Data-Driven Requirements Engineering: A Guided Tour

Xavier Franch[0000-0001-9733-8830]

Universitat Politècnica de Catalunya (UPC-BarcelonaTech), Barcelona
franch@essi.upc.edu

Abstract. Data-driven approaches are becoming dominant in almost every single software engineering activity, and requirements engineering is not the exception. The analysis of data coming from several sources may indeed become an extremely useful input to requirements elicitation and management. However, benefits do not come for free. Techniques such as natural language processing and machine learning are difficult to master and require high-quality data and specific competences from different fields, whilst their generalization remains as a challenge. This paper introduces the main concepts behind data-driven requirements engineering, provides an overview of the state of the art in the field and identifies the main challenges to be addressed.

Keywords: Requirements Engineering, Data-driven Requirements Engineering, Feedback, Natural Language Processing, Software Analytics, Monitoring, Decision-making, Release Planning.

1 Introduction

Identifying, documenting and managing requirements has been part of engineering tasks from old times. In the realm of software systems, requirements engineering (RE) [1] as a discipline originated more than 40 years ago [2]. The importance of managing requirements properly became evident very soon. Several studies quantified the cost of fixing errors to be about 10-100 times greater in later phases of software development and maintenance than in the requirements phase [3][4]. This observation motivated the fast emergence of research related to requirements and the consolidation of RE as a well-established software engineering area on its own.

In recent years, we still find evidence that RE plays a central role in software project success. Requirements understanding and "-ilities" (non-functional requirements) are reported to be the most influential factors on cost[1] and in the particular case of non-functional requirements, failure to satisfy them can be catastrophic (resulting in a system worse than useless) [5]. Industry reports go along the same direction. For instance, the Project Management Institute (PMI) reported that inaccurate requirements

[1] Quote from Ricardo Valerdi (U. Arizona & SpaceX) slides in seminar "Cost Estimation in Systems Engineering" given at UPC-BarcelonaTech, Sept. 2017.

management is the primary cause of unsuccessful projects (i.e., not meeting their original goals and business objectives) almost half of the times (47%) [6].

Given that requirements are aimed to express needs of all system's stakeholders, the question that arises is: how to ensure that a system is delivering the right value to its stakeholders? Among the several directions of research addressing this question, data-driven RE is a prominent, emerging strand of research. Data-driven RE adopts a different perspective than traditional RE methods, shifting the focus from the interaction with the stakeholders at system design time, to the exploitation of runtime data.

2 An Overall View of Data-Driven Requirements Engineering

The term data-driven requirements engineering (DDRE) was proposed to the community in a seminal paper by Maalej et al. published in 2016 [7], in which they defined DDRE as "RE by the masses and for the masses". The main motivation for DDRE is to take profit of the existence of large amounts of data in the form of feedback to guide requirement engineers in their decisions about what requirements to include in subsequent system releases. While the concept of feedback is very generic and exists from long ago [8][9] (e.g., in the form of issues stored in an issue tracker in open source projects), it has been with the emergence of applications for mobile devices (apps) that feedback has gained momentum. Apps' users can easily comment and provide their opinions through adequate feedback gathering mechanisms in app stores [10] while gathering data about system usage is also commonplace today.

DDRE conveys the need of a continuous cycle to make actionable the gathered data. This is illustrated in **Fig. 1**, which adapts the cycle proposed in the Q-Rapids project [11][12]. At the topmost left part of the figure we find the set of requirements for the software system, which can be stored in a product backlog or some other format (even a word file). Requirements are prioritized in a way that a software development team implements a subset of them in the next release (possibly after refining them into concrete development tasks). Again, this can be done in different ways depending on the software development process, ranging from a traditional process to an agile one, even in the extreme a continuous software development process [13] where the concept of release gets diluted and instead, requirements are continuously selected and implemented.

Once the (latest release of the) software system is deployed, users will use it for their own purposes. While they use it, some data can be collected in a transparent manner, through usage logs, system monitors and similar instruments. These data are known as implicit feedback, since their gathering does not require the explicit intervention of the user. Besides, the user can provide explicit feedback provided that some communication channels are available. Explicit feedback will be often textual, although other modalities exist.

Feedback can be seen as a stream of data with potential to deliver insights that ideally can be made actionable. Therefore, the DDRE cycle inherently includes an activity for data analysis. The capabilities provided by this activity are a key point in every DDRE approach. Typically, this activity will clean, combine and analyse the feedback in order to compute values for factors or indicators, uncover patterns of behaviour, raise alerts, etc. In the ideal case, these consolidated data are rendered through a software analytics tool. Beyond advanced visualization techniques, this type of tool offers diverse capabilities to support requirement engineers in deciding upon new or modified (even removed) requirements, which are eventually implemented, generating new data for the next iteration in the cycle.

In the following sections, we examine the key elements of this cycle.

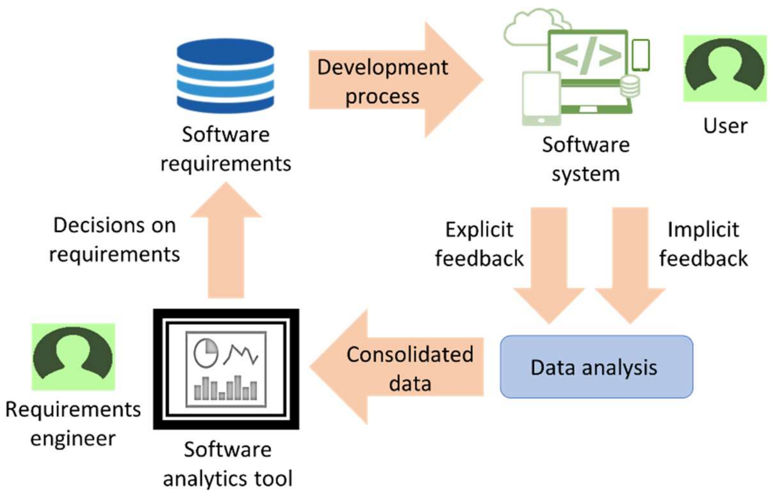

Fig. 1. The Data-Driven Requirements Engineering cycle (adapted from [11]).

3 Explicit Feedback Management

Explicit feedback is the term that denotes the feedback directly provided by the system's users. User involvement is what differentiates explicit feedback from implicit feedback: users may choose if and when to provide such type of feedback. It is worth to mention that many researchers still use the traditional term "user feedback" with the meaning of explicit feedback. However, we find "explicit feedback" more accurate and less prone to ambiguity.

Roughly speaking, explicit feedback management comprises two phases: gathering and analysis.

3.1 Explicit Feedback Gathering

Explicit feedback gathering is determined by the following characteristics.

Types of explicit feedback. As it happens with requirements themselves, explicit feedback is most usually provided in natural language. But differently than requirements, the language used in explicit feedback is normally unstructured, eventually with typos or careless grammar, including emoticons or punctuation signs to emphasise the message given, and hints and clues that makes difficult its analysis [14]. In addition, explicit feedback may be multi-modal, i.e. may include images (photos, screenshots, …), audio recordings, videos or other attachments. Together with text, or alternatively to text, explicit feedback may include some type of evaluation, through ratings (typically using stars or a number) or emoticons.

Communication channel. In order to be processable in a DDRE cycle, the communication channel needs to be persistent and accessible for easy processing. In the case of apps, app stores are the most popular channel nowadays. Forums, ticket systems and social media (typically Twitter) are also widely used, although the difficulty of discriminating the information increases. There are also tools in the market such as UserVoice[2] and Usabilla[3] aimed at adding feedback channels to existing software systems or web pages.

Communication style. Whilst the typical style is push, in which the user has the lead and decides when to provide the feedback, we may find also the pull style, in which the system prompts the user at designated moments, either a fixed moment (e.g., Skype when finalizing a call) or when some condition happens.

Advanced features. For instance, the capability given to requirement engineers or developers to rate the quality of the given explicit feedback, which may help to identify the most valuable users from the perspective of feedback provision. Also, the possibility to establish a bidirectional feedback channel in which the requirements engineer may interact with the user, to ask for clarifications or more details on the feedback initially given.

3.2 Explicit Feedback Analysis

Given that, as said above, explicit feedback consists mostly of text written in natural language, we focus on this type of feedback in the rest of the subsection.

The complexity of dealing with natural language is well known from many decades ago. As a response to this difficulty, the research area of natural language processing (NLP) emerged in the late 40s. NLP explores how computers can be used to understand and manipulate natural language text or speech to do useful things [15]. Many disciplines have used and are increasingly using NLP with different purposes, and RE is one of them [16][17]. NLP techniques in RE research are complemented with machine learning (ML), which allows learning from data (in different variations, which may involve the human in the loop, e.g. supervised learning [18]).

In RE, NLP is used for undertaking different types of analysis. In this paper, we cover three of them:

[2] https://usabilla.com/

[3] https://www.uservoice.com/

- Categorization: the supervised grouping of feedback items into predefined categories.
- Sentiment analysis: the capability of understanding the person's intention behind her feedback.
- Topic modelling: the unsupervised organization of feedback items into one or more thematic topics.

All of these activities have a common need, namely the need of preprocessing the natural language text that forms the feedback communicated explicitly by the user.

Preprocessing. The main purpose of preprocessing is transforming a stream of characters that form a piece of text, in our case the explicit feedback provided by a user, into a syntactic structure formed by lexical units. This activity is typically broken into the following steps:

- Tokenization, which splits a stream of text into a list of words or phrases (the tokens) [19]. Stop words like "the", "an" or "with" are usually removed (some authors consider stop words removal as a step on its own). Stop words are either predefined or are computed analysing their frequency, discrimination power and prediction capability [20].
- Stemming or Lemmatization, both aimed at reducing variant forms into a base form (for instance, past/present/future of a verb; plural/singular of a noun). While stemming basically "cuts" suffixes [21], lemmatization is able to find inflections of variant forms (e.g., "be" and "was"), looking up headwords in a dictionary [22]. Lemmatization is not always convenient, e.g. the use of verb tenses may help classifying a feedback item into bug or feature request.
- Part-of-speech (PoS) tagging [23], segments the sentence into syntactic units with tags as "adjective" or "verb"). It can be followed by a parsing step that creates a parse tree showing the syntactic nature of the text.

All these techniques face several challenges. For instance, tokenization needs to handle multi-word terms. For PoS tagging, the main problem is to determine the right tags for those words that allow for more than one; for instance, the word "back" that can be labelled as a noun, as a verb or as part of a phrasal verb.

Once the text is preprocessed, we can further apply other techniques over the resulting syntactic structure.

Categorization. The classical example of categorization in the field of explicit feedback analysis is the classification of an explicit feedback item as bug report of feature request. For instance, Morales et al. proposed a categorization technique [24] based on speech-act analysis [25]. This technique first gathers feedback from discussions held in online media (e.g., forums). Then it applies a preprocessing pipeline based on the techniques mentioned above, first removing noisy text and then annotating the resulting input with speech-acts using lexico-syntactic rules. Last, it runs some ML algorithms over the annotated sentence in order to classify the feedback into three possible categories: new feature request, enhancement request or bug report.

This simple categorization can be further elaborated using finer-grained classification schemas. For instance, Guzman et al. implemented a detailed classification of twitter opinions on software [26]. Their classification schema shows that feedback can be related not only to particular features (shortcoming, strength, request) or bug reports, but also general feedback as “General praise”, “Software price” and others. Furthermore, their schema proposes some general-purpose categories: “Noise” (which means “impossible to process”, e.g. too many illegible symbols), “Unclear” (ambiguous), “Unrelated” (valid tweet but out of scope of the study) and “Other” (valid but it cannot be classified in the predefined categories).

Sentiment Analysis. It is the process of deciding if a piece of text expresses a particular affect or mood [27].

Current approaches to sentiment analysis combine the use of dictionaries and other syntactic elements [28] with machine learning or even deep learning techniques [29]. Guzmán and Maalej [30] proposed an approach based on the assignment of quantitative values to different sentence tokens that compose an explicit feedback item. In the general case, a sentence may combine positive and negative messages (e.g., “had fun using it before but now it is really horrible :(help!!”). Therefore, the individual values are combined into two values, the aggregated positive score and the aggregated negative score. Constructs as booster words (“really”), emoticons and punctuation emphasis are crucial in this assignment of values.

In DDRE, sentiment analysis may help requirement engineers to understand the general position of the user. For instance, consider the sentences “pleeeeeeease add an unlike button and I will love you forever!!” and “uploading pictures with the app is so annoying!” [30]. While both of them are stating some dissatisfaction (asking for an additional feature the first, and complaining about a feature the second), the tone is very different and sentiment analysis will show that the first user is basically happy with the system while the second one is really complaining.

Topic modelling. This type of unsupervised analysis identifies the topics that best describe a corpus of knowledge, where each topic is a repeating pattern of co-occurring terms in such corpus, described by a probability distribution of words (i.e., the probability that a word pertains to a topic) [31].

Topic modelling is well known in information retrieval for the analysis of large documents, e.g. in order to recommend contents to readers of newspapers [32], but it has spread into software engineering in general, and RE in particular [33]. The most popular algorithm used to identify the topics and their words is Latent Dirichlet Allocation (LDA) [34], where the word “latent” means that the distribution emerges during the analysis by statistical inference. Results are not unique and in fact, one of the most challenging issues on putting LDA into action is parameterization. There are several parameters to determine: number of topics, number of words per topic, number of iterations in the LDA algorithm for convergence, and others. Another challenging property of LDA is instability meaning that it suffers from “order effects”, i.e. the output of the algorithm may depend on the order in which the terms are processed. Given these problems, other algorithms have been formulated, such as the Biterm Topic Model (BTM) [35] that models topics by exploring word-word (i.e., biterm) patterns. BTM has over-performed LDA in the context of short texts [33], which is a typical situation in explicit feedback.

Putting all together. All of the techniques above and others that are not covered in this paper (e.g., summarization [36][37]) deal with a particular NLP-related activity, but usually it is necessary to combine them in order to achieve a research goal. For instance, the ultimate goal of Guzman and Maalej’s paper cited above [30] is to identify the positive or negative sentiment that users may have with respect to app features. They used two apps as examples, Pinterest in Android and Dropbox in iOS, mined reviews in their app stores and computed the number of positive and negative reviews they found for these features. In order to get these results, they built the workflow presented in **Fig. 2**. From the user reviews, they extracted titles and comments and initiated two parallel paths. On the one hand, for each review, they applied the sentiment analysis technique outlined above (without preprocessing, in order not to eliminate for instance stop words or other elements that may convey emotions). On the other hand, they extracted the features in the review by preprocessing their contents first and then extracting fine-grained features. Last, their grouped the fine-grained features into high-level features using topic modelling, combining adequately the sentiment scores. This example is representative of the type of solutions prevalent in explicit feedback analysis.

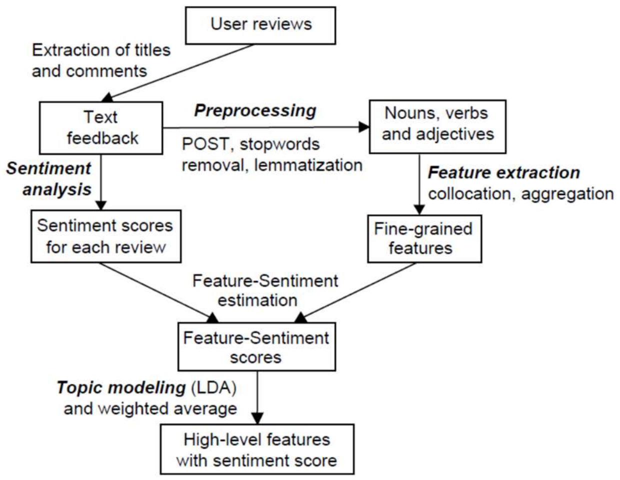
User reviews
Extraction of titles and comments
Text feedback
Preprocessing
POST, stopwords removal, lemmatization
Nouns, verbs and adjectives
Sentiment analysis
Feature extraction
collocation, aggregation
Sentiment scores for each review
Fine-grained features
Feature-Sentiment estimation
Feature-Sentiment scores
Topic modeling (LDA) and weighted average
High-level features with sentiment score

Fig. 2. Guzman and Maalej’s approach to sentiment analysis in app reviews (as appears in [30]).

It is also worth to mention the existence of a large number of libraries of components that implement some of the algorithms mentioned here, e.g. Standford CoreNLP toolkit [38][4], NLTK[5] and GenSim[6] (for similarity analysis).

4 Implicit Feedback Management

Implicit feedback is the term that denotes the feedback gathered from the system usage as it is used by their users. The main difference with explicit feedback, as said, is that data comes from the users without their explicit communication, but with their explicit consent. This unobtrusiveness is the main advantage over explicit feedback, although instrumentation is generally more complex.

The origins of implicit feedback come from the information retrieval discipline, where implicit feedback techniques are used for query expansion and user profiling in information retrieval tasks [39]. Also, the concept was heavily used in the web page ranking [40]. In software engineering, implicit feedback is having a momentum in the last years. For instance, with the advent of the Internet of Things and smart cities, sensors are continuously gathering data from users and their context. Also, big corporations and governments are going in the direction of collecting more and more information from citizens, and responses to crisis such as COVID-19 are increasing this trend [41]. In the rest of the section, we focus on the use of implicit feedback in DDRE.

4.1 Types of Implicit Feedback

The first type of implicit feedback we mention is quality of service (QoS). This is an old topic emerging in the 70s-80s in the areas of networking, real-time applications and middleware, adopted in the 2000s in the fields of service-oriented computing [42] and cloud-based systems [43], but still it plays an important role for analysing contemporary systems. QoS include attributes as response time, availability and security, which can provide very useful information to understand which parts of the system need improvement.

The second type is usage data. It may include the individual clicks of the users [44], telemetry[7], interactions with the user interface and navigational paths (clickthroughs) [45]. Usage data can be especially useful for detecting patterns of behaviour that may serve to discover unused functionalities or different ways to organize the user interface more fit to the real needs of the system users.

We mention a third type of data, non-verbal human data that can be sensed through appropriate sensors [46]. The most popular technique is eye tracking [47], but we can also mention gesture, heartrate, face muscles, etc.

[4] https://stanfordnlp.github.io/CoreNLP/
[5] https://www.nltk.org/
[6] https://pypi.org/project/gensim/
[7] https://firefox-source-docs.mozilla.org/toolkit/components/telemetry/index.html

Different types of feedback may be combined. For instance, Joachims et al. report an empirical study in which clickthroughs and eye-tracking are combined to achieve better results in web page ranking [48].

4.2 Gathering Implicit Feedback

Implicit feedback is usually stored in logs. Logs are files that contain a trace of the behaviour of the user when using the system, in the form of implicit feedback of any of the types mentioned above. Each entry in the log represents an interaction. The data fields that compose an entry are not standard but we will usually find: the timestamp, the user ID, the event type, the type of element, the URL or endpoint invoked, etc. From these logs, some typical observations are: which functionalities are most used, which navigational paths prevail (or not, even if expected), which calls result often in error codes, etc. This information may be enriched with QoS (which may require some additional monitoring infrastructure, e.g. in the case of cloud-based systems [49]). All in all, this is valuable input for understanding functionalities that are problematic, features that are missing (e.g., because the users often follow bizarre navigational paths), features that can eventually be merged (because they are always used one after the other), etc.

4.3 Importance of Context

When analysing feedback, context is utterly important. Both the response of a user and the behaviour of the system can be strongly influenced by some contextual characteristic. For instance, a system may have a good user interface as a web application but a terrible user interface in its version for mobile phones. Although context can be provided explicitly, the usual case is that it is collected as part of the system's implicit feedback, using the same gathering instruments and channels.

Context is a very wide term and includes classical concepts as time and location, but also others as user's profile and type of device for the connection. Therefore, a great amount of context ontologies has been devised especially in the field of context-aware computing and self-adaptive systems [50], which can be used as a basis to implement the concept of context in an implicit feedback gathering and analysis approach.

From the point of view of RE, there are some approaches that deal with context by defining contextual requirements [51]. A contextual requirement is a requirement whose satisfaction is guarded by a condition that represents a context. The context is operationalized as a function over a set of context variables. Each variable is sensed through one or more monitors whose values are gathered with a monitoring infrastructure. While the concept of contextual requirement is clear and intuitive, many challenges arise, as dealing with uncertainty [52] and in general, discovering unknown unknowns [53] (e.g., context conditions that are not known in advance).

4.4 Combining Explicit and Implicit Feedback

We have seen that explicit and implicit feedback are very different in nature. Explicit feedback is mostly related (still) to natural language, while (contextual) implicit feedback has to be mainly with analysing streams of data stored in logs. However, they are two different kinds of input for the main purpose: to gather feedback from the user in order to understand the actual use and acceptance of the system. The natural question that arises is: may these two types of feedback be combined into one single input? For instance, if a user provides an explicit complaint about a concrete functionality of the system, it may be useful to have available as many implicitly collected data as possible. Maybe the user entered the review in a moment where the network experienced some downtime, or maybe she was using a particular type of device that does not support well this functionality.

This mixed approach is a hot topic of investigation in the field. One example is the FAME approach [54], which implements two streams for data acquisition (see **Fig. 3**): explicit feedback using feedback forms, and runtime events in the form of logs that were captured by a monitoring infrastructure. Since implicit feedback comes as a continuous flow, FAME uses a data lake to ensure performance. Both streams are combined in a component that uses domain ontologies [55] to be able to match concepts and inform the requirements engineer in order to elicit new requirements.

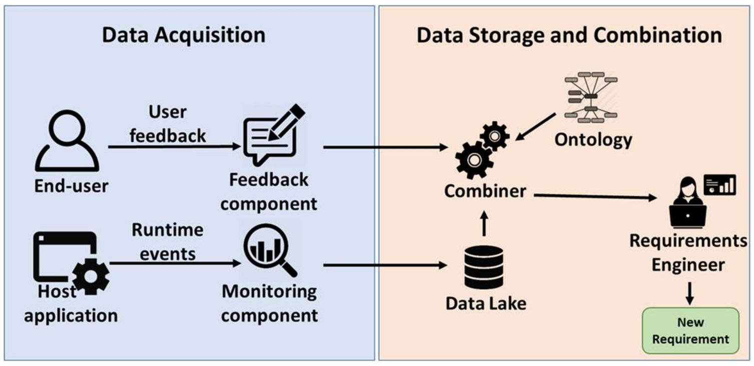

Fig. 3. The ontology-based FAME approach to integrated explicit and implicit feedback analysis (as appearing in [54]).

As a particular case on this combination of explicit and implicit feedback, we find the concept of crowd-based RE. As defined by Groen et al., crowd-based RE "is an umbrella term for automated or semiautomated approaches to gather and analyse information from a crowd to derive validated user requirements" [56]. Implicit feedback is aggregated to multi-modal explicit feedback similarly as done in the FAME approach [54].

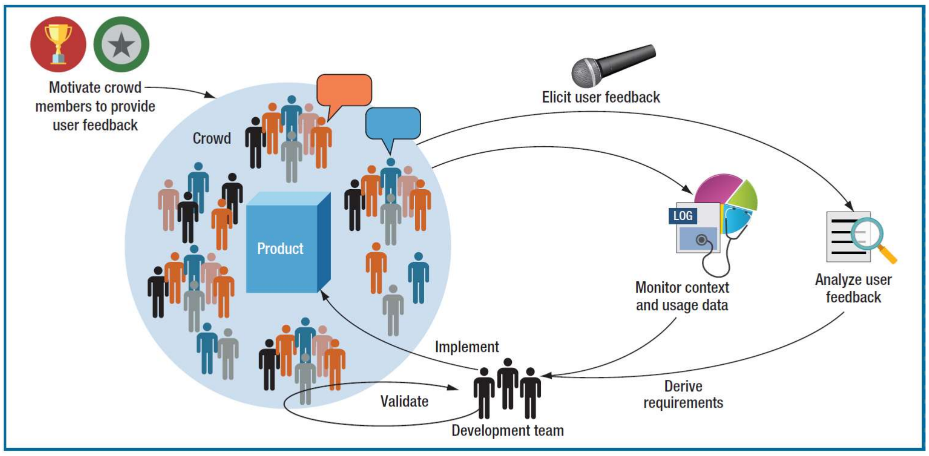

Fig. 4. Actors and their relationships in crowd-based RE (as appearing in [56]).

Other approaches to combine explicit and implicit feedback in DDRE exist. Wüest et al. follow a different strategy, in which implicit feedback is used as a trigger for explicit feedback [57]. Their argument is that this approach will engage users to provide more explicit feedback, and it will be provided when some situation uncovered by the implicit feedback requires to be analysed. It remains as a challenge to transfer these approaches into industry projects, which still rely mainly on explicit feedback [58].

5 Decision-making

As **Fig. 1** shows, once feedback is gathered and analysed, requirements engineers still need some support to make informed decisions. There are two key dimensions to decision-making in DDRE. First, to visualize the data in an actionable manner and to have at hand techniques for further analysis; to this end, requirements engineers may use software analytic tools. Second, to arrange these decisions in the form of a software release plan.

5.1 Software Analytic Tools

According to the outcome of the Dagstuhl Seminar 14261, software analytics is "to utilize data-driven approaches to obtain insightful and actionable information to help software practitioners with their data related tasks" [59]. This very generic definition accommodates a large variety of tools, some of them heavily used by the software engineering community during the development process, e.g. SonarQube[8].

Buse and Zimmermann defined several guidelines for software analytics tools, such as easiness of use and interactivity [60]. In addition, they suggested to map indicators to features, and this links well to DDRE: when applied to DDRE, the ultimate goal of

[8] https://www.sonarqube.org/

a software analytics tool is to assist in the evolution of a requirements specification. This means suggesting new requirements, or modifying existing ones (e.g., by enforcing some threshold in a quality requirement, or by changing the priority or business value of a requirement).

To obtain the aforementioned indicators, it is necessary to aggregate the data that was gathered as feedback into more elaborated attributes until we reach a level of indicator. These indicators convey actionable information for requirements engineers, e.g. product quality, time to market or business value. Usually, the aggregation is multilevel and can be conducted bottom-up using the classical concept of quality model as driver [61]. For instance, the Q-Rapids dashboard (a software analytics tool aimed at eliciting quality requirements) [62] builds upon the Quamoco approach to software quality model construction [63]. Quamoco proposes the use of utility functions and weighted sums in order to define qualitative values for these quality factors. In Q-Rapids, top-level indicators are visualized in a gauge form with three values (ok, warning and failure) depending on their distance to a given threshold.

The Q-Rapids dashboard also implements several techniques suggested by Buse and Zimmermann [60], among which we can mention:

- Visualization capabilities as drill-down navigation, from one indicator to the quality factors that compose it.
- Analysis of trends and summarization of results, to understand the direction of a software project.
- Prediction of the evolution of a particular indicator or some of the quality factors used to compute it [64].
- Definition and triggering of alerts, to report underperformance or (in combined use with prediction) to anticipate future threshold violations.
- Simulation through what-if analysis, using sliders to understand the effects of changes in factors' values over the indicators.

Oriol et al. propose to associate mitigation actions to alerts triggered by underperforming quality factors, so that these mitigation actions operate over the requirements specification [65]. The new requirements are proposed to the requirements engineer as instantiations of requirements patterns stored in a catalogue [66]. Possible instantiations of the patterns with a description of their consequences are presented to the requirements engineer through the software analytics tool. With a similar aim of identifying requirements, Dalpiaz and Parente proposed the RE-SWOT method [67]. RE-SWOT aims at eliciting requirements from app store reviews through competitor analysis. Results are presented to the requirements engineer by means of a dashboard that visualizes a Strengths-Weaknessess-Opportunities-Threats analysis of identified features.

5.2 Release Planning

The next activity in order to close the data-driven cycle is deciding how to allocate the requirements that have emerged or changed, to the next or even further system releases. The problem of software release planning is well known in software engineering [68][69] and can be stated as follows: given a set of requirements to be allocated, and a set of constraints in terms of resources, budget, and similar criteria, maximize some utility or multi-objective function and thus design a release plan, where every requirement has a release assigned (or eventually remains undecided or is even discarded) [70].

When applied to DDRE, feedback becomes the critical object that guides release planning. Having an app's release strategy is a factor that affects the ongoing success of mobile apps [71]. For instance, Villaroel et al. [72] propose a technique that processes feedback by forming clusters of related reviews (bug reports and new feature suggestions) and then prioritizes the clusters according to their: *(i)* number of reviews, *(ii)* average rating, *(iii)* difference of cluster average rating and app average rating, *(iv)* difference of ratings assigned by users who reviewed older releases of the app, and *(v)* number of different devices from which users reported reviews (which is a basic but still useful combination of explicit and implicit feedback).

Maalej et al. [7][73] mention other possible ways to adapt usual release planning approaches to DDRE characteristics: involving an increasing number of stakeholders in the process (from single person to group-based process), relying on real-time and rigorous data analytics instead of intuition, or allowing stakeholders to play a more proactive role. Several techniques to involve the right stakeholders have been proposed, mainly in relation to gamification [74], but also others as applying the concept of liquid democracy to requirements engineering [75], so that a stakeholder can nominate others to rank a requirement on her behalf when she is not knowledgeable on the requirement's facet.

6 Challenges

In this section we outline some challenges ahead for DDRE, both from research and from practical perspective.

Integration with data-oriented analysis cycles. The first challenge is to be able to integrate DDRE with existing development processes. Given the data-driven nature, companies may need to adopt some data management method, e.g. CRISP-DM [76]. CRISP-DM was formulated in the early 2000s as a cycle aimed at supporting data scientists in making data actionable through several well-defined stages: business understanding, data understanding, preparation, modelling, evaluation and deployment. Reconciling DDRE and CRISP-DM is a research direction that has been already subject of attention [77].

Integration with other software engineering approaches. Beyond integration with data-oriented approaches, DDRE needs to be integrated also in the software life cycle. DDRE is usually connected to agile approaches [78] but we can imagine integration with other processes. For instance, Franch et al. [79] explore the integration of DDRE into a model-driven development software life cycle, where the generation of the feedback gathering infrastructure is integrated with the generation of the system itself, supporting thus evolution of the embedded mechanisms as the system requirements evolve.

User motivation and trust. While implicit feedback mainly depends on the availability of a feedback gathering infrastructure (although of course privacy concerns are also a challenge to consider), explicit feedback requires the engagement of users. Gamification is the most usual strategy to motivate users to provide explicit feedback [80], although it has shown mixed results in this field; instead, tangible incentives can be more attractive to users (e.g., in the gaming domain, alpha/beta players get early versions of games). A problem in the opposite direction to having scarce data is the reliability of the explicit feedback given. Especially in the app market, the fight against fake reviews has become critical. A recent empirical study by Martens and Maalej analysed thoroughly the business behind fake reviews, ultimately revealing their significant impact [81].

Context-driven feedback gathering. To make any feedback management approach fully contextual, the feedback gathering instruments themselves need to adapt to context. For instance, the frequency of monitoring can decrease when the device executing the system is running out of battery, or on the contrary, it can be increased when it comes to monitoring the behaviour of a problematic feature. Approaches for context-driven implicit feedback gathering exist in the form of adaptive monitoring infrastructures [82]. For explicit feedback, Almaliki et al. use the concept of *Persona* [83] to gather explicit feedback depending on the profile of the user [84].

Analysis of implicit feedback. The analysis of usage logs faces several recurrent challenges. First, data is usually noisy, both in terms of log entries that are useless, and fields that are not useful for the purpose of DDRE. Second, data is often incomplete, so that fields may be missing or may be too coarse-grained to be useful. Third, the concept of session is not always evident. Sessions are useful because they represent chunks of work of a user, and it is convenient to delimit them in the log files. Fourth, as applications evolve, so should do log files, but then it is difficult to analyse them along time. Fifth, data collection should be non-intrusive and as minimal as possible according to the objectives sought. Last but not least, anonymization is critical and required by law.

Use of domain knowledge. While DDRE is based on analysing as much data as possible, it remains a challenge to investigate whether it can be effectively leveraged using domain knowledge. For instance, we have already mentioned in Section 4.4 the use of domain ontologies for matching explicit and implicit feedback concepts [54]. If we look into the details, this domain ontology bridges both worlds through some connecting concepts, for instance *TimeStamp*, *User* and *Application*.

Adoption by companies. In the addition to the research-oriented challenges above, other more practical barriers emerge for companies to adopt DDRE [62][85]. We may classify them into three categories:

- **Organizational**. First, the general concept of DDRE needs to be tailored to every company. For instance, the indicators to be used for decision-making will be ultimately determined by both the business priorities of the company and the availability of data. Second, integration with the company way of working, since it cannot be expected that a company will completely change its current processes and practices. Last, aligning the vocabulary, which may seem not so important at a first glance, but it becomes an important impediment, especially when it comes to discuss about particular types of quality requirements
- **Value-related**. Two complementary challenges are: providing explanations (informative dashboards and generation of reports, for instance), and transparency, allowing decision-makers to drill from recommendations in terms of requirements or decisions, down to data.
- **Technological**. Given that it is necessary to implement a software infrastructure to gather, analyse and decide upon feedback, the technology needs to be as less invasive as possible, simplifying the installation of the tool and making efficient its configuration.

7 Discussion

7.1 Related areas

While DDRE is a recent research direction, it is clear from this guided tour that it benefits from consolidated results produced in other software engineering areas that exist for many years, some of them already mentioned. We highlight:

- **User-centred design**. This area emerged in the 70s [86] and originated the concept of user feedback. For instance, as early as in 1971, Hansen reported the following in the design of a text editor called Emily: “A log is kept of all user interactions, user errors, and system errors. There is a command to let the user type a message to be put in the log and this message is followed by a row of asterisks. When the user is frustrated he can push a 'sympathy' button. In response, Emily displays at random one of ten sympathetic messages. More importantly, frustration is noted in the log and the system designer can examine the user's preceding actions to find out where his understanding differed from the system implementation” [87].
- **Process mining**. The area of process mining appeared in the mid-90s under the label of process discovery, and more related to software process and workflow technologies [88][89]. These approaches used event-based logs to capture the actions that occurred during the execution of the process [90]. In the early 2000s, there was a shift of focus into business process and information systems [91] and service design [92]. Techniques appearing in these areas [93] can be used in DDRE.
- **Mining software repositories**. Software repositories, such as issue and bug tracking systems and project management tools, are a valuable source of information that can be used to understand software development practices and uncover software quality issues [94]. It is used for many purposes like fault prediction, productivity analysis, impact analysis and in general, product and process dynamics [95]. In the last years,

given the large amounts of data to be processed and the complexity to analyze them, there is a corpus of knowledge delivering increasingly sophisticated analysis solutions [96]. Connection to DDRE appears especially when considering software repositories' data as a possible source for software quality defects, which can eventually generate internal quality requirements (related to maintainability, portability, etc.) [61].
- **Service monitoring**. With the advent of service-oriented computing in the early 2000s [97], one of the areas of research was that of service monitoring. Oriol et al. surveyed the different areas in which monitoring is important, and remarkably monitoring quality of service is one of them [98]. An ample body of research on this topic appeared, with special emphasis on using the monitoring infrastructure for checking service level agreements [99] and even suggesting explanations or repair rules when those agreements were violated [100].
- **Requirements monitoring**. The concept of requirements monitoring emerged in the mid-90s [101] and is still present in the RE area [102]. Requirements monitoring has been frequently used over goal-oriented models, for instance in the context of self-adaptive systems [103] and obstacle resolution [104]. This concept can also be connected with DDRE, by considering requirements monitoring as part of the software analytics activities, and especially focusing on monitoring of user requirements [105].

It is also worth to mention the relationship of DDRE with the topic of online controlled experimentation [106]. This approach to software product development proposes to collect data from users based in two competing versions that differ in a particular feature [107]. It is a generalization of the concept of A/B testing and shares several principles with DDRE, as the need of quality data and the convenience to establish necessary competencies [108].

Other areas not part of software engineering, as information retrieval or linguistics, have also had an influence to DDRE, as shown in this paper.

7.2 Lessons Learned

The concepts presented in this paper have been applied in the last years in several EU collaborative projects (Q-Rapids [109], SUPERSEDE [110] and OpenReq [111]) where large corporations and small-medium enterprises have adopted DDRE at some extent. Some observations arising from these projects follow:

- DDRE is not for free. Adopting DDRE requires both adapting organizational processes and mindset and developing some infrastructure in order to make it happen.
- DDRE is different for every company. There are not two companies adopting the proposed DDRE approach in the same way. This means that DDRE needs to define general process with high customization capabilities. Situational method engineering [112] can be helpful in dealing with such diversity, as we have explored in one of these projects, SUPERSEDE [113].
- DDRE requires expertise in terms of specialized roles, e.g. data scientists accompanying the requirements engineers and software engineers.

- DDRE needs to be implemented in an incremental way, in order to gradually master its intricacies and complexities, and create awareness in the organization.
- DDRE requires full transparency. Decisions made during data-informed software analysis need to be clearly justified [114] and with a rationale behind such that the requirements engineer can understand the decision and then accept or decline, or elaborate further.

8 Conclusions

In this paper, we have presented a guided tour to data-driven requirements engineering (DDRE). The main message is that DDRE offers a great opportunity for delivering more business value to systems' stakeholders by evolving the system according to the real needs elicited through the analysis of the gathered feedback and tool-supported decision-making. In line with Ebert et al. [77], we can say that those companies that do not consider data from system usage in their development processes are increasingly putting themselves at competitive disadvantage.

DDRE changes the focus of traditional requirements engineering from human-oriented to data-oriented, although this does not mean that it can replace completely the existing requirements engineering management approaches and in particular, data will still need to be analysed and validated by humans. First, data (mainly represented by feedback) is agnostic per se, and therefore interpretation by humans is still necessary, even if machine learning techniques are adopted. This is why we have presented the decision-making process as tool-supported but not as automatic. Second, in order to gather feedback, an initial system needs to be made available to users. Requirements for this minimal viable product need to be gather without data, i.e. using traditional requirement elicitation and prioritization techniques [1].

Considering the paragraph above, and the challenges and lessons learned enumerated in previous sections, we can conclude that it may not be appropriate to blindly adopt DDRE in the context of some companies or systems. At the end, DDRE is another approach that composes the requirements engineer toolbox, adding some new activities to those that are more traditional [115], to be used wisely in the right moment with the right customization.

Acknowledgment

This work is partially supported by the GENESIS project, funded by the Spanish Ministerio de Ciencia e Innovación under contract TIN2016-79269-R. The author wants to deeply thank Fabiano Dalpiaz, Silverio Martínez-Fernández and Marc Oriol for their comments and suggestions over a first draft of the paper.

References

1. Pohl, K.: *Requirements Engineering: Fundamentals, Principles and Techniques*. Springer (2010).
2. Ross, D.T. (ed): Special Collection on Requirement Analysis. *IEEE Transactions on Software Engineering*, SE-3(1) (1977).
3. Boehm, B.: Software Engineering. *IEEE Transactions on Computers*, C-25 (12): 1226–1241 (1976).
4. Kuffel, W.: Extra Time Saves Money. *Computer Language* (1990).
5. Spinellis, D.: *Code Quality – The Open Source Perspective*. Pearson (2006).
6. PMI: Pulse of the Profession® In-Depth Report: Requirements Management — A Core Competency for Project and Program Success. https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/requirements-management.pdf (2014).
7. Maalej, W. Nayebi, M., Johann, T., Ruhe, G.: Toward Data-Driven Requirements Engineering. *IEEE Software* 33(1): 48–54 (2016).
8. Lucas, H.C.: A User-oriented Approach to Systems Design. In Proceedings of the 26th Annual Conference of the Association for Computing Machinery (ACM): 325–338, ACM Press (1971).
9. Trotter, P.: User Feedback and How to Get it. *In Proceedings of the 4th Annual Conference on User Services* (SIGUCCS): 130–132, ACM Press (1976).
10. Pagano, D., Maalej, W.: User Feedback in the Appstore: An Empirical Study. In *Proceedings of the 21st International Requirements Engineering Conference* (RE): 125–134, IEEE Press (2013).
11. Guzmán, L., Oriol, M., Rodríguez, P., Franch, X., Jedlitschka, J., Oivo, M.: How Can Quality Awareness Support Rapid Software Development? - A Research Preview. In *Proceedings of the 23rd International Working Conference on Requirements Engineering: Foundation for Software Quality* (REFSQ), LNCS 10153: 167–173. Springer (2017).
12. Franch, X., Ayala, C., López, L., Martínez-Fernández, S., Rodríguez, P., Gómez, C., Jedlitschka, A., Oivo, M., Partanen, J., Raty, T., Rytivaara, V.: Data-Driven Requirements Engineering in Agile Projects: The Q-Rapids Approach. In *Proceedings of the 25th International Requirements Engineering Conference Workshops* (REW): 411–414, IEEE Computer Society (2017).
13. Fitzgerald, B., Stol, K.J.: Continuous Software Engineering: A Roadmap and Agenda. *Journal of Systems and Software* 123: 176–189 (2017).
14. Hosseini, M., Groen, E.C., Shahri, A., Ali, R.: CRAFT: A Crowd-Annotated Feedback Technique. In *Proceedings of the IEEE 25th International Requirements Engineering Conference Workshops* (REW): 170–175 (2017).
15. Chowdhury, G.: Natural Language Processing. *Annual Review of Information Science and Technolog*y, 37: 51–89 (2003).
16. Zhao, L., Alhoshan, W., Ferrari, A., J. Letsholo, K.J. Ajagbe, M.A. Chioasca, E.-V., Batista-Navarro, R.T.: Natural Language Processing (NLP) for Requirements Engineering: A Systematic Mapping Study. arXiv:2004.01099v2 [cs.SE] (2020).
17. Dalpiaz, F., Ferrari, A., Franch, X., Palomares, C.: Natural Language Processing for Requirements Engineering; The Best Is Yet to Come. *IEEE Software* 35(5), pp. 115–119 (2018).
18. El Shawi, R., Maher, M., Sakr, S.: Automated Machine Learning: State-of-the-Art and Open Challenges. arXiv:1906.02287v2 [cs.LG] (2019).

19. Webster, J.J., Kit, C.: Tokenization as the Initial Phase in NLP. In *Proceedings of the 14th Conference on Computational Linguistics* (COLING) - Volume 4: 1106–1110, ACM Press (1992).
20. Ladani, D.J., Desai, N.P.: Stopword Identification and Removal Techniques on TC and IR Applications: A Survey. In *Proceedings of the 6th International Conference on Advanced Computing and Communication Systems* (ICACCS): 466–472, IEEE Press (2020).
21. Singh, J., Gupta, V.: A Systematic Review of Text Stemming Techniques. *Artificial Intelligence Review* 48**:** 157–217, Springer (2017).
22. Balakrishnan, V., Lloyd-Yemoh, E.: Stemming and Lemmatization: A Comparison of Retrieval performances. *Lecture Notes on Software Engineering* 2(3): 262–267 (2014).
23. Abney, S.: Part-of-Speech Tagging and Partial Parsing. In *Corpus-Based Methods in Language and Speech Processing. Text, Speech and Language Technology*, vol 2: 118–136, Springer (1997)
24. Morales-Ramirez, I., Kifetew, F.M., Perini, A.: Speech-acts based Analysis for Requirements Discovery from Online Discussions. *Information Systems* 86: 94–112 (2019).
25. Searle, J.R.: *Speech Acts: An Essay in the Philosophy of Language*. Cambridge University Press (1969).
26. Guzman, E., Alkadhi, R., Seyff, N.: A Needle in a Haystack: What do Twitter Users Say about Software? In *Proceedings of the 24th International Requirements Engineering Conference* (RE): 96–105, IEEE Computer Society (2016).
27. Nasukawa, T., Yi, J.: Sentiment Analysis: Capturing Favorability using Natural Language Processing. In *Proceedings of the 2nd international Conference on Knowledge Capture* (K-CAP): 70–77, ACM Press (2003).
28. Taboada, M., Brooke, J., Tofiloski, M., Voll, K., Stede, M.: Lexicon-Based Methods for Sentiment Analysis. *Computational Linguistics* 37(2): 267–307 (2011).
29. Zhang, L., Wang, S., Liu, B.: Deep Learning for Sentiment Analysis: A Survey. *Data Mining and Knowledge Discovery* 8(4), Wiley (2018).
30. Guzman, E., Maalej, W.: How do Users like this Feature? A Fine Grained Sentiment Analysis of App Reviews. In *Proceedings of the 22nd International Requirements Engineering Conference* (RE): 153–162, IEEE Computer Society (2014).
31. Wallach, H.M.: Topic Modeling: Beyond Bag-of-Words. In *Proceedings of the 23rd International Conference on Machine Learning* (ICML): 977–984, ACM Press (2006).
32. Jacobi, C., van Atteveldt, W., Welbers, K.: Quantitative Analysis of Large Amounts of Journalistic Texts using Topic Modelling. *Digital Journalism* 4(1): 89–106, Taylor&Francis (2016).
33. Abad, Z.S.H., Karras, O., Ghazi, P., Glinz, M., Ruhe, G., Schneider, K.: What Works Better? A Study of Classifying Requirements. arXiv:1707.02358 [cs.SE] (2017).
34. Blei, D.M., Ng, A.Y., Jordan, M.I.: Latent Dirichlet Allocation. *Journal of Machine Learning Research* 3: 993–1022, The MIT Press (2003).
35. Yan, X., Guo, J., Lan, Y., Cheng, X.: A Biterm Topic Model for Short Texts. In *Proceedings of the 22nd International Conference on World Wide Web* (WWW): 1445–1456, ACM press (2013).
36. Nenkova A., McKeown K.: A Survey of Text Summarization Techniques. In: Aggarwal C., Zhai C. (eds.) *Mining Text Data*: 43–76, Springer (2012).
37. Allahyari, M., Pouriyeh, S., Assefi, M., Safaei, S., Trippe, E.D., Gutierrez, J.B., Kochut, K.: Text Summarization Techniques: A Brief Survey. arXiv:1707.02268v3 [cs.CL] (2017).
38. Manning, C.D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S.J., McClosky, D.: The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the 52nd Annual*

Meeting of the Association for Computational Linguistics: System Demonstrations (ACL): 55–60 (2014).

39. Kelly, D., Teevan, J.: Implicit Feedback for Inferring User Preference: A Bibliography. *ACM SIGIR Forum* 37(2): 18–28, ACM Press (2003).
40. Agichtein, E., Brill, E., Dumais, S.: Improving Web Search Ranking by Incorporating User Behavior Information. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR): 19–26, ACM Press (2006).
41. Carvalho, V.M., Hansen, S., Ortiz, A., Garcia, J.R., Rodrigo, T., Rodriguez Mora, S., Ruiz de Aguirre, P.: Tracking the Covid-19 Crisis with High-Resolution Transaction Data. CEPR Discussion Paper No. DP14642 (2020).
42. Papazoglou, M.P., Georgakopoulos, D.: Introduction: Service-Oriented Computing. *Commununications of the ACM* 46(1), 24–28 (2003).
43. Abdelmaboud, A., Jawawi, D.N.A., Ghani, I., Elsafi, A., Kitchenham, B.: Quality of Service Approaches in Cloud Computing: A Systematic Mapping Study. *Journal of Systems and Software* 101: 159–179 (2015).
44. Janes, A.: Non-distracting, Continuous Collection of Software Development Process Data. In: Nalepa G., Baumeister J. (eds) *Synergies Between Knowledge Engineering and Software Engineering*, AISC 626. Springer (2018).
45. Joachims. T.: Optimizing Search Engines using Clickthrough Data. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD): 133–142, ACM Press (2002).
46. Harrigan, J., Rosenthal, R., Scherer, K. (eds.): *The New Handbook of Methods in Nonverbal Behavior Research*. Oxford University Press (2005).
47. Sharafi, Z., Soh, Z., Guéhéneuc, Y.-G.: A Systematic Literature Review on the Usage of Eye-Tracking in Software Engineering. *Information and Software Technology* 67: 79–107 (2015).
48. Joachims, T., Granka, L., Pan, B., Hembrooke, H., Gay, G.: Accurately Interpreting Clickthrough Data as Implicit Feedback. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR): 154–161, ACM Press (2005)
49. Kertesz, A., Kecskemeti, G., Oriol, M., Kotcauer, P., Acs, S., Rodríguez, M., Mercè, Marosi, A.Cs., Marco, J., Franch, X.: Enhancing Federated Cloud Management with an Integrated Service Monitoring Approach. *Journal of Grid Computing* 11(4): 699–720 (2013).
50. Cabrera, O., Franch, X., Marco, J.: Ontology-based Context Modeling in Service-oriented Computing: A Systematic Mapping. *Data & Knowledge Engineering* 110: 24–53 (2017).
51. Ali, R., Dalpiaz, F., Giorgini, P.: Reasoning with Contextual Requirements: Detecting Inconsistency and Conflicts. *Information and Software Technology* 55: 35–57 (2013).
52. Knauss, A., Damian, D.E., Franch, X., Rook, A., Müller, H.A., Thomo, A.: ACon: A Learning-based Approach to deal with Uncertainty in Contextual Requirements at Runtime. *Information and Software Technology* 70: 85–99 (2016).
53. Sutcliffe, A., Sawyer, P.: Requirements Elicitation: Towards the Unknown Unknowns. In *Proceedings of the 21st International Requirements Engineering Conference* (RE): 92–104, IEEE Press (2013).
54. Oriol, M., Stade, M.J.C., Fotrousi, F., Nadal, S., Varga, J., Seyff, N., Abelló, A., Franch, X., Marco, J., Schmidt, O.: FAME: Supporting Continuous Requirements Elicitation by Combining User Feedback and Monitoring. In *Proceedings of the 26th International Requirements Engineering Conference* (RE): 217–227, IEEE Computer Society (2018).

55. McDaniel, M., Storey, V.C.: Evaluating Domain Ontologies: Clarification, Classification, and Challenges. *ACM Computing Surveys*. 52(4), Article 70 (2019).
56. Groen, E.C., Seyff, N., Ali, R., Dalpiaz, F., Doerr, J., Guzman, E., Hosseini, M., Marco, J., Oriol, M., Perini, A., Stade, M.: The Crowd in Requirements Engineering: The Landscape and Challenges. *IEEE Software* 34(2): 44–52 (2017)
57. Wüest, D., Fotrousi, F., Fricker, S.: Combining Monitoring and Autonomous Feedback Requests to Elicit Actionable Knowledge of System Use. In: *Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality* (REFSQ): 209–225, LNCS 11412 (2019).
58. Johanssen, J.O., Kleebaum, A., Bruegge, B., Paech, B.: How do Practitioners Capture and Utilize User Feedback during Continuous Software Engineering? In *Proceedings of the 27th International Requirements Engineering Conference* (RE): 153–164, IEEE Press (2019).
59. Gall, H., Menzies, T., Williams, L., Zimmermann, T. (eds.): Software Development Analytics. *Dagstuhl Reports* 4(6): 64–83 (2014).
60. Buse, R.P.L., Zimmermann, T.: Information Needs for Software Development Analytics. In *Proceedings of the 34th International Conference on Software Engineering* (ICSE). IEEE Press, 987–996 (2012).
61. The ISO Organization: *ISO/IEC 25010:2011 –Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*. (2011).
62. Martínez-Fernández, S., Vollmer, A.-M., Jedlitschka, A., Franch, X., López, L., Ram, P., Rodríguez Marín, P., Aaramaa, S., Bagnato, A., Choras, M., Partanen, J.: Continuously Assessing and Improving Software Quality with Software Analytics Tools: A Case Study. *IEEE Access* 7: 68219–68239 (2019).
63. Wagner, S., Goeb, A., Heinemann, L., Kläs, M., Lampasona, C., Lochmann, K., Mayr, A., Plösch, R., Seidl, A., Streit, J., Trendowicz, A.: Operationalised Product Quality Models and Assessment: The Quamoco Approach. *Information and Software Technology* 62: 101–123 (2015).
64. Choraś, M., Kozik, R., Pawlicki, M., Hołubowicz, W., Franch, X.: Software Development Metrics Prediction using Time Series Methods. In *Proceedings of the 18th IFIP International Conference on Computer Information Systems and Industrial Management* (CISIM), LNCS 11703. Springer (2019).
65. Oriol, M., Martínez-Fernández, S., Behutiye, W., Farré, C., Kozik, R., Seppänen, P., Vollmer, A.M., Rodríguez, P., Franch, X., Aaramaa, S., Abhervé, A., Choraś, Partanen, J.: Data-driven and Tool-supported Elicitation of Quality Requirements in Agile Companies. *Software Quality Journal* (in press) (2020). https://doi.org/10.1007/s11219-020-09509-y.
66. Renault, S., Mendez-Bonilla, O., Franch, X., Quer, C.: PABRE: Pattern-based Requirements Elicitation. In *Proceedings of the 3rd International Conference on Research Challenges in Information Science* (RCIS): 81–92, IEEE Press (2009).
67. Dalpiaz, F., Parente, M.: RE-SWOT: From User Feedback to Requirements via Competitor Analysis. In *Proceedings of the 25th International Working Conference on Requirements Engineering: Foundation for Software Quality* (REFSQ): 55–70, LNCS 11412, Springer (2019).
68. Svahnberg, M., Gorschek, T., Feldt, R. Torkar, R., Saleem, S.B., Shafique, M.U.: A Systematic Review on Strategic Release Planning Models. *Information and Software Technology* 52(3): 237–248 (2010).
69. Ameller, D., Farré, C., Franch, X., Rufian, G.: A Survey on Software Release Planning Models. In *Proceedings of the 17th International Conference on Product-Focused Software Process Improvement* (PROFES), LNCS 10027, Springer (2016).

70. Greer, D., Ruhe, G.: Software Release Planning: An Evolutionary and Iterative Approach. *Information and Software Technology* 46(4): 243–253 (2004).
71. Nayebi, M., Adams, B., Ruhe, G.: Release Practices for Mobile Apps -- What do Users and Developers Think? In *Proceedings of the* 23rd *International Conference on Software Analysis, Evolution, and Reengineering* (SANER)*:* 552–562 (2016).
72. Villarroel, L., Bavota, G., Russo, B., Oliveto, R., di Penta, M.: Release Planning of Mobile Apps based on User Reviews. In *Proceedings of the 38th International Conference on Software Engineering* (ICSE): 14–24, IEEE Computer Society (2016).
73. Maalej, W., Nayebi, M., Ruhe, G.: Data-Driven Requirements Engineering - An Update. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice* (ICSE-SEIP): 289–290 (2019).
74. Kifetew F.M., Munante, D., Perini, A., Susi, A., Siena, A., Busetta, P., Valerio, D.: Gamifying Collaborative Prioritization: Does Pointsification Work? In *Proceedings of the 25th International Requirements Engineering Conference* (RE): 322–331, IEEE Press (2017).
75. Johann, T., Maalej, W.: Democratic Mass Participation of Users in Requirements Engineering? In *Proceedings of the 23rd International Requirements Engineering Conference* (RE): 256–261, IEEE Press (2015).
76. Shearer, C.: The CRISP-DM Model: The New Blueprint for Data Mining. *Journal of Data Warehousing* 4(5): 13–22 (2000).
77. Ebert, C., Heidrich, J., Martinez-Fernandez, S., Trendowicz, A.: Data Science: Technologies for Better Software. *IEEE Software* 36(6): 66–72 (2019).
78. Svensson, R.B., Feldt, R., Torkar, R.: The Unfulfilled Potential of Data-Driven Decision Making in Agile Software Development. In *Agile Processes in Software Engineering and Extreme Programming – Proceedings of the 20th International Conference on Agile Software Development* (XP): 69–85, LNBIP 355. Springer (2019).
79. Franch, X., Seyff, N., Oriol, M., Fricker, S., Groher, I., Vierhauser, M., Wimmer, M.: Towards Integrating Data-Driven Requirements Engineering into the Software Development Process: A Vision Paper. In *Proceedings of the 26th International Working Conference on Requirements Engineering: Foundation for Software Quality* (REFSQ): 135–142, LNCS 12045, Springer (2020).
80. Dalpiaz, F., Snijders, R., Brinkkemper, S., Hosseini, M., Shahri, A., Ali, R.: Engaging the Crowd of Stakeholders in Requirements Engineering via Gamification. In *Gamification – Using Game Elements in Serious Contexts*: 123–135, PROIS, Springer (2017).
81. Martens, D., Maalej, W.: Towards Detecting and Understanding Fake Reviews in App Stores. *Empirical Software Engineering* 24: 3316–3355, Springer (2019).
82. Zavala, E., Franch, X. Marco, J.: Adaptive Monitoring: A Systematic Mapping. *Information and Software Technology* 105: 161–189 (2019).
83. Pruitt, J., Grudin, J.: Personas: Practice and Theory. In *Proceedings of the 2003 Conference on Designing for User Experiences* (DUX): 1–15, ACM Press (2003).
84. Almaliki, M., Ncube, C., Ali, R.: Adaptive Software-based Feedback Acquisition: A Persona-based Design. In *Proceedings of the 9th International Conference on Research Challenges in Information Science* (RCIS): 100–111, IEEE Press (2015).
85. Choras, M., Springer, T., Kozik, R., López, L., Martínez-Fernández, S., Ram, P., Rodríguez Marín, P., Franch, X.: Measuring and Improving Agile Processes in a Small-Size Software Development Company. *IEEE Access* 8: 78452–78466 (2020).
86. Kling, R.: The Organizational Context of User-Centered Software Designs. *MIS Quarterly*, 1(4): 41–52 (1977).
87. Hansen, W.J.: User Engineering Principles for Interactive Systems. In *Proceedings of the Fall Joint Computer Conference* (AFIPS): 523–532, ACM Press (1971).

88. Cook, J.E., Wolf, A.L.: Automating Process Discovery through Event-Data Analysis. In *Proceedings of the 17th International Conference on Software Engineering* (ICSE): 73–82, IEEE Press (1995).
89. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In *Proceedings of the 6th International Conference on Extending Database Technology* (EDBT): 467–483, LNCS 1377, Springer (1998).
90. Wolf, A.L., Rosenblum, D.S.: A Study in Software Process Data Capture and Analysis. In *Proceedings of the 2nd International Conference on the Software Process-Continuous Software Process Improvement* (SPCON): 115–124, IEEE Press (1993).
91. van der Aalst, W.: *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer (2011).
92. van der Aalst, W.: Service Mining: Using Process Mining to Discover, Check, and Improve Service Behavior. *IEEE Transactions on Services Computing* 6(4): 525–535 (2013).
93. Garcia, C.D.S., Meincheim, A., Faria Junior, E.R., Dallagassa, M.R., Sato, D.M.V., Carvalho, D.R., Portela Santos, E.A:, Scalabrin, E.E.: Process Mining techniques and Applications – A Systematic Mapping Study. *Expert Systems with Applications* 133: 260–295 (2019).
94. Hassan, A.E.: Mining Software Repositories to Assist Developers and Support Managers. In *Proceedings of the 22nd IEEE International Conference on Software Maintenance* (ICSM): 339-342, IEEE Press (2006).
95. Kagdi, H., Collard, M.L., Maletic, J.I.: A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution. *Journal of Software Evolution and Process* 19(2): 77–131 (2007).
96. Bird, C., Menzies, T., Zimmermann, T.: *The Art and Science of Analyzing Software Data*. Elsevier (2016).
97. Papazoglou, M.P., Georgakopoulos. D.: Introduction: Service-Oriented Computing. *Communications of the ACM* 46(10): 24–28 (2003).
98. Oriol, M., Franch, X., Marco, J.: Monitoring the Service-based System Lifecycle with SALMon. *Expert Systems with Applications* 42(19): 6507–6521 (2015).
99. Comuzzi, M., Kotsokalis, C., Spanoudakis, G., Yahyapour, R.: Establishing and Monitoring SLAs in Complex Service Based Systems. In *Proceedings of the 2009 IEEE International Conference on Web Services* (ICWS): 783–790, IEEE Press (2009).
100. Müller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruíz-Cortés, A., Rodríguez, M.: Comprehensive Explanation of SLA Violations at Runtime. *IEEE Transactions on Services Computing* 7(2): 168–183 (2014).
101. Fickas, S., Feather, M.S.: Requirements Monitoring in Dynamic Environments. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering* (ISRE): 140–147, IEEE Press (1995)
102. Vierhauser, M, Rabiser, R, Grünbacher, P.: Requirements Monitoring Frameworks: A Systematic Review. *Information and Software Technology* 80: 89–109 (2016).
103. Oriol, M., Qureshi, N.A., Franch, X., Perini, A., Marco, J.: Requirements Monitoring for Adaptive Service-Based Applications. In Proceedings of the 18th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ): 280–287, LNCS 7195, Springer (2012).
104. Cailliau, A., van Lamsweerde, A.: Runtime Monitoring and Resolution of Probabilistic Obstacles to System Goals. *ACM Transactions on Autonomous and Adaptive Systems* 14(1): Article 3 (2019).
105. Robinson, W.N.: Seeking Quality through User-Goal Monitoring. *IEEE Software* 26(5): 58–65 (2009).

106. Kohavi, R., Deng, A., Frasca, B., Walker, T., Xu, Y., Pohlmann, N.: Online Controlled Experiments at Large Scale. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (KDD): 1168–1176, ACM Press (2013).
107. Fabijan, A., Dmitriev, P., McFarland, C., Vermeer, L., Holmström Olsson, H., Bosch, J.: Experimentation Growth: Evolving Trustworthy A/B Testing Capabilities in Online Software Companies. *Journal of Software: Evolution and Processes* 30: e2113, Wiley (2018).
108. Lindgren, E., Münch, J.: Raising the Odds of Success: The Current State of Experimentation in Product Development. *Information and Software Technology* 77: 80–91 (2016).
109. Franch, X., Lopez, L., Martínez-Fernández, S., Oriol, M., Rodríguez, P., Trendowicz, A.: Quality-Aware Rapid Software Development Project: The Q-Rapids Project. In *Proceedings of the 51st International Conference on Objects, Components, Methods and Patterns* (TOOLS): 378–392, LNCS 11771, Springer (2019).
110. Perini, A.: Data-Driven Requirements Engineering. The SUPERSEDE Way. In *Proceedings of the 5th Annual International Symposium on Information Management and Big Data* (SIMBig): 13–18, CCIS 898. Springer (2019).
111. Felfernig, A., Stetinger, M., Falkner, A., Atas, M., Franch, X., Palomares, C.: OpenReq: Recommender Systems in Requirements Engineering. In *Proceedings of the International Workshop on Recommender Systems and Social Network Analysis* (RS-SNA): 1–4, CEUR 2025 (2017).
112. Henderson-Sellers, B., Ralyté, J. Ågerfalk, P., Rossi, M.: *Situational Method Engineering*. Springer (2014).
113. Franch, X., Ralyté, J., Perini, A., Abelló, A., Ameller, D., Gorroñogoitia, J., Nadal, S., Oriol, M., Seyff, N., Siena, A., Susi, A.: A Situational Approach for the Definition and Tailoring of a Data-Driven Software Evolution Method. In *Proceedings of the 30th International Conference on Advanced Information Systems Engineering* (CAiSE): 603–618, LNCS 10816, Springer (2018).
114. Dam, H.K., Tran, T., Ghose, A.: Explainable Software Analytics. In *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results* (ICSE-NIER): 53–56, ACM Press (2018).
115. Franch, X., Palomares, C., Gorschek, T.: On the Requirements Engineer Role. *Communications of the ACM* (in press), http://dx.doi.org/10.1145/3418292.