Clicks on recommended programs	-	700
Total download rate (in %)	66.40	51.68
Click-through rate (in %)	-	34.40

Table 6.3: Statistics of Versions A and B

Figure 6.4 shows the downloading behavior of the two countries. The download count on the y-axis is compared with the date on the x-axis. Russia without the RS is always higher than Germany with the RS. However, the overall download behavior is much the same. The low download count of Russia in the month April is also reflected in the line of Germany.

6.3 Discussion and Conclusion

6.3.1 Tagging System

The results of the tagging system should answer the first research question which was formulated in the beginning of this thesis:

> *"Can the activity on the community site be raised by adding a tagging system, which helps users to sort and find programs?"*

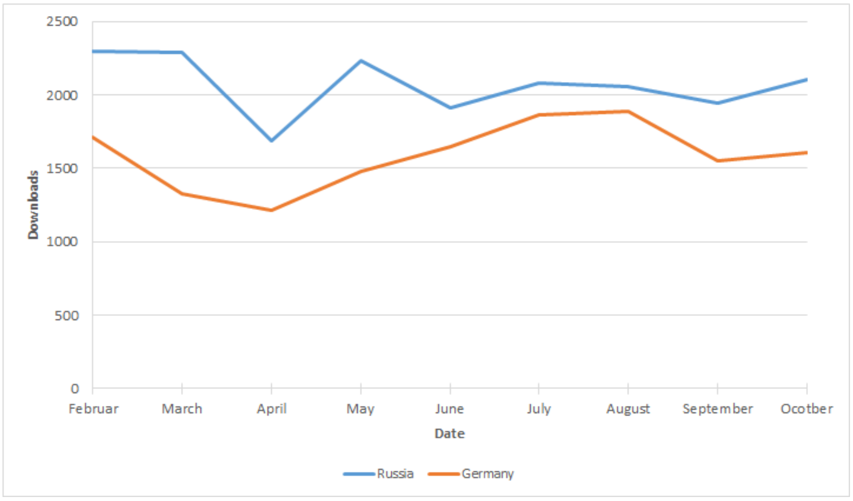


Figure 6.4: Downloads of Germany and Russia

In Figure 6.2, the trend-line shows a slight increase in the activity. This is not due to the implementation of the tagging system but rather of the high increase at the beginning of September. This increase is probably due the beginning of school. Without this peak, the trend-line would be more stable.

Unfortunately, the usage of extensions is very low and therefore not worth analyzing in more detail. The usage of the tags on the program detail site lies only by 9.20%. This is because only 19.26% of all users were viewing the program detail site.

On the other side, from 34, 588 uploaded programs, 23, 070 programs were provided with tags, which equals 66.70%. The tags are maybe not much used directly on the site, but they are used frequently in the uploading process. As can be seen in Figure 6.3, which shows the upload and download count of programs with tags, the most popular tag is “Game”. This is probably due to the fact that the app is built to teach programming easy and playful. With this available data which is achieved by the tagging system, the research question cannot be answered in full, but there is some potential in the tagging system and especially in the tag “Game” itself.

6.3.2 Recommendation System

The results of the RS should answer the second research question which was:

> *"Can the download numbers on the community site be increased by implementing a RS based on tags?"*

As summarized in Table 6.3, the data shows no increase in the download behavior of B where the RS was implemented. During the evaluation period, only 217 downloads were made over this system. However, it is important to note that only 2, 035 users have accessed the program detail site where the RS was shown. A had 1, 684 visits on the detail site which is less than B, but had a higher download count.

On probable conclusion could be that users find the programs they desire easier with an implemented RS and therefore the download rate is lower in Germany than in Russia. However, the difference of 4, 381 downloads between Germany and Russia cannot be compensated by the 217 downloads via the RS. Furthermore, Figure 6.4 shows that the download behavior is overall the same. During the observation period, the download count of A was always higher than the download count of B with the RS.

Furthermore, it is possible that the position of the RS on the site was not beneficial, because most users are browsing the community site with a smartphone or tablet. This would mean that the users must scroll/swipe to the bottom of the site to see the RS. With only 700 clicks on recommended programs during 2, 035 visits this is highly probable. Additionally, not all users are going to the program detail site. Based on the achieved data by the implemented recommendation system, the research question can not be affirmed.

7 Summary and Future Work

7.0.1 Summary

This thesis is constructed in two parts. The first part is theoretical and includes a short introduction where the research questions are defined and also an overview of techniques and approaches of RS, annotations and tags are provided. The second part is practical and deals with the implementation of a tagging system and a RS which is based on the tagging system. This part also includes the results as well as their evaluation and discussion. Furthermore, a summary and ideas for further work is provided.

For the decision which system should be used for answering the research questions, some considerations were made, such as design changes or hosting more events for the users. The best decision was to implement a tagging system and a RS. The tagging system should raise the activity on the community site of Catrobat and the RS should increase the downloads of programs. For the measurement of the activity, the visits on the community site, which were provided from Google Analytics, were used. The results of the RS were evaluated through an A/B test, where A (Russia) was not provided with a RS. The data for both measurements and tests were tracked and logged on the community site.

The results from the tagging system and the RS were evaluated and used to try to answer the research questions. The first question where the data of the tagging system was used, cannot be fully answered because the data is not sufficient enough to make a reliable statement. Nevertheless, the upload of programs with tags shows that the tagging system had potential for further improvement and testing.

On the other side, the data of the RS implies that this system was barely used. There was no increase in the download count of B with the RS.

Furthermore, the data shows that the download behavior is much the same of both versions, regardless that B has less downloads than A. The reason for not using the RS probably lies in the position of the recommendation area, but to be sure a closer examination would be needed. Furthermore, the low visit rate of the program detail site can be a reason for the bad performance.

7.0.2 Future Work

Both systems which are implemented in this thesis have room for improvement. The tagging system can be extended with new tags or sub-tags can be provided for the tag “Game” such as “Puzzle” or “Role Playing Game (RPG)”. In order to achieve more activity on the community site, the tagging system could be advertised more with an own banner. Another way would be to advertise the app on social media channels in order to gain more data for this system. Further, it is possible to ask an *influencer*, who are people which have a strong presence and high reputation on social media networks, to advertise the app with its features. Additionally, the tagging system could be upgraded to show a tag cloud on the community site with the most used, downloaded or uploaded tags. The tag cloud can also be personalized, to highlight tags in which the user is may be interested. Thus, it can be made possible that users can not only choose from given tags but create their own. For the use of user created tags a bad word filter would be necessary and a dictionary should be implemented for multi-language support.

To test the use of the RS in more detail, the position of the recommended programs should be changed, e.g. it could be placed on the index page or a pop-up window can be made which shows the recommended programs. Additionally, another recommendation approach can be used. A hybrid would be a great choice here, because the user would get programs recommended based on the tagging or content-based approaches. When the system has enough information about the user, it can switch to a collaborative approach. However, the problem is to define the threshold when to switch between the systems.

List of Abbreviations

Ajax	Asynchronous JavaScript and XML
API	Application Programming Interface
BPC	BellKor's Pragmatic Chaos
BSc	Bachelor of Science in Engineering
CF	Collaborative Filtering
DB	Database
DQL	Doctrine Query Language
FOSS	Free and Open Source Software
GUI	Graphical User Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
IMDb	Internet Movie Database
IST	Institute of Software Technology
kNN	k-nearest-neighbors
MSc	Master of Science
NFC	Near Field Communication
OS	Operating System
PCo	Pocket Code
RMSE	Root Mean Square Error
RPG	Role Playing Game
RS	Recommender System
RSS	Rich Site Summary
SQL	Structured Query Language
TUGraz	Technical University of Graz
WWW	World Wide Web
XML	Extensible Markup Language

Appendix

Implementation Code

The complete code of the **Tagging and Recommendation System - Implementation** can be found on **Github** - https://github.com/Catrobat/Catroweb-Symfony

Web API Reference

HTTP Request	Method and Parameters	Description
/api/tags/getTags.json	taggingAction(Request $request)	Gives all supported tags in the supported languages back.
/api/upload/upload.json	uploadAction(Request $request)	Programs are uploaded via the API. The file from the request is further processed in the program manager.
/api/projects/search/tagPrograms.json	tagSearchProgramsAction(Request $request)	Returns a list object with programs which has the given tag. The list is in descending order by upload time.
/api/projects/search/extensionPrograms.json	extensionSearchProgramsAction(Request $request)	Returns a list object with programs which has the given extension. The list is in descending order by upload time.
/api/projects/recsys.json	listRecsysProgramAction(Request $request)	Returns a list object with similar programs for the target program.

Table .2: API Functions - Server Side

Bibliography

Adomavicius, G. and A. Tuzhilin (2005). "Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions." In: *IEEE Transactions on Knowledge and Data Engineering* 17.6, pp. 734–749. ISSN: 1041-4347. DOI: 10.1109/TKDE.2005.99 (cit. on p. 14).

Apple (2018). *iOS - iOS10 - Apple (AT)*. URL: http://www.apple.com/at/ios/ios-10/ (visited on 06/16/2018) (cit. on p. 8).

Arndt, Richard et al. (2007). "COMM: Designing a Well-Founded Multimedia Ontology for the Web." In: *The Semantic Web: 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, ISWC 2007 + ASWC 2007, Busan, Korea, November 11-15, 2007. Proceedings*. Ed. by Karl Aberer et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 30–43. ISBN: 978-3-540-76298-0. DOI: 10.1007/978-3-540-76298-0_3. URL: http://dx.doi.org/10.1007/978-3-540-76298-0_3 (cit. on p. 30).

Balabanović, Marko and Yoav Shoham (1997). "Fab: Content-based, Collaborative Recommendation." In: *Commun. ACM* 40.3, pp. 66–72. ISSN: 0001-0782. DOI: 10.1145/245108.245124. URL: http://doi.acm.org/10.1145/245108.245124 (cit. on p. 14).

Belkin, Nicholas J and W Bruce Croft (1992). "Information filtering and information retrieval: Two sides of the same coin?" In: *Communications of the ACM* 35.12, pp. 29–38 (cit. on p. 1).

Bell, Robert M. and Yehuda Koren (2007). "Lessons from the Netflix Prize Challenge." In: *SIGKDD Explor. Newsl.* 9.2, pp. 75–79. ISSN: 1931-0145. DOI: 10.1145/1345448.1345465. URL: http://doi.acm.org/10.1145/1345448.1345465 (cit. on p. 28).

Bennett, J. and S. Lanning (2007). "The Netflix Prize." In: *Proceedings of the KDD Cup Workshop 2007*. New York: ACM, pp. 3–6 (cit. on p. 28).

Blogger (2018). *Blogger.com - Create a unique and beautiful blog. It's easy and free.* URL: https://www.blogger.com/ (visited on 06/15/2018) (cit. on p. 33).

Bobadilla, J. et al. (2013). “Recommender systems survey.” In: *Knowledge-Based Systems* 46, pp. 109–132. ISSN: 0950-7051. DOI: http://doi.org/10.1016/j.knosys.2013.03.012. URL: http://www.sciencedirect.com/science/article/pii/S0950705113001044 (cit. on p. 16).

Böhmer, Matthias, Lyubomir Ganev, and Antonio Krüger (2013). “AppFunnel: A Framework for Usage-centric Evaluation of Recommender Systems That Suggest Mobile Applications.” In: *Proceedings of the 2013 International Conference on Intelligent User Interfaces*. IUI ’13. Santa Monica, California, USA: ACM, pp. 267–276. ISBN: 978-1-4503-1965-2. DOI: 10.1145/2449396.2449431. URL: http://doi.acm.org/10.1145/2449396.2449431 (cit. on p. 1).

Breese, John S., David Heckerman, and Carl Kadie (1998). “Empirical Analysis of Predictive Algorithms for Collaborative Filtering.” In: *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*. UAI’98. Madison, Wisconsin: Morgan Kaufmann Publishers Inc., pp. 43–52. ISBN: 1-55860-555-X. URL: http://dl.acm.org/citation.cfm?id=2074094.2074100 (cit. on p. 14).

Brin, Sergey and Lawrence Page (2012). “Reprint of: The anatomy of a large-scale hypertextual web search engine.” In: *Computer Networks* 56.18. The WEB we live in, pp. 3825–3833. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2012.10.007. URL: http://www.sciencedirect.com/science/article/pii/S1389128612003611 (cit. on p. 39).

Builder, Query (2018). *15. The QueryBuilder - Doctrine 2 ORM 2 documentation*. URL: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/query-builder.html (visited on 06/11/2018) (cit. on p. 49).

Burke, Robin (2002). “Hybrid Recommender Systems: Survey and Experiments.” In: *User Modeling and User-Adapted Interaction* 12.4, pp. 331–370. ISSN: 1573-1391. DOI: 10.1023/A:1021240730564. URL: http://dx.doi.org/10.1023/A:1021240730564 (cit. on pp. 20, 21).

Burke, Robin (2005). “Hybrid Systems for Personalized Recommendations.” In: *Intelligent Techniques for Web Personalization: IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers*. Ed. by Bamshad Mobasher and Sarabjot Singh Anand. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 133–152. ISBN: 978-3-540-31655-8. DOI:

10.1007/11577935_7. URL: http://dx.doi.org/10.1007/11577935_7 (cit. on p. 27).

Burke, Robin (2007). "Hybrid Web Recommender Systems." In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 377–408. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_12. URL: http://dx.doi.org/10.1007/978-3-540-72079-9_12 (cit. on pp. 20–22).

Burke, Robin, Alexander Felfernig, and Mehmet H Göker (2011). "Recommender systems: An overview." In: *Ai Magazine* 32.3, pp. 13–18 (cit. on pp. 13, 24).

Burke, Robin and Maryam Ramezani (2011). "Matching Recommendation Technologies and Domains." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 367–386. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_11. URL: http://dx.doi.org/10.1007/978-0-387-85820-3_11 (cit. on pp. 25, 26).

Capocci, Andrea and Guido Caldarelli (2008). "Folksonomies and clustering in the collaborative system CiteULike." In: *Journal of Physics A: Mathematical and Theoretical* 41.22, p. 224016. URL: http://stacks.iop.org/1751-8121/41/i=22/a=224016 (cit. on p. 40).

Catrobat (2018a). *Catrobat*. URL: http://developer.catrobat.org/ (visited on 06/13/2018) (cit. on p. 7).

Catrobat (2018b). *Catrobat Website*. URL: https://www.catrobat.org/ (visited on 06/15/2018) (cit. on p. 5).

Catrobat (2018c). *Galaxy Game Jam*. URL: http://www.galaxygamejam.com/ (visited on 06/14/2018) (cit. on p. 44).

Catrobat (2018d). *Pocket Code: Programmiere Apps*. URL: https://play.google.com/store/apps/details?id=org.catrobat.catroid (visited on 06/15/2018) (cit. on p. 8).

Chirita, Paul - Alexandru et al. (2007). "P-TAG: Large Scale Automatic Generation of Personalized Annotation Tags for the Web." In: *Proceedings of the 16th International Conference on World Wide Web*. WWW '07. Banff, Alberta, Canada: ACM, pp. 845–854. ISBN: 978-1-59593-654-7. DOI: 10.1145/1242572.1242686. URL: http://doi.acm.org/10.1145/1242572.1242686 (cit. on p. 29).

CiteULike (2018). *CiteULike: Everyone's library*. URL: http://www.citeulike.org/ (visited on 06/16/2018) (cit. on p. 35).

Costa-Montenegro, Enrique, Ana B. Barragáns-Martínez, and Marta Rey-López (2012). "Which App? A recommender system of applications in markets: Implementation of the service for monitoring users' interaction." In: *Expert Systems with Applications* 39.10, pp. 9367–9375. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2012.02.131. URL: http://www.sciencedirect.com/science/article/pii/S0957417412003946 (cit. on p. 2).

Cremonesi, Paolo, Yehuda Koren, and Roberto Turrin (2010). "Performance of Recommender Algorithms on Top-n Recommendation Tasks." In: *Proceedings of the Fourth ACM Conference on Recommender Systems*. RecSys '10. Barcelona, Spain: ACM, pp. 39–46. ISBN: 978-1-60558-906-0. DOI: 10.1145/1864708.1864721. URL: http://doi.acm.org/10.1145/1864708.1864721 (cit. on p. 22).

Crowdin (2018). *Localization Management Platform: collaborative internationalization and easy to use translation system - Crowdin*. URL: https://crowdin.com/ (visited on 06/15/2018) (cit. on p. 55).

Dareddy, Manoj Reddy (2017). "Recommender Systems: Research Direction." In: *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*. WSDM '17. Cambridge, United Kingdom: ACM, pp. 831–831. ISBN: 978-1-4503-4675-7. DOI: 10.1145/3018661.3022748. URL: http://doi.acm.org/10.1145/3018661.3022748 (cit. on p. 11).

Davidsson, Christoffer (2010). *Mobile application recommender system* (cit. on p. 1).

Desrosiers, Christian and George Karypis (2011). "A Comprehensive Survey of Neighborhood-based Recommendation Methods." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 107–144. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_4. URL: http://dx.doi.org/10.1007/978-0-387-85820-3_4 (cit. on p. 15).

Doctrine (2018). *Home - Doctrine Project*. URL: http://www.doctrine-project.org/ (visited on 06/15/2018) (cit. on p. 47).

Dwyer, Catherine, Starr Hiltz, and Katia Passerini (2007). "Trust and privacy concern within social networking sites: A comparison of Facebook and MySpace." In: *AMCIS 2007 proceedings*, pp. 1–3 (cit. on p. 34).

Ebersbach, Anja, Markus Glaser, and Richard Heigl (2016). *Social Web*. UTB GmbH, pp. 24–33. ISBN: 9783825239336. URL: https://books.google.at/books?id=lSn3DAAAQBAJ (cit. on p. 31).

Ekstrand, Michael D., John T. Riedl, and Joseph A. Konstan (2011). “Collaborative Filtering Recommender Systems.” In: *Foundations and Trends® in Human–Computer Interaction* 4.2, pp. 81–173. ISSN: 1551-3955. DOI: 10.1561/1100000009. URL: http://dx.doi.org/10.1561/1100000009 (cit. on p. 11).

Facebook (2018). *Facebook - Log In or Sign UP*. URL: https://www.facebook.com/ (visited on 06/15/2018) (cit. on pp. 9, 34).

Felfernig, Alexander and Robin Burke (2008). “Constraint-based Recommender Systems: Technologies and Research Issues.” In: *Proceedings of the 10th International Conference on Electronic Commerce*. ICEC ’08. Innsbruck, Austria: ACM, 3:1–3:10. ISBN: 978-1-60558-075-3. DOI: 10.1145/1409540.1409544. URL: http://doi.acm.org/10.1145/1409540.1409544 (cit. on pp. 24–26).

Felfernig, Alexander, Gerhard Friedrich, et al. (2011). “Developing Constraint-based Recommenders.” In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 187–215. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_6. URL: http://dx.doi.org/10.1007/978-0-387-85820-3_6 (cit. on pp. 18, 19).

Felfernig, Alexander, Michael Jeran, et al. (2014). “Basic Approaches in Recommendation Systems.” In: *Recommendation Systems in Software Engineering*. Ed. by Martin P. Robillard, Robert J. Maalej Walidand Walker, and Thomas Zimmermann. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15–37. ISBN: 978-3-642-45135-5. DOI: 10.1007/978-3-642-45135-5_2. URL: http://dx.doi.org/10.1007/978-3-642-45135-5_2 (cit. on p. 12).

Flickr (2018). *Find your inspiration. — Flickr*. URL: https://www.flickr.com/ (visited on 06/14/2018) (cit. on p. 35).

Ghazanfar, Mustansar Ali and Adam Prugel-Bennett (2010). “A scalable, accurate hybrid recommender system.” In: *Knowledge Discovery and Data Mining, 2010. WKDD’10. Third International Conference on*. IEEE, pp. 94–98 (cit. on p. 2).

Goldberg, David et al. (1992). “Using Collaborative Filtering to Weave an Information Tapestry.” In: *Commun. ACM* 35.12, pp. 61–70. ISSN: 0001-0782. DOI: 10.1145/138859.138867. URL: http://doi.acm.org/10.1145/138859.138867 (cit. on p. 13).

Golder, Scott A. and Bernardo A. Huberman (2006). “Usage patterns of collaborative tagging systems.” In: *Journal of Information Science* 32.2, pp. 198–208. DOI: 10.1177/0165551506062337. eprint: https://doi.org/10.1177/0165551506062337. URL: https://doi.org/10.1177/0165551506062337 (cit. on p. 35).

Gomez-Uribe, Carlos A. and Neil Hunt (2015). "The Netflix Recommender System: Algorithms, Business Value, and Innovation." In: *ACM Trans. Manage. Inf. Syst.* 6.4, 13:1–13:19. ISSN: 2158-656X. DOI: 10.1145/2843948. URL: http://doi.acm.org/10.1145/2843948 (cit. on p. 28).

Gong, S., H. Ye, and X. Shi (2008). "A Collaborative Recommender Combining Item Rating Similarity and Item Attribute Similarity." In: *2008 International Seminar on Business and Information Management*. Vol. 2, pp. 58–60. DOI: 10.1109/ISBIM.2008.190 (cit. on p. 13).

Google (2018a). *Android*. URL: https://www.android.com/ (visited on 06/15/2018) (cit. on p. 8).

Google (2018b). *Goggle+*. URL: https://plus.google.com/ (visited on 06/15/2018) (cit. on p. 9).

Google (2018c). *Google Analytics Solutions - Marketing Analytics & Measurement*. URL: https://www.google.com/analytics/ (visited on 06/08/2018) (cit. on p. 46).

Google (2018d). *Google Play*. URL: https://play.google.com/store (visited on 06/18/2018) (cit. on p. 8).

Gruber, Tom (2008). "Collective knowledge systems: Where the Social Web meets the Semantic Web." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 6.1. Semantic Web and Web 2.0, pp. 4–13. ISSN: 1570-8268. DOI: http://dx.doi.org/10.1016/j.websem.2007.11.011. URL: http://www.sciencedirect.com/science/article/pii/S1570826807000583 (cit. on p. 31).

Hallinan, Blake and Ted Striphas (2016). "Recommended for you: The Netflix Prize and the production of algorithmic culture." In: *New Media & Society* 18.1, pp. 117–137. DOI: 10.1177/1461444814538646. eprint: http://dx.doi.org/10.1177/1461444814538646. URL: http://dx.doi.org/10.1177/1461444814538646 (cit. on p. 28).

Hanani, Uri, Bracha Shapira, and Peretz Shoval (2001). "Information Filtering: Overview of Issues, Research and Systems." In: *User Modeling and User-Adapted Interaction* 11.3, pp. 203–259. ISSN: 0924-1868. DOI: 10.1023/A:1011196000674. URL: http://dx.doi.org/10.1023/A:1011196000674 (cit. on p. 12).

Herlocker, Jonathan L. et al. (2004). “Evaluating Collaborative Filtering Recommender Systems.” In: *ACM Trans. Inf. Syst.* 22.1, pp. 5–53. ISSN: 1046-8188. DOI: 10.1145/963770.963772. URL: http://doi.acm.org/10.1145/963770.963772 (cit. on p. 23).

IMDb (2018). *IMDb - Movies, TV and Celebrities - IMDb*. URL: http://www.imdb.com/ (visited on 06/17/2018) (cit. on p. 40).

Jaindl, Stefan (2016). “Social Media Software Integration for the Symfony Web Framework and Android and iOS Versions of the Catrobat Project.” MA thesis. Graz University of Technology, pp. 115–149 (cit. on p. 49).

Jannach, Dietmar et al. (2010). *Recommender systems: an introduction*. Cambridge University Press (cit. on pp. 16, 18, 19, 40).

Jäschke, Robert et al. (2007). “Tag Recommendations in Folksonomies.” In: *Knowledge Discovery in Databases: PKDD 2007: 11th European Conference on Principles and Practice of Knowledge Discovery in Databases, Warsaw, Poland, September 17-21, 2007. Proceedings*. Ed. by Joost N. Kok et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 506–514. ISBN: 978-3-540-74976-9. DOI: 10.1007/978-3-540-74976-9_52. URL: https://doi.org/10.1007/978-3-540-74976-9_52 (cit. on p. 39).

Kim, Hak Lae et al. (2008). “The state of the art in tag ontologies: a semantic model for tagging and folksonomies.” In: *International Conference on Dublin Core and Metadata Applications*, pp. 128–137 (cit. on p. 40).

Kiryakov, Atanas et al. (2004). “Semantic annotation, indexing, and retrieval.” In: *Web Semantics: Science, Services and Agents on the World Wide Web* 2.1, pp. 49–79. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2004.07.005. URL: http://www.sciencedirect.com/science/article/pii/S1570826804000162 (cit. on p. 29).

Kohavi, Ron and Roger Longbotham (2015). “Online controlled experiments and A/B tests.” In: *Encyclopedia of machine learning and data mining*, pp. 1–11 (cit. on p. 46).

Lam, Chuck P. (2005). “Collaborative Filtering Using Associative Neural Memory.” In: *Intelligent Techniques for Web Personalization: IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers*. Ed. by Bamshad Mobasher and Sarabjot Singh Anand. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 153–168. ISBN: 978-3-540-31655-8. DOI: 10.1007/11577935_8. URL: http://dx.doi.org/10.1007/11577935_8 (cit. on p. 15).

Language, Doctrine Query (2018). *14. Doctrine Query Language - Doctrine 2 ORM 2 documentation*. URL: http://docs.doctrine-project.org/projects/doctrine-orm/en/latest/reference/dql-doctrine-query-language.html (visited on 06/19/2018) (cit. on p. 49).

Last.fm (2018). *Last.fm - Listen to free music and watch videos with the largest music catalogue online*. URL: https://www.last.fm/ (visited on 06/15/2018) (cit. on p. 35).

Le, Hoang Lam, Quoc Cuong Nguyen, and Minh Tri Nguyen (2017). "An improvement on recommender systems by exploring more relationships." In: *International Journal of Advanced Computer Research* 7.29, pp. 42–51 (cit. on pp. 20–22).

Liu, Zhiyuan, Chuan Shi, and Maosong Sun (2010). "FolkDiffusion: A Graph-Based Tag Suggestion Method for Folksonomies." In: *Information Retrieval Technology: 6th Asia Information Retrieval Societies Conference, AIRS 2010, Taipei, Taiwan, December 1-3, 2010. Proceedings*. Ed. by Pu-Jen Cheng et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 231–240. ISBN: 978-3-642-17187-1. DOI: 10.1007/978-3-642-17187-1_22. URL: https://doi.org/10.1007/978-3-642-17187-1_22 (cit. on p. 40).

Lorenzi, Fabiana and Francesco Ricci (2005). "Case-Based Recommender Systems: A Unifying View." In: *Intelligent Techniques for Web Personalization: IJCAI 2003 Workshop, ITWP 2003, Acapulco, Mexico, August 11, 2003, Revised Selected Papers*. Ed. by Bamshad Mobasher and Sarabjot Singh Anand. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 89–113. ISBN: 978-3-540-31655-8. DOI: 10.1007/11577935_5. URL: http://dx.doi.org/10.1007/11577935_5 (cit. on p. 19).

Lu, Jie et al. (2015). "Recommender system application developments: a survey." In: *Decision Support Systems* 74, pp. 12–32. ISSN: 0167-9236. DOI: http://doi.org/10.1016/j.dss.2015.03.008. URL: http://www.sciencedirect.com/science/article/pii/S0167923615000627 (cit. on p. 14).

Marlow, Cameron et al. (2006). "Position paper, tagging, taxonomy, flickr, article, toread." In: *In Collaborative Web Tagging Workshop at WWW'06*, pp. 31–40 (cit. on p. 36).

Mika, Peter (2007). "Ontologies are us: A unified model of social networks and semantics." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 5.1. Selected Papers from the International Semantic Web Conference, pp. 5–15. ISSN: 1570-8268. DOI: https://doi.org/10.

1016/j.websem.2006.11.002. URL: http://www.sciencedirect.com/science/article/pii/S1570826806000552 (cit. on p. 40).

Morita, Masahiro and Yoichi Shinoda (1994). "Information filtering based on user behavior analysis and best match text retrieval." In: *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., pp. 272–281 (cit. on p. 1).

Murugesan, San (2007). "Understanding Web 2.0." In: *IT Professional* 9.4, pp. 34–41. ISSN: 1520-9202. DOI: 10.1109/MITP.2007.78 (cit. on p. 33).

Myspace (2018). *Featured Content on Myspace*. URL: https://myspace.com/ (visited on 06/15/2018) (cit. on p. 34).

Nardi, Bonnie A., Diane J. Schiano, and Michelle Gumbrecht (2004). "Blogging As Social Activity, or, Would You Let 900 Million People Read Your Diary?" In: *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. CSCW '04. Chicago, Illinois, USA: ACM, pp. 222–231. ISBN: 1-58113-810-5. DOI: 10.1145/1031607.1031643. URL: http://doi.acm.org/10.1145/1031607.1031643 (cit. on p. 33).

Netflix (2018a). *Netflix Prize Home*. URL: http://www.netflixprize.com (visited on 06/14/2018) (cit. on p. 28).

Netflix (2018b). *Netflix - Watch TV Shows Online, Watch Movies Online*. URL: https://www.netflix.com/ (visited on 06/09/2018) (cit. on p. 28).

Nguyen, Hanh T. H. et al. (2017). "Personalized Deep Learning for Tag Recommendation." In: *Advances in Knowledge Discovery and Data Mining: 21st Pacific-Asia Conference, PAKDD 2017, Jeju, South Korea, May 23-26, 2017, Proceedings, Part I*. Ed. by Jinho Kim et al. Cham: Springer International Publishing, pp. 186–197. ISBN: 978-3-319-57454-7. DOI: 10.1007/978-3-319-57454-7_15. URL: https://doi.org/10.1007/978-3-319-57454-7_15 (cit. on p. 39).

O'Reilly, Tim (2007). *What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software*. MPRA Paper. University Library of Munich, Germany, pp. 17–18. URL: http://EconPapers.repec.org/RePEc:pra:mprapa:4578 (cit. on p. 31).

Ovadia, Steven (2012). "Syncing Bookmarks: An Overview of Current Options." In: *Behavioral & Social Sciences Librarian* 31.1, pp. 76–79. DOI: 10.1080/01639269.2012.657571. eprint: http://dx.doi.org/10.1080/01639269.2012.657571. URL: http://dx.doi.org/10.1080/01639269.2012.657571 (cit. on p. 35).

Park, Deuk Hee et al. (2012). "A literature review and classification of recommender systems research." In: *Expert Systems with Applications* 39.11, pp. 10059–10072. ISSN: 0957-4174. DOI: http://doi.org/10.1016/j.eswa.2012.02.038. URL: http://www.sciencedirect.com/science/article/pii/S0957417412002825 (cit. on p. 14).

Parker, Kevin and Joseph Chao (2007). "Wiki as a teaching tool." In: *Interdisciplinary Journal of e-learning and Learning Objects* 3.1, pp. 57–72 (cit. on p. 33).

Pazzani, Michael J. and Daniel Billsus (2007). "Content-Based Recommendation Systems." In: *The Adaptive Web: Methods and Strategies of Web Personalization*. Ed. by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 325–341. ISBN: 978-3-540-72079-9. DOI: 10.1007/978-3-540-72079-9_10. URL: http://dx.doi.org/10.1007/978-3-540-72079-9_10 (cit. on p. 17).

Pinboard (2018). *Pinboard: social bookmarking for introverts*. URL: https://pinboard.in/ (visited on 06/15/2018) (cit. on p. 35).

Resnick, Paul and Hal R. Varian (1997). "Recommender Systems." In: *Commun. ACM* 40.3, pp. 56–58. ISSN: 0001-0782. DOI: 10.1145/245108.245121. URL: http://doi.acm.org/10.1145/245108.245121 (cit. on p. 13).

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). "Introduction to Recommender Systems Handbook." In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 1–35. ISBN: 978-0-387-85820-3. DOI: 10.1007/978-0-387-85820-3_1. URL: http://dx.doi.org/10.1007/978-0-387-85820-3_1 (cit. on p. 23).

Richter, Alexander and Michael Koch (2007). *Social software: Status quo und Zukunft*. Fak. für Informatik, Univ. der Bundeswehr München (cit. on pp. 30, 32, 34).

Schafer, J. Ben, Joseph Konstan, and John Riedl (1999). "Recommender Systems in e-Commerce." In: *Proceedings of the 1st ACM Conference on Electronic Commerce*. EC '99. Denver, Colorado, USA: ACM, pp. 158–166. ISBN: 1-58113-176-3. DOI: 10.1145/336992.337035. URL: http://doi.acm.org/10.1145/336992.337035 (cit. on p. 22).

Schmidt, Jan (2006). "Social Software: Onlinegestütztes Informations-, Identitäts-und Beziehungsmanagement." In: *Forschungsjournal Neue Soziale Bewegungen* 19.2, pp. 37–47 (cit. on p. 31).

Schnedlitz, Adrian (2016). "TimePunch - An Online Timetracking Tool for Education." MA thesis. Graz University of Technology, p. 28 (cit. on p. 7).

Scratch (2018a). *Project Tags - Scratch Wiki*. URL: https://wiki.scratch.mit.edu/wiki/Project_Tags (visited on 06/19/2018) (cit. on p. 45).

Scratch (2018b). *Scratch - Imagine, Program, Share*. URL: https://scratch.mit.edu/ (visited on 06/15/2018) (cit. on p. 5).

Sharma, Lalita and Anju Gera (2013). "A survey of recommendation system: Research challenges." In: *International Journal of Engineering Trends and Technology (IJETT)* 4.5, pp. 1989–1992 (cit. on p. 19).

Shepitsen, Andriy et al. (2008). "Personalized Recommendation in Social Tagging Systems Using Hierarchical Clustering." In: *Proceedings of the 2008 ACM Conference on Recommender Systems*. RecSys '08. Lausanne, Switzerland: ACM, pp. 259–266. ISBN: 978-1-60558-093-7. DOI: 10.1145/1454008.1454048. URL: http://doi.acm.org/10.1145/1454008.1454048 (cit. on p. 40).

Shu, Zhaoxin, Li Yu, and Xiaoping Yang (2010). "Personalized Tag Recommendation Based on User Preference and Content." In: *Advanced Data Mining and Applications: 6th International Conference, ADMA 2010, Chongqing, China, November 19-21, 2010, Proceedings, Part II*. Ed. by Longbing Cao, Jiang Zhong, and Yong Feng. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 348–355. ISBN: 978-3-642-17313-4. DOI: 10.1007/978-3-642-17313-4_34. URL: https://doi.org/10.1007/978-3-642-17313-4_34 (cit. on p. 39).

Slany, Wolfgang (2012). "A mobile visual programming system for Android smartphones and tablets." In: *2012 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 265–266. DOI: 10.1109/VLHCC.2012.6344546 (cit. on p. 5).

Slany, Wolfgang (2014). "Pocket Code: A Scratch-like Integrated Development Environment for Your Phone." In: *Proceedings of the Companion Publication of the 2014 ACM SIGPLAN Conference on Systems, Programming, and Applications: Software for Humanity*. SPLASH '14. Portland, Oregon, USA: ACM, pp. 35–36. ISBN: 978-1-4503-3208-8. DOI: 10.1145/2660252.2664662. URL: http://doi.acm.org/10.1145/2660252.2664662 (cit. on p. 5).

Song, Yang, Lu Zhang, and C. Lee Giles (2011). "Automatic Tag Recommendation Algorithms for Social Recommender Systems." In: *ACM Trans.*

Web 5.1, 4:1–4:31. ISSN: 1559-1131. DOI: 10.1145/1921591.1921595. URL: http://doi.acm.org/10.1145/1921591.1921595 (cit. on p. 39).

Specia, Lucia and Enrico Motta (2007). "Integrating Folksonomies with the Semantic Web." In: *The Semantic Web: Research and Applications: 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007. Proceedings*. Ed. by Enrico Franconi, Michael Kifer, and Wolfgang May. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 624–639. ISBN: 978-3-540-72667-8. DOI: 10.1007/978-3-540-72667-8_44. URL: https://doi.org/10.1007/978-3-540-72667-8_44 (cit. on p. 36).

Su, Xiaoyuan and Taghi M. Khoshgoftaar (2009). "A Survey of Collaborative Filtering Techniques." In: *Adv. in Artif. Intell.* 2009, pp. 1–19. ISSN: 1687-7470. DOI: 10.1155/2009/421425. URL: http://dx.doi.org/10.1155/2009/421425 (cit. on p. 15).

Symfony (2018). *Symfony, High Performance PHP Framework for Web Development*. URL: https://symfony.com/ (visited on 06/15/2018) (cit. on p. 9).

Szomszor, Martin et al. (2007). "Folksonomies, the Semantic Web, and Movie Recommendation." In: *Bridging the Gap between Semantic Web and Web 2.0 (SemNet 2007)*, pp. 71–84. URL: http://www.kde.cs.uni-kassel.de/ws/eswc2007/proc/Folksonomies.pdf (cit. on pp. 40, 41).

Tso-Sutter, Karen H. L., Leandro Balby Marinho, and Lars Schmidt-Thieme (2008). "Tag-aware Recommender Systems by Fusion of Collaborative Filtering Algorithms." In: *Proceedings of the 2008 ACM Symposium on Applied Computing*. SAC '08. Fortaleza, Ceara, Brazil: ACM, pp. 1995–1999. ISBN: 978-1-59593-753-7. DOI: 10.1145/1363686.1364171. URL: http://doi.acm.org/10.1145/1363686.1364171 (cit. on pp. 41, 42).

Twig (2018). *Home - Twig - The flexible, fast, and secure PHP template engine*. URL: https://twig.symfony.com/ (visited on 06/15/2018) (cit. on p. 55).

Twitter (2018). *Twitter. Alles, was gerade los ist.* URL: https://twitter.com/ (visited on 06/10/2018) (cit. on p. 9).

Type, Movable (2018). *Movable Type - Content Management System, Blog Software & Publishing Platform*. URL: https://www.movabletype.com/ (visited on 06/15/2018) (cit. on p. 33).

Uren, Victoria et al. (2006). "Semantic annotation for knowledge management: Requirements and a survey of the state of the art." In: *Web Semantics: Science, Services and Agents on the World Wide Web* 4.1, pp. 14–28. ISSN: 1570-8268. DOI: https://doi.org/10.1016/j.websem.2005.10.

002. URL: http://www.sciencedirect.com/science/article/pii/S1570826805000338 (cit. on p. 29).

Van Meteren, Robin and Maarten Van Someren (2000). "Using content-based filtering for recommendation." In: *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, pp. 47–56 (cit. on pp. 11, 17).

Wang, Jun, Arjen P. de Vries, and Marcel J. T. Reinders (2006). "Unifying User-based and Item-based Collaborative Filtering Approaches by Similarity Fusion." In: *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR '06. Seattle, Washington, USA: ACM, pp. 501–508. ISBN: 1-59593-369-7. DOI: 10.1145/1148170.1148257. URL: http://doi.acm.org/10.1145/1148170.1148257 (cit. on pp. 15, 16).

WikiIndex (2018). *WikiIndex*. URL: http://wikiindex.com (visited on 09/20/2017) (cit. on p. 34).

Wikipedia (2018). *Wikipedia*. URL: https://www.wikipedia.org/ (visited on 06/19/2018) (cit. on p. 34).

Winer, Dave (2018). *Scripting News*. URL: http://scripting.com/ (visited on 06/10/2018) (cit. on p. 33).

WordPress (2018). *WordPress.com: Create a free website or blog*. URL: https://wordpress.com/ (visited on 06/15/2018) (cit. on p. 33).

XING (2018). *XING - For a better working life*. URL: https://www.xing.com/ (visited on 06/15/2018) (cit. on p. 34).

Zhang, Zi-Ke, Tao Zhou, and Yi-Cheng Zhang (2010). "Personalized recommendation via integrated diffusion on user–item–tag tripartite graphs." In: *Physica A: Statistical Mechanics and its Applications* 389.1, pp. 179–186. ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2009.08.036. URL: http://www.sciencedirect.com/science/article/pii/S0378437109006839 (cit. on p. 40).

Zhang, Zi-Ke, Tao Zhou, and Yi-Cheng Zhang (2011). "Tag-Aware Recommender Systems: A State-of-the-Art Survey." In: *Journal of Computer Science and Technology* 26.5, pp. 767–777. ISSN: 1860-4749. DOI: 10.1007/s11390-011-0176-1. URL: https://doi.org/10.1007/s11390-011-0176-1 (cit. on p. 40).