Feedback Recommendation System Based on Structured Feedback Acquisition

E Hassan¹, N Sherief², W M Abdelmoez² and A A Saad¹
¹Department of Computer Engineering, College of Engineering and Technology, Arab Academy for Science, Technology and Maritime Transport - Egypt
²Department of Software Engineering, College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport - Egypt

Abstract. Users' feedback are increasingly becoming a vital factor during the software evolution process. Users' feedback are widely considered to be an indication of how satisfied are the system's users. It was found that large percentage of feedback represents problems reported by users who face problems in using the system. Users' feedback are usually communicated in an ad-hoc manner with no defined structure. The ad-hoc nature of the feedback makes it difficult to study and analyse feedback. Also, it usually happens that similar problems are reported by different users, so engineers spend wasted time and effort in duplicate issues. In this paper we propose a recommendation system which recommends to the user reported problems that are similar to his from previous feedback threads to reuse existing solutions. We based our work on a structured feedback system to ensure we get better results. Structured feedback ensures minimum level of useful and meaningful information that helps enhance the analysis results. Initial evaluation was conducted and the results are promising.

1. Introduction
Users' feedback are increasingly becoming a vital factor during software evolution process. Feedback are widely considered to be an indication of how satisfied are the system's users. It has been demonstrated that feedback may affect the rate of application download [1]. Users may also give suggestions and ideas for new features based on their experience with the software. User's feedback are the basic way to measure the users' satisfaction for the software system. Feedback are considered a rich source of information in terms of software evolution [2] [3]. In their work, [4] defined users' feedback in software systems as follows: “User feedback is a reaction of the user upon her experience in using a software service or application. It could be based on multi-modal communication, such as natural language text, images, emoticons, etc. Moreover, it contains meaningful information with the purpose of suggesting improvements, i.e. requesting new needs, reporting failures, and asking for modifications or clarifications”. In addition, it was found that large percentage of feedback represents problems reported in the maintenance phase by users who face problems in using the system [5]. These reported problems should be taken into consideration during the new version's implementation. Sometimes we can find that the reported problem was previously reported by other users, or even more, it may have been fixed in newer versions.
Although users' feedback are main source of information, especially in the maintenance phase, they are usually communicated in an ad-hoc manner. The ad-hoc nature of the feedback makes it difficult to study and analyse feedback. A structured feedback acquisition process was introduced by [6] in order to ease the communication and feedback acquisition between developers and end-users. Structuring the feedback makes it easy to extract valuable information that can be used to enhance the system. It was also noted that it results in more expressive feedback. Several studies were performed and the structured feedback system is accepted by both engineers and end-users.

In the maintenance phase, recommender systems can be used in several ways. For example a developer may get help in which source-code should be changed based on the project history [7]. Other researchers proposed a system for recommending code changes, such as bug fixes, function enhancements, and code refactoring [8].

In this paper we propose a feedback recommendation system. This system aims to gain the most benefits from the problems reported by the end-users. It is intended to help both developers and end-users during the maintenance phase throughout their experience in dealing with the software system. This approach relied mainly on the structured feedback acquisition method proposed by [6]. We used commercially available software, namely Solr, as a search engine to match between the newly entered feedback and the existing ones based on a set of pre-defined search configurations.

The rest of the paper is organized as follows: Section 2.0 the background and the related work. Section 3.0 is the proposed framework. Section 4.0 discusses the case study and the evaluation results. Section 5.0 is the conclusion and the future work.

2. Background and related work

During software development process there are some artifacts that are used all together to continue the process. Mining software repositories (MSR) is the use the wealth of information available as a result of the software-development process to learn how the process can be improved [9]. Recommendation Systems in Software Engineering (RSSE) is a major part of MSR. As described in [9] RSSE is "...a software application that provides information items estimated to be valuable for a software engineering task in a given context.". Recommendation systems are usually interactive tool used by either developers or end-users to retrieve information to the current context.

Authors in [10] proposed a solution to recommend documents from digital libraries based on a Vector Space Model (VSM) of subject-based representation of documents. The results are obtained by comparing a subject-based vector for each document with a subject-based vector of the user's query. A subject-based distance is calculated to find the similarity between query and documents. The weight for each term in the vector is an indication of how much this word is related to some topic. After evaluating the results it was found that 83% of users preferred the proposed recommendations. [11] proposed a text-based search system to find the appropriate software design pattern to solve a certain design problem. It allows the developer to enter his design problem as natural language text. It uses Latent Dirichlet Allocation (LDA) and VSM to describe both the design patterns and the design problem. Sqrt-cosine similarity is selected to measure the similarity between the design problem and design patterns. Good results are obtained by using this approach, 72% precision is calculated by evaluating the results. [12] proposed a solution to recommend products to users in and E-commerce system. Unlike most recommender systems in the E-commerce fields, this approach does not rely on other users ratings for products. It uses the VSM to represent both user's query and products. Some pre-processing is applied on the user's query (remove stop-words and stemming). Similarity is calculated using the cosine similarity equation. The results from this approach are considered correct and consistent with most of the literature.

As mentioned in the introduction, structuring the feedback can be very beneficial during the maintenance phase. In her work [6], she proposed a new structured feedback acquisition and communication process. Feedback acquired using the defined structure are categorised by their type. Additional contextual and depth information are used to enrich the feedback and make it more meaningful. To describe a feedback we can say that it belongs to a feedback type, detailed by level of
detail which is described by some method. In addition, [6] defined some rules that regulate how these feedback components are used in order to describe feedback in a structured representation. For example the "Topic Definition" feedback type: 1) cannot reply to an existing feedback, i.e. must be the first feedback in the thread, 2) must be detailed by an "Explanation", and 3) must be detailed by a "Task". Similar rules are defined for each feedback type.

Authors of [13] evaluated how automated approaches can help in extracting features and requirements from different users' feedback. They adopted a semi-automated approach to cluster different features and requirements entered in natural language based on their linguistic similarities. Text is analyzed and preprocessed using some well-known preprocessing techniques: tokenization, stop words removal, stemming, and special-characters removal. After text preprocessing, features are extracted using Term Frequency-Inverse Document Frequency (TF-IDF) which gives each word a rank based on its importance in the text. Also some NLP is used to extract domain objects and operations. Similarity between requirements is calculated using cosine similarity functions. The final stage in feature extraction is requirements clustering and summarization. Results are evaluated using three different unsupervised clustering techniques: K-Means algorithm, Latent Dirichlet Allocation (LDA), and Biterm Topic Model (BTM). K-Means algorithm gives more useful results as compared to the other clustering techniques.

In [14], authors proposed a text-based search system to find the appropriate software design pattern to solve a certain design problem. It allows the developer to enter his design problem as natural language text. It uses Latent Dirichlet Allocation (LDA) and VSM to describe both the design patterns and the design problem. Sqrt-cosine similarity is selected to measure the similarity between the design problem and design patterns. Good results are obtained by using this approach, 72% precision is calculated by evaluating the results.

In [15], the research aims to experiment the most common attribute extraction approaches to extract candidate features from the unstructured users' review data. They introduced a feature extraction process which extracts sentences from the online reviews and categorize them to features and non-features using a Machine Learning (ML) algorithm. Three different binary classification ML algorithms are evaluated: Naïve Bayes (with multinomial and Bernoulli variants), Support Vector Machines (with linear and multinomial variants) and Logistic Regression. The results confirmed that the proposed process is sufficient in extracting candidate features from the online users' reviews.

In their work, [16] they proposed a requirement-driven Software Product Line (SPL) process to collect users' feedback from Social Network Sites (SNS) to evolve the SPL's architecture. NLP and data mining techniques are used to collect requirements from the collected users' feedback. Afterwards the requirements are categorized to build an adaptive architecture. The first step in the process is the requirement elicitation. In this step NPL and data mining techniques like: topic modeling, text summarization, clustering, sentiment, and term extraction techniques, are used to extract the requirements from the users' feedback. The second step is manually creating the feature model based on Common Variability Language (CVL). The final step is to update and modify the SPL architecture to match the new requirement.

Change history of architectural changes can also be used to support in software architectural decisions. In their work [17] introduced an automated approach that extracts architecture change patterns from the architecture-change logs. These extracted patterns are considered to be reusable in other projects. Pattern extraction and identification is performed using a 3-step process that enables the discovery, specification and application of architecture change patterns. Architecture logs are analyzed and formalized as a graph. Sub-graphs are identified using sub-graph mining. These sub-graphs represent the change patterns. The reusability of the change patterns is documented using template-based change pattern specification.

To improve recommender performance, authors of [18] proposed different session-based recommender systems which use both item attributes and user-item interactions. The three suggested approaches are: Item Session-Based Recommender (ISBR), Attribute Session-Based Recommenders (ASBRs), and Feature-Weighted Session-Based Recommenders (FWSBRs). Authors defined what are
items and attributes that are used by the first two recommender approaches. The ISBR and ASBR rely on session data related to items or attributes and also to the user's interaction with the system. FWSBR can be considered as multiple ASBRs with various feature weighting schemes. The results show that FWSBR performs significantly better than ISBRs and ASBRs.

In [19], authors introduced Complex collaborative filtering and Quaternion collaborative filtering (QCF) by making use of complex and Quaternion number in recommender models. The system can be described as a collaborative filtering model but the users and items are represented by complex vectors. They experimented the system using six real-world data sets and proved the effectiveness of their proposed approach.

In their work, [20] introduced a component-based recommender system RESDEC (REcommender System that suggest implementation Components from selecteD fEatures) that is used in the domain of Software Product Line (SPL). RESDEC basically consists of two main parts: knowledge base and recommender generator. They have identified three different scenarios that are related to selecting implementation components to configure a SPL: cold start, Recommendations of implementation components based on ratings, and Recommendations of implementation components based on features. They used Wordpress data set, and the results show that system is able to suggest recommendation with error less than 13%.

3. Proposed framework

To retrieve good recommendations, we need to determine the valuable feedback components that can be used as search criteria. Since we are following the feedback types and rules defined in the structured feedback system [6], feedback items are already described and stored in a structured way which makes it easier to extract the information.

The value of "affected feature" can be used to limit the results to the feedbacks that are affected by exactly the same "affected feature" of the newly entered problem. For example there is a user who has a certain problem when uploading a new file, it wouldn't be helpful at all if he got a feedback recommendation related to another feature of the system even if they have similar text content. That's why we need to exclude all the feedbacks except for those who affect the same feature. The most important feedback components are "task" and "explanation" since they contain the main text that describes the problem. "Feedback title" is also important, it can be considered as a summary of the problem. Additional feedback components that can be used as search fields are "social context", "personal context", "environmental", "exemplification", and "feature definition" [6]. These may contain some extra information that might refine the search to get better and more relevant results.

Based on the mentioned search criteria, we propose a structured feedback recommendation system which provides recommendations of users' feedback. As described in [6], feedback are stored in an ontology and users can enter new feedback by specifying the feedback type and filling all the required components for this type, for example fill the "task" and "explanation" for feedback of type "Topic Definition", and this new feedback serves as an input for the proposed structured feedback recommendation system. Figure 1 describes the main architecture of our proposed system.
As shown in Figure 1, the "Structured feedback system" refers to the system proposed by [6] in which the already existing structured feedback are stored. In the following sub-sections we will describe the details of the structured feedback recommendation system. As an assumption we assume that all the feedbacks are indexed and stored in the Solr server. If not, we need to do so in order to use the recommendation system. In our approach we only search for the feedback of type "Topic Definition", so in this step we only need to index the Topic feedbacks and ignore any other feedback type.

3.1 Index handler
Index handler is responsible for converting the feedback components to a Solr document. As shown in Figure 2, the input is the feedback entered by the user after being validated by the structured feedback system. The Solr document fields are filled with the feedback's components that will be used in both index and query. Data preprocessing may be required in order to set the Solr document field values.

3.2 Query handler
We believe search query is the heart of the recommendation system. Considerable attention must be paid when structuring the query. Solr query primarily consists of several parameters. In our approach values for the parameters filter query "fq", and search fields "q" are set. As was stated in the search criteria, the recommended feedback should be limited to the feedback which affect the same "affected feature". This leads to using the "affected feature" field's value as a filter query. Query handler is also responsible for adding boost values for some fields based on their importance. Sometimes a hit in some field is more important than a hit in another field. There is no defined rule for selecting the boost fields or values; it is completely dependent on the system's need. Based on our defined search criteria the values for the "task" and "explanation" are the most important feedback elements since they contain the main information of the user's problem. The value for the "feedback title" field comes next. Using the afore-mentioned filter query and search fields recommendations for the feedback can be retrieved from Solr server. After the query is generated, it is sent to Solr server and then the resulted documents are translated back to feedback components and returned to the user as feedback recommendations. An overview of how the query handler works is represented in Figure 3.
4. Experiment and case study

When a user enters a new problem there is a chance that there exists another similar problem created by some other user which is already fixed. This could be due to a common mistake like missing configuration, or due to the lack of user's experience in using the software. Recommending the similar problem feedback may help the user to fix his own problem without the need of communicating with the development team.

Since there are no available data sets for the structured feedback, we had to build our own. We manually analyzed several threads from moodle forum "https://bit.ly/318DAAB" to map them to the corresponding feedback types. Moodle is a learning platform designed to provide educators, administrators and learners with a single robust, secure and integrated system to create personalised learning environments. Usually users tend to write their feedback in an unstructured and unorganized way, that's why we had to search for suitable feedback that can be converted to structured feedback. In this phase we went through hundreds of feedback to determine whether they are good candidates to be structured or not. As a result, we had over 40 feedback ready to be converted to structured feedback. Each of these feedback affects one of the following moodle features: adding new course, bulk course creation, or using course as a template. The next step was to extract structured feedback elements from the feedback set. We individually analysed the feedback to map them to the structured feedback elements and then as a second iteration we together discussed the individual findings to come up with the final representation of the feedback. At this point the feedback are ready to be entered in the structured feedback ontology knowledge base. The ontology implementation is already available in "https://bit.ly/2Gzoxbi". Now the recommender system is ready to be tested.

In order to test the system we used one of the previously entered feedback as an input feedback then analyzed the results to check if the sorted recommended list of feedback is logical. For example the following feedback was used as an input feedback:

**Add fields to Create New Course page - some work some don't**

**I am trying to add few new fields to Create Course page, also added columns to mdl_course table, **

**but can't figure it out what was wrong as some of them work but some don't. Does any one has the experience to share?**

The first result was the following:

**Add custom fields to the "Add Course" page**

**can i add custom fields to the "Add Course" page , without made any changing in the core of moodle ? can i add custom fields to the "Edit Activity or resource" page , without made any changing in the core of moodle ?**

As noticed, the two feedback affect the same feature "Adding a new course" , and both users have problem in adding custom fields in the create course page, gladly the recommender system managed to figure out they are similar despite that each user described the problem in his own words.

We performed the same experiment using different feedback as an input and analysed the recommender system results. In real life, it is not necessary that every new problem entered by the user has exactly the same problem that was entered by other user, the same is for our data set, some of the feedback are duplicates of the exact same problem and others are not. For the feedback that have duplicate problems (similar to the example mentioned in the previous paragraph), the recommender system managed to recommend the similar problem as a first result, for the others problems the results
were also logical, the list of recommended results were sorted based on the relativity to the original problem. For example, the following feedback was used as an input feedback:

**Setting number of sections in bulk course upload [feedback title]** In the CSV file for the bulk course upload, I can set the format, but I can't seem to set the number of sections. **[feedback task]** Our default is set at 16 weeks/topics **[social context]**, and even if the template is at 8 weeks/topics, the CSV file is unable to change the default. Does anyone know if this even possible? Or will the default number of sections always carry over? **[feedback explanation]** Thanks! (PS Should have added that we're on 2.7) **[environmental context]**

There is no exact duplicate for this feedback; however the first result was the following:

**Bulk upload of course section titles? [feedback title]** At our institution we use a CSV file to bulk upload all courses in the syllabus at the beginning of the academic year. We then manually enter all section titles (a combination of the date and the title of each lecture) for all courses - about 300 courses with on average 30 to 40 sections each. Manually entering about 10,000 section titles is a huge job **[social context]**, and we would love to be able to populate the section titles in bulk with a CSV file **[feedback task]**. Apparently, this used to be possible with the old Bulk Course Upload tool, as described in the book: Jason Hollowell. "Moodle as a Curriculum and Information Management System." Packt Publishing Ltd, 2011. https://books.google.co.jp/books?id=Zu0O100q7IC&lpg=PT126&ots=uybxESgdua&dq=moodle%20create%20courses%20and%20sections%20by%20csv&hl=ja&pg=PT120#v=onepage&q&f=false

As described in the above link, the CSV file could contain the fields: 'topic0', 'topic1', 'topic2',... 'topic52', corresponding to topic/week headings **[feature definition]**. However, the current Upload Courses tool does not allow these fields. Any advice on how to accomplish this either with the current Upload Courses tool or in some other way would be much appreciated. **[feedback explanation]** Just in case, we are using the following: Course format: Collapsed Topics 3.1.1.2, Moodle version: 3.1.1, Theme: More **[environmental context]**

The previous example shows that even if the input does not have an exact match, the first result proposed by the recommender system is logical, both feedback affect the same feature "Bulk course creation" and both problems are related to adding extra information while bulk creating the courses. We believe that the results emphasize the validity of our proposed recommender system and the defined search criteria.

5. Conclusion and Future Work

An approach for structured feedback recommendation system was introduced. Analysis of the structured feedback system proposed by [6] was performed; as a result we could define the search criteria which are the base for our recommendation system. The system architecture is divided to two main modules: Index handler and Query handler. The actual search is done using a Solr search server which is configured to match the search criteria.

As future work the search may be enhanced by using other Solr features like synonyms. In our approach we only care about the feedback of type "Topic definition" but there are other feedback types that may contain problem statements, i.e. Addition and Problem Extension. Even more they may rephrase or change the actual problem. We propose that further research should be undertaken to investigate how these feedback types can be used for better recommendations. In addition, further evaluation should be conducted with real system users to evaluate the feedback recommendations.

References

[1] Pagano, Dennis & Maalej, Walid. (2013). User Feedback in the AppStore: An Empirical Study. 2013 21st IEEE International Requirements Engineering Conference, RE 2013 - Proceedings. 10.1109/RE.2013.6636712.

[2] Hattie, John & Timperley, Helen. (2007). The Power of Feedback. Review of Educational Research. 77. 81-112. 10.3102/003465430298487.
[3] Schneider, Kurt. (2011). Focusing spontaneous feedback to support system evolution. 165-174. 10.1109/RE.2011.6051645.

[4] Morales-Ramirez, Itzel & Perini, Anna & Guizzardi, Renata. (2015). An ontology of online user feedback in software engineering. Applied Ontology. 10. 297-330. 10.3233/AO-150150.

[5] Panichella, Sebastiano & Di Sorbo, Andrea & Guzman, Emitza & Visaggio, Corrado Aaron & Canfora, Gerardo & C. Gall, Harald. (2015). How Can I Improve My App? Classifying User Reviews for Software Maintenance and Evolution. 10.1109/ICSM.2015.7332474.

[6] Sherief, N. H. H., 2017. A framework for modelling and utilization of users' feedback for software systems evolution. Doctorate Thesis (Doctorate). Bournemouth University.

[7] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl. Mining version histories to guide software changes. Software Engineering, IEEE Transactions on, 31(6):429 - 445, june 2005. doi:10.1109/TSE.2005.72.

[8] Higo, Yoshiki & Ohtani, Akio & Hayashi, Shinpei & Hata, Hideaki & Shinji, Kusumoto. (2015). Toward Reusing Code Changes. 372-376. 10.1109/MSR.2015.43

[9] Kwan, Irwin and Damian, Daniela, 2011. A Survey of Techniques in Software Repository Mining. Technical Report DCS-340-IR, University of Victoria

[10] Mahmood Bakhshayesh, Sayed & Mohebi, Azadeh & Ahmadi, Abbas & Badamchi, Amir. (2018). A New Subject-based Document Retrieval from Digital Libraries Using Vector Space Model. 161-164. 10.15439/2018F260.

[11] Hamdy, A & Elsayed, M. (2018). Towards more accurate automatic recommendation of software design patterns. Journal of Theoretical and Applied Information Technology. 96. 5069-5079.

[12] Redhu, R., Shubham, Ittan, V., & Rasal, P. S. (2018). PRODUCT RECOMMENDATION SYSTEM. International Journal of Computer Engineering and Applications., 155-160.

[13] Khan, Javed & Liu, Lin & Jia, Yidi & Wen, Lijie. (2018). Linguistic Analysis of Crowd Requirements: An Experimental Study. 24-31. 10.1109/EmpiRE.2018.00010.

[14] Hamdy, A & Elsayed, M. (2018). Towards more accurate automatic recommendation of software design patterns. Journal of Theoretical and Applied Information Technology. 96. 5069-5079.

[15] Buchan, Jim & Bano, Muneera & Zowghi, Didar & Volabouth, Phonephasouk. (2018). Semi Automated Extraction of New Requirements from Online Reviews for Software Product Evolution. 10.1109/ASWEC.2018.00013.

[16] Ali, Nazakat & Hoing, Jung-Eui. (2018). Requirements-driven Evolution in Software Product Lines.

[17] Ahmed, Aakash & Pahl, Claus & Altamimi, Ahmed & Alreshidi, Abdulrahman. (2018). Mining Patterns from Change Logs to Support Reuse-Driven Evolution of Software Architectures. Journal of Computer Science and Technology. 33. 1278-1306. 10.1007/s11390-018-1887-3.

[18] Hwangbo, Hyunwoo & Kim, Yang Sok. (2019). Session-Based Recommender System for Sustainable Digital Marketing. Sustainability. 11. 3336. 10.3390/su11123336.

[19] Zhang, Shuai & Yao, Lina & Vinh Tran, Lucas & Zhang, Aston & Tay, Yi. (2019). Quaternion Collaborative Filtering for Recommendation.

[20] Rodas-Silva, Jorge & Galindo, José & García-Gutiérrez, Jorge & Benavides, David. (2019). Selection of software product line implementation components using recommender systems: An application to Wordpress. IEEE Access. PP. 1-1. 10.1109/ACCESS.2019.2918469.