

Toward Data-Driven Requirements Engineering

Walid Maalej, University of Hamburg

Maleknaz Nayebi, University of Calgary

Timo Johann, University of Hamburg

Guenther Ruhe, University of Calgary

// Developers, requirements analysts, and managers could systematically use explicit and implicit user feedback in an aggregated form to support requirements decisions. The goal is data-driven requirements engineering by the masses and for the masses. //



REQUIREMENTS ENGINEERING

(RE) identifies, documents, negotiates, and manages the desired properties and constraints of software-intensive systems, the goals to accomplish in a software project, and the assumptions about the environment.¹ Requirements are a verbalization of decision alternatives regarding a system's functionality and quality.²

RE focuses mostly on involving system users, capturing their needs, and getting their feedback.

Conventional RE typically involves users through interviews, workshops, and focus groups. Open source projects let users publicly report issues and ideas through issue trackers. Recently, software vendors also started collecting user feedback

through social media channels, user forums, and review systems. In particular, with the emergence of app stores as a software marketplace and deployment infrastructure, users can easily submit their feedback, review new releases, report bugs, rate apps and their features, or request new features.³

Software products' success often depends on user feedback.⁴ For instance, apps with more positive reviews get better rankings, better visibility, and higher download and sales numbers.^{5,6} In contrast, some frustrated users might harm an app's reputation by organizing social media campaigns against it.⁷ As large portions of software are delivered and used through app stores, more feedback from the user masses becomes available. Popular apps such as Dropbox or Facebook can get several thousand reviews in a day.³ Conventional environments (such as desktop OSs) are following this trend in delivering and deploying software. Large systems such as Microsoft Office, Facebook, Firefox, or Eclipse even have their own plug-in stores with similar rate-and-review features.

Here, we discuss how software developers and analysts can use this user data to identify, prioritize, and manage the requirements for their software products. We see three future directions in practice. First, tools for feedback analytics will help deal with the large numbers of user comments by classifying, filtering, and summarizing them. Second, automatically collected usage data, logs, and interaction traces could improve feedback quality and help developers understand feedback and react to it. We call this automatically collected information about software usage *implicit feedback*. Finally, with explicit and implicit



feedback now available (almost) continuously, questions arise. How can practitioners use this information and integrate it into their development processes to decide when to release updates? What requirements and features should they add, change, or eliminate?

User Feedback Analytics

Analytics is the use of analysis, data, and systematic reasoning to make decisions.⁸ App store operators such as Apple, Google, and Microsoft, as well as specialized companies such as AppAnnie and SimilarWeb, offer analytics services based on user feedback to software vendors. At the beginning of 2014, a survey of 7,000 developers found that approximately 40 percent employed user analytics tools.⁹ By the third quarter of 2014, only 21 percent of the surveyed developers still used these tools.¹⁰ A primary reason was that most analytics services focus on sales and marketing decisions, showing download and sales numbers over time, user demographics, and sales behavior. However, research has shown that feedback analytics could also support software engineering and RE decisions.^{11–13}

Classifying User Feedback

User feedback includes a variety of information.³ It might include a bug report, which describes a problem in an app that needs correction (such as a crash, erroneous behavior, or a performance issue). Feedback might also include a feature request, in which users ask for missing functionality (possibly provided by other apps) or missing content (for example, in catalogs and games). Users might share ideas on how to improve the app by adding or changing features. Some feedback describes

a user experience, documenting a particular experience or specific feature (such as how the app helped in a certain situation). This feedback helps provide documentation of the app, its requirements, and features. Finally, user feedback frequently includes a rating comment, a brief text

contains only short rating comments can be summarized or filtered out. Rating comments are pervasive in app reviews (they appear in up to 70 percent of reviews) but are rather uninformative for development teams.³ So, filtering them will save time. Feedback with bug reports can be

Feedback describing user experiences can serve as ad hoc documentation for software or to derive user stories.

explanation of a numeric or star rating. Rating comments are less informative, often containing only praise or blame.

Analytics tools can help classify and filter user feedback according to the information it contains. Researchers have studied how to use text classification, natural-language processing, and other heuristics (such as star ratings, text length, and the text's tense) to categorize reviews.^{11,14} Recent preliminary results show that no single classifier works best for all review types and data sources.¹¹ Reviews might include information that falls into multiple categories. A review might include sentences with different types of information—for example, one sentence being a bug report and another being a feature request. Thus, analytics tools might need to first tokenize the reviews into parts (sentences or phrases) and then extract the information types separately.

Classifying user feedback can facilitate considering and processing it more efficiently—for example, by forwarding single entries to appropriate stakeholders and development teams. For instance, feedback that

forwarded in an aggregated form to developers and quality management representatives. Assigning feedback with feature requests to requirements analysts and managers will let them discuss and identify requirements for future releases.

Feedback describing user experiences can serve as ad hoc documentation for software or to derive user stories. The feedback entries can also serve as manuals or posted FAQs to help other users. This type of feedback can also help requirements analysts better understand users, particularly those who have never been considered a target group. Some feedback describing user experiences also describes workarounds and unusual situations that could inspire designers to develop test cases, capture exceptions and alternate flows, or design new features.

Automatic feedback classification can provide an overall picture of app usage and user engagement (for example, the number of bug reports or feature requests). This can also be used for comparing releases over time (in terms of the provided and requested requirements or the bug reports) or comparing similar apps.

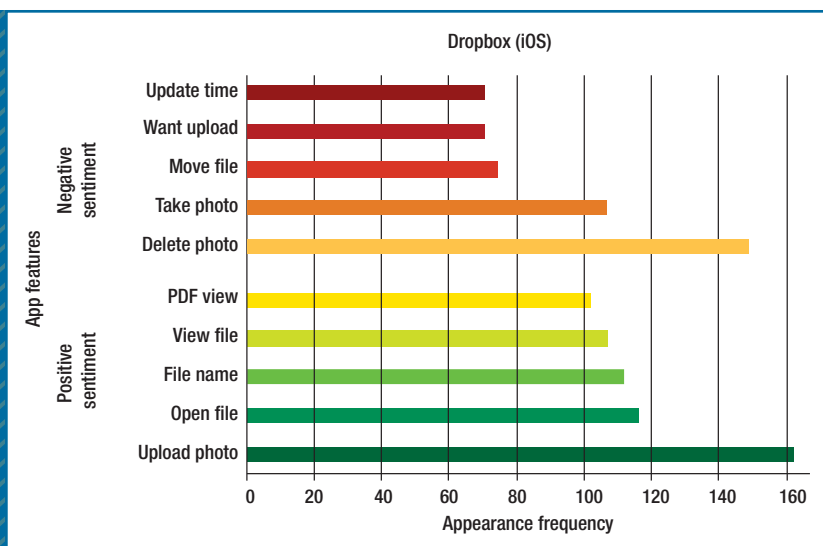


FIGURE 1. Feature-based feedback summarization for Dropbox.¹² Such an approach helps engineering teams monitor users' reactions to the software features by answering two questions: Which features do users talk about most? Which features do they perceive positively or negatively?

Classifying Stakeholders

Analytics can also help identify and classify user groups and stakeholders with different characteristics and needs. In particular, deviant users and exceptional use cases can be of interest to requirements analysts and developers.

For instance, searching the iOS App Store for health-related terms revealed that a magnifier app and a translator app received many positive reviews by nurses. Interestingly, neither app was from the Health & Fitness category. Nurses frequently used the magnifier app to read medicine containers' small-text package leaflets. They frequently used the translator app to communicate with patients who speak a different language.

Analytics tools can mine feedback to identify descriptions of usage patterns, uncommon vocabulary, or role and task descriptions. Observing software's "unintended

use" might lead to repackaging an app, reordering priorities, or adding requirements to support new target groups.

Classifying stakeholders might also help analysts identify, understand, and specify nonfunctional requirements more concretely. Examples include accessibility (for example, identifying how people with disabilities use the software), privacy (identifying which privacy tradeoffs are acceptable for which stakeholders), or performance (identifying which performance requirements apply to which user groups). A simple search for app reviews should include sentences such as "As a blind person, I ...," "As an official, I ...," or "As a frequent traveler, I ...," which will reveal reviews with such details.

Summarizing Reviews

Recently, researchers suggested two probabilistic approaches to summarize informative review content.

Feature-based summarization. Researchers have applied natural-language processing and sentiment analysis to extract software features from user reviews and summarize opinions about each feature (see Figure 1).¹² This approach helps engineering teams monitor users' reactions to the software features by answering two questions: Which features do users talk about most? Which features do they perceive positively or negatively?

Such information helps analysts quantify software features' importance and prioritize work for future releases. Breaking down user evaluations at the feature level also helps monitor the features' "health" over time (before and after major releases). Analysts can also use this to locate software's buggy parts or components more easily.

The major challenge for feature-based summarization is to identify the features in the reviews together with their associated opinions. Users might have different opinions about different features and mention them all in one review. Moreover, they might use different vocabulary to refer to the same feature, which makes purely syntactical approaches inaccurate. Finally, user reviews often contain sarcasm, such as "Great, now I lost all my data with the update." This makes automated sentiment analysis more challenging.

Topic-based summarization. Researchers extracted word-based topics from reviews and assigned sentiments to them using topic modeling (such as latent Dirichlet allocation) and sentiment analysis.¹³

Similarly, Ning Chen and his colleagues proposed AR-Miner, a review analytics tool for summarizing informative app reviews.¹⁵ The

tool first filters noisy and irrelevant reviews, such as rating comments. Then, it summarizes and ranks informative reviews using topic modeling and heuristics from the review metadata. This tool's main use case is to summarize and visualize discussion topics in reviews. For instance, the topic can be the whole release, the app, the price, or the support quality.

Similar Apps: Looking beyond the Fence

App stores use categories to group similar apps. Popular categories are generic (such as games, social media, or productivity) but can be more fine-grained (such as email apps or navigation systems). Combining official information about similar apps (such as the list of features, the release history, and prices) together with reviews and rating comments helps identify and prioritize new app requirements. Requirements definition and scoping becomes an optimization problem: determining the new app's optimal set of features on the basis of experience from similar apps.

Examining similar apps in a category or across categories is a great way to define potential features.⁶ We studied 271 similar apps in the same category to create apps by combining features from existing apps. Applying a crowdsourced evaluation of feature attractiveness among app users in conjunction with expert-based effort estimation, we synthesized user satisfaction with new feature combinations. Then, we balanced the users' opinions and priorities with the effort estimate to determine which features to choose for a new app.

This approach can also serve to recommend features to remove from or add to an existing app. Overall, the results help vendors understand

which features to change in order to improve overall performance. However, vendors should consider additional analysis dimensions, such as innovative features, business or technical requirements, and dependencies between features and related requirements.

Like explicit feedback, usage data must be analyzed, filtered, summarized, and visualized to add value for developers and analysts. Implicit usage data might include users' individual clicks and interactions with the user interface. Graphical elements are typically associated with

Examining similar apps in a category or across categories is a great way to define potential features.

Implicit Feedback from Usage Data

So often, user feedback—particularly negative feedback—isn't helpful to developers and analysts because it lacks context.^{3,16} Users tend to discuss what led to positive experiences (and thus positive ratings). But they give short, uninformative reviews for negative ratings, such as “fire the idiot who developed this app,” or “worst experience since dating my ex.”

Usage data and interaction history, though, help developers and analysts better understand the circumstances under which users submit feedback.⁷ If captured continuously and presented to development teams, this implicit feedback enhances explicit feedback in the form of reviews and comments. Whereas explicit feedback summarizes users' subjective opinions, implicit feedback helps developers understand their objective opinions (such as how and when they actually use the software and its features). With usage data, developers can better understand users' feedback or decisions (for instance, to uninstall, reconfigure, or downgrade software) and derive respective development decisions.

specific app features, components, or requirements; analyzing their usage frequency and sequence also might help developers understand users' needs. Other usage data includes clusters of feature-usage sequences, performance information, and information about overall usage sessions. This might simply include the time, location, duration, or apps that are used together with the target app. However, more advanced situation descriptions based on physical sensors or explicit interactions (for example, activating the private-browsing button or a virtual private network) can also reveal usage situations and context.

Much research exists on collecting and processing usage data for software engineering, focusing mainly on error reproduction and localization. Tools such as Apple CrashReporter, bug-buddy, Mozilla Talkback, and Google Breakpad collect memory dumps and report to central failure repositories. These approaches don't capture user interactions and usage data, concentrating instead on performance and quality. Other approaches use interaction data and usage data to improve

system usability, provide recommendations to users, or conduct usability testing.¹⁷

Finally, usage data can support developers' and requirements analysts' decisions, even without explicit feedback. They can use it to study feature usage's sequence, duration, and other contexts (such as the time of day used) to better understand users' needs. Thermal tracking of a user's finger movement on touchscreens is a common approach for usage data analysis.

The Future of RE Decision Making

RE has been considered decision-centric—and nothing about that will change. Nevertheless, with the new analytics trends initiated from collecting masses of explicit and implicit user data, decisions about requirements will change (along with the decision process). We project a transition from stakeholder-centered, intuition- or rationale-based decisions to group-based, mass-driven, user-centered decisions. Such decisions will be based on real-time analysis of a range of information sources.¹⁸ The changes will relate to how decisions are made, who makes them, what's being decided, and when they're made.

How Are Decisions Made?

Currently, RE decision making is typically based on the stakeholders' intuition and experience, rationale schemes (such as criteria, options, and arguments), or both.^{1,2} Intuition is subjective, is potentially inconsistent, and lacks explanation. Rationale changes over time and is difficult to capture and externalize. With the mass of available explicit and implicit user feedback, synergies between computational

intelligence and human expertise look promising.

From systematically observing user communities, forums, social media channels, and review platforms, a range of information that supports requirements decisions is available.⁷ Future research will combine known and used data sources (business requirements, system and technical requirements, stakeholder preferences, and requirements interdependencies) with aggregated user data. To do so, we'll need new data analytics methods to synthesize information and provide constructive insights and recommendations. Technologically, we expect to have predictive modeling, scenario analysis, wideband Delphi analysis, and adaptive group decision making to support this process.

Who Makes the Decisions?

The traditional setting of well-defined decision makers and nominated stakeholders will roll into a more comprehensive setting that includes a group of unknown users. Social media and other communication channels (issue trackers and user communities such as UseVoice) can involve users and their data in the process. Crowdsourcing could enlarge the set of stakeholders who elicit and manage requirements. The First International Workshop on Crowd-Based Requirements Engineering produced an agenda for crowd-based RE. This agenda embraces event logging, usage mining, text mining, and motivational elements.

Communication with users will move from being unidirectional (from vendors to users) to being bidirectional, including development teams in addition to conventional sales and marketing teams. To increase user involvement, the

crowd-based RE community has discussed approaches to harness social networks to perform RE activities such as elicitation, prioritization, and negotiation. Another option is to customize e-democracy and e-participation approaches for requirements scenarios.¹⁹

What's Being Decided?

With development processes' increasing agility, decisions' content is also changing. Selecting a requirement is no longer a Boolean decision. Instead, requirements are implemented incrementally. Decisions are related to enhancing or shrinking the functionality and increasing the level of specified target quality. Software engineering for game development is a forerunner in this trend, as features are trialed and—depending on the user's response—enhanced or eliminated. Also, the criteria for these decisions are changing both generally and specifically over time. Explicit and implicit user feedback should and will be taken into consideration more for decisions about what and when to implement and deliver.

Finally, the boundary between conventional requirements decisions, quality assurance, project management, or sales decisions will continue to dissolve as data increasingly mixes and the time between prior development and postdevelopment becomes less important.

When Are Decisions Made?

The shift from reactive to real-time and even proactive decision making is crucial in developing software-intensive products. To create products rapidly with higher customer acceptance, developers must incrementally build and deploy products that rely on deep customer insight and real-time feedback. Changes

to objectives, priorities, and dependencies (along with performance and other constraints) occur in real time. Significant changes in any of these parameters trigger the need to adjust the underlying development processes. Infrastructures and methodologies for real-time (or at least just-in-time) decision making will become increasingly important for software engineering projects.

We foresee requirements identification, negotiation, and analysis as an open process with stronger interaction between developers, analysts, and users. There's evidence of this for mobile-app stores and marketplaces. In addition, we see this as an essential trend for a broader class of software products and services. However, we're not saying that this applies to all types of software development.

Yet moving toward data-driven RE—by the masses and for the masses—bears challenges for researchers and practitioners beyond the technical issues mentioned in this article. For example, scaling analysis and management of user input to Internet levels requires vast human resources and sophisticated support tools.

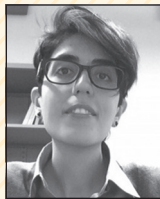
Also, motivating users to qualitatively contribute to tasks for which they aren't accountable is difficult, yet crucial. Giving the right incentives to receive evenly distributed and qualitative feedback will be important. In addition, when making decisions based on user data, we must ensure the representativeness of the sample of users and the data's quality.

Sometimes users' contributions seem illogical or subjective, lacking a full understanding of requirements

ABOUT THE AUTHORS



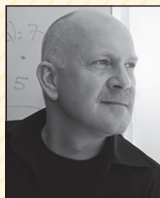
WALID MAALEJ is a professor of informatics at the University of Hamburg and the chair for applied software technology. His research interests include data-driven software engineering, context-aware adaptive systems, e-participation and crowdsourcing, and software engineering's impact on society. Maalej received a PhD in software engineering from the Technical University of Munich. He's an editorial board member of the *Journal of Systems and Software* and a Junior Fellow of the German Computer Science Society (GI). Contact him at maalej@informatik.uni-hamburg.de.



MALEKNAZ NAYEBI is a doctoral candidate in the University of Calgary's Software Engineering Decision Support Lab. Her main research interests are mining software repositories, release engineering, open innovation, and empirical software engineering. Nayebi received a master's in information technology from the Amirkabir University of Technology. She's a student member of IEEE and ACM. Contact her at nayebi@ucalgary.ca.




TIMO JOHANN is a research associate and doctoral candidate in the University of Hamburg's Applied Software Technology Group. His research interests include user involvement, crowdsourcing requirements engineering, social aspects of software, and social software engineering. Johann received an MSc in computer science from the University of Applied Sciences in Trier. He holds a scholarship from the Friedrich Ebert Foundation and is a student member of IEEE and ACM. Contact him at johann@informatik.uni-hamburg.de.



GUENTHER RUHE is the Industrial Research Chair in Software Engineering at the University of Calgary. His research focuses on product release planning, software project management, decision support, data analytics, empirical software engineering, and search-based software engineering. Ruhe received a habilitation in computer science from the University of Kaiserslautern. As of 2016, he's the editor in chief of *Information and Software Technology*. He's a senior member of IEEE and a member of ACM. Contact him at ruhe@ucalgary.ca.

decisions; this subjectivity can hinder innovation. To resolve this, we need ways for developers and analysts to communicate with users and explain requirements decisions to them. Similarly, masses of crowd data will most likely lead to conflicting contributions that will need to be resolved.

Moreover, users are often concerned about how their contributions will be used. Analyzing the data could result in the ethically questionable extraction of users' personal information. On the other hand, it's important to ensure the authenticity of the participation.

Finally, changing software engineering teams' mind-set to accept users as equal stakeholders with potentially good ideas and suggestions is an important cultural challenge. 

References

1. A. Davis, "The Art of Requirements Triage," *Computer*, vol. 36, no. 3, 2003, pp. 42–49; <http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1185216>.
2. A. Aurum and C. Wohlin, "The Fundamental Nature of Requirements Engineering Activities as a Decision-Making Process," *Information and Software Technology*, vol. 45, no. 14, 2003, pp. 945–954; <http://linkinghub.elsevier.com/retrieve/pii/S095058490300096X>.
3. D. Pagano and W. Maalej, "User Feedback in the AppStore: An Empirical Study," *Proc. 21st IEEE Int'l Conf. Requirements Eng. (RE 13)*, 2013, pp. 125–134.
4. H. Li et al., "A User Satisfaction Analysis Approach for Software Evolution," *Proc. IEEE Int'l Conf. Progress in Informatics and Computing*, vol. 2, 2010, pp. 1093–1097.
5. T.W. Gruen, T. Osmonbekov, and A.J. Czaplewski, "eWOM: The Impact of Customer-to-Customer Online Know-How Exchange on Customer Value and Loyalty," *J. Business Research*, vol. 59, no. 4, 2006, pp. 449–456.
6. F. Sarro et al., "Feature Lifecycles as They Spread, Migrate, Remain, and Die in App Stores," *Proc. 23rd IEEE Int'l Requirements Eng. Conf. (RE 15)*, 2015, pp. 76–85.
7. W. Maalej and D. Pagano, "On the Socialness of Software," *Proc. IEEE 9th Int'l Conf. Dependable, Autonomous, and Secure Computing (DASC 11)*, 2011, pp. 864–871; <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6118890>.
8. T. Davenport, J. Harris, and R. Morison, *Analytics at Work: Smarter Decisions, Better Results*, Harvard Business Press, 2010; <https://books.google.de/books?id=2otJuvfvfgC>.
9. *Developer Economics Q1 2014: State of the Developer Nation*, VisionMobile, 2014; www.visionmobile.com/product/developer-economics-q1-2014-state-developer-nation.
10. *Developer Economics Q3 2014: State of the Developer Nation*, VisionMobile, 2014; www.visionmobile.com/product/developer-economics-q3-2014.
11. W. Maalej and H. Nabil, "Bug Report, Feature Request, or Simply Praise? On Automatically Classifying App Reviews," *Proc. 23rd IEEE Int'l Requirements Eng. Conf. (RE 15)*, 2015, pp. 116–125.
12. E. Guzman and W. Maalej, "How Do Users Like This Feature? A Fine Grained Sentiment Analysis of App Reviews," *Proc. IEEE 22nd Int'l Requirements Eng. Conf. (RE 14)*, 2014; <http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=6912257>.
13. L.V. Galvis Carreño and K. Winblad, "Analysis of User Comments: An Approach for Software Requirements Evolution," *Proc. 2013 Int'l Conf. Software Eng. (ICSE 13)*, 2013, pp. 582–591; <http://dl.acm.org/citation.cfm?id=2486788.2486865>.
14. C. Iacob and R. Harrison, "Retrieving and Analyzing Mobile Apps Feature Requests from Online Reviews," *Proc. 10th Working Conf. Mining Software Repositories*, 2013, pp. 41–44; <http://dl.acm.org/citation.cfm?id=2487085.2487094>.
15. N. Chen et al., "AR-Miner: Mining Informative Reviews for Developers from Mobile App Marketplace," *Proc. 36th Int'l Conf. Software Eng. (ICSE 14)*, 2014, pp. 767–778; <http://doi.acm.org/10.1145/2568225.2568263>.
16. N. Bettenburg et al., "What Makes a Good Bug Report?," *Proc. 16th ACM SIGSOFT Int'l Symp. Foundations of Software Eng.*, 2008; <http://portal.acm.org/citation.cfm?id=1453101.1453146>.
17. W. Maalej, T. Fritz, and R. Robbes, "Collecting and Processing Interaction Data for Recommendation Systems," *Recommendation Systems in Software Engineering*, Springer, 2014, pp. 173–197; <http://dx.doi.org/10.1007/978-3-642-45135-5>.
18. M. Nayeibi and G. Ruhe, "Analytical Product Release Planning," *The Art and Science of Analyzing Software Data*, Morgan Kaufmann, 2015, pp. 550–580.
19. T. Johann and W. Maalej, "Democratic Mass Participation of Users in Requirements Engineering?," *Proc. 23rd IEEE Int'l Requirements Eng. Conf. (RE 15)*, 2015, pp. 256–261.



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.