# Software Requirements: Past Present and Future

Shaun Kinsella[1], James Nolan[1], Warren Purcell[1] and Mikhail Arkhangelskiy[1]

[1] Dublin City University
shaun.kinsella38@mail.dcu.ie

**Abstract.** Software requirements engineering is a challenging, complex area and if misguided, can make or break a software project. In this paper, we aim to document requirements engineering from the not so distant past, to the techniques used in the present day. We document the early core principles, established in the 1990s, which are still relevant today. Examining the move from traditional methods to Agile, which acted as a catalyst in creating more areas of innovation within the field of requirements engineering. We take a look at today's requirements engineering issues and their candidate solutions. Further, we examine possible trends of note for the future to provide a broad timeline of how requirements engineering has evolved.

**Keywords:** Software Requirements, Requirements Elicitation, Requirements Validation, Requirements Automation, Agile, Requirements Engineering.

## 1 Introduction

Software requirements engineering (RE) has always been and will continue to be a difficult, important and necessary part of quality software development. Organisations that lack a well-planned, orderly and repeatable software development methodology will be unable to adequately meet commitments and be constantly operating in crisis mode [1].

As argued in Brooks "87 [2], if done erroneously, requirements engineering has the potential to cripple the expected software product and requires great effort and cost to rectify. Therefore, it is critical for organisations to establish a well-planned, orderly and repeatable requirements engineering methodology. Reviewing requirements specifications early is an effective and efficient way to ensure quality software development. [3] However, no single technique adequately addresses all of the problem areas that exist in software requirements engineering. [4]

Motivated by this observation, we explore the evolution of software requirements engineering techniques. From the not so distant past, to the techniques used in the present day. Ultimately, our objective is to document and discuss how requirements engineering has developed from a small area of research to the enormous and important area of research that it is today.

This paper is organised as follows. We conduct analysis under four main areas. In "Foundations of today's requirements", we discuss how the RE principles established

in the 1990s are the foundation of RE today. We also discuss the radical views that emerged which changed the RE research area.

We contrast the traditional RE methods with Agile methods in "The move from traditional to modern requirements" and address why this move happened whilst discussing the methodologies issues.

Under "The issues with today's requirements" we discuss the shortfalls of both Trad RE and Agile RE in regards to ambiguity with NLP, non-functional requirements and traceability of requirement. We contrast possible solutions, frameworks and tools for dealing with RE issues.

In "Move towards the future" we discuss the emergence of new requirement gathering techniques and the trend toward the automation of RE. The analysis is then followed by "limitations of research", "areas of future research" and the "conclusion".

## 2 Related Literature

Taking example from two previous studies in their methodology we carried out our search as follows [5-6] In order to gather the necessary data for our research paper, we generated a rule set for the inclusion and exclusion of candidate papers. This allowed us to quickly filter out irrelevant papers without committing our time to reading them fully.

Inclusion set:

1. The paper is written in English
2. The paper is peer-reviewed or is referred to by a reputable source
3. The paper is not behind a paywall.
4. The title, abstract or introduction discusses RE or use of RE practices.
5. The papers keywords contain related RE keywords.

Exclusion set:

1. The paper is aimed at solely discussing SDLC's.
2. The paper discusses RE tailored to a specific platform or system.
3. The paper was published in an unknown or disreputable journal.
4. The paper predates 1980's
5. The paper is a duplicate of a previously found one.

*Selection of databases:* In choosing the databases listed (table 1) we concentrated on well-known sites containing peer-reviewed research papers that are kept up to date to ensure we had a broad selection in terms of historic papers and the latest findings relating to our study.

**Table 1.** Selected Databases

| | |
|---|---|
| IEEE Xplore | CMU Digital Library |
| SpringerLink | Research Gate |
| ScienceDirect | AIS Electronic Library |
| ACM Digital Library | Google Scholar |

*Generation of Search Strings:* The search terms documented in table 2 were generated mainly through brainstorming. Initially, keywords were proposed based on the researcher's current knowledge of the Requirements Engineering. This list was then refined and filtered for keywords found to be off topic or too large in scope.

Also included were people of note, associated with modelling of requirements, also from current knowledge. Additional known experts in RE were added to the keyword list as they were mentioned in relevant papers.

**Table 2.** Search Strings

| | |
|---|---|
| Software Requirements: Past, Present, Future | Software Requirements in XP |
| Software Requirements engineering | XP Requirements |
| Software Requirements evolution | Validating Requirements |
| Evolution of Software Requirements | History of requirements engineering |
| Software Requirements | Just in time requirements |
| OOP software Requirements | Non-functional Requirements in Agile Software Development |
| Detecting Requirements | Alistair Cockburn |
| Requirements Elicitation | Sommerville and Sawyer |
| Requirements Engineering | Goldsmith |
| Agile Requirements Engineering | |
| Agile Requirements | Requirements Engineering: A good practice guide |
| Scrum Requirements Engineering | Extreme chaos The Standish group Inc. |

*Paper Selection:* Each researcher took two databases each, and applied the search strings given in table 2, searching the full body of the paper. Taking the top …. results, each researcher then applied the inclusion and exclusion set to quickly filter the returned set of papers. This filtered set was then summarised into bullet points to allow discussion amongst researchers on the acceptance of a paper.

# 3 Analysis

## 3.1 Foundations of today's requirements

During the 1990s, requirements engineering progressed rapidly. Springer established an international requirements journal and the IEEE sponsored conference began, both of which, drove increased international cooperation. These rapid advancements led to requirements engineering evolving from an area of research in the field of software engineering to becoming its own respective field of study.

Three radical views emerged during the 1990s which overturned the orthodox views of requirements engineering. [7] Firstly, requirements modelling and analysis cannot be performed satisfactorily unless it is done with the knowledge of and in conjunction with the organisation and the social context of the proposed system. Without understanding this social context, requirements cannot be correctly elicited.

Secondly, any discussion of whether a given system meets its purpose can only become possible if the environment and achievements of the system and its purpose in that environment, are well documented. Prior to this, information flow modelling was used to ascertain if a system had met its purpose. It is the view of the authors that the switch toward modelling based on the stakeholder's goals resulted in models which increased the likelihood of a given system satisfying its purpose.

Lastly the attempt to produce consistent and complete requirements through RE modelling was dismissed as being a futile effort. Instead, research switched to trying to reason with inconsistent models. This was a key development, as it allowed the field to progress beyond trying to achieve an unrealistic gold standard model. Research then began to focus on areas, such as speeding up the requirements engineering process and identifying key problems. [8]

Christel and Kang highlighted some of the pressing issues in requirements elicitation, such as problems with scope, understanding and volatility, of requirements which were major issues in the 1990s. [4] Sommerville and Sawyer addressed these ambiguity problems in 1996 by establishing general good practice guidelines. [10] Their work provided classifications of how well companies implemented requirement activities such as system modelling, describing requirements and negotiation practices. These classifications were ranked as basic, intermediate or advanced. These guidelines led to further research and improved requirements engineering in general.

After a number of years, the advanced techniques advised by Sommerville and Sawyer in 1996 were considered the bare minimum in regards to successfully modelling a project. This was due to advances made in versioning software and tooling. In 2009 a group of researchers re-visited the work of Sommerville and Sawyer and updated the difficulty classification levels in line with new practices and technologies, such as version control and modelling software. [11]

Nuseibeh and Easterbrook highlighted the heavy-handed role of the "requirements engineer" and the importance of models in elicitation techniques. [7] Some techniques they documented, such as documentation analysis, consensus-building workshops and cognitive techniques are obsolete in today's software company. [7] Traditional techniques have been ditched for more cost-effective and speedy methods today. However,

we believe their research forms the basis for present-day techniques, such as prototyping, problem frames, use cases, and participant observation which forms a major part of Agile practices. [7][9]

Nuseibeh and Easterbrook also suggested that the future focus in RE would be on developing a number of techniques. [7] In particular, the need for multi-disciplinary training for requirements engineers to develop their social skills. This enables effective communication between a variety of stakeholders, reducing the number of issues caused by conflict amongst stakeholders. This is also a view held by Beus-Dukic & Alexander, who discussed the importance of learning the needs of stakeholders to create better requirements. They also state the need to involve different stakeholders with differing levels of experience and skill sets. [13]

Without this, requirements engineering in Agile would be significantly weaker, as Agile relies on effective communication to quickly establish core requirements. Nuseibeh and Easterbrook also highlighted the need for richer non-functional requirements models. [7] This is still a very big issue today and a continuing area of research.

### 3.2 The move from traditional to modern requirements

The two main approaches of pre-00's requirements engineering, in terms of user involvement, were Soft System Methodology (SSM) and Structured Systems Analysis and Design Method (SSADM).

SSADM supplies several sets of standards to aid requirements specification. The primary focus of SSADM is on functional and data requirement specification, with the secondary focus on non-functional, requirement goals and design. SSADM is documentation heavy and requires a considerate time investment. Components of SSADM, such as entity life histories and logical data structures require that users have a professional level of knowledge to understand. The SSADM process also has a large upfront cost and is not very flexible. [12]

SSM achieves requirements understanding through basic user involvement. Developers must act in a user-friendly manner, helping foster a better understanding of requirements by satisfying Nuseibeh and Easterbrook concept of the social developer. [13] This is done in an effort to discover problems and present simplistic diagrams to aid user-understanding. [7] Further analysis is conducted with users, through debating problems with them. From this, requirements are prioritized and identified.

It was stated that "Meaningful user involvement in systems development and an overall user orientation is critical to the success of any development project". [14] However, too much user involvement can be negative and result in issues such as feature creep and distract the main course of a software project. This can lead to delays, cost overruns and software that does not correctly fulfil the user's needs.

Positive user involvement in SSADM is achieved through following a strict set of guidelines with the involvement of users. Problems are defined through a feasibility study in SSADM. Further analysis is conducted through requirements investigation of business system operations in SSADM. The high level of user involvement in SSM can cause confusion amongst developers. SSADM requires users to have deep system understanding and knowledge of the language that describes complex systems. If users do

not possess the required knowledge of requirements it can lead to imprecise or incorrect requirements. SSADM's strict guidelines mean that changes to requirements are very difficult to implement. [14]

Extreme programming (XP) is a way of addressing many of the shortcomings found in SSADM. With the turn of the millennium, the need for Agile approaches to projects and applications became apparent. XP works with minimal requirements, only core details and concept knowledge is needed to satisfy the requirements. "Spike solutions" are used to aid user stories, which motivate the developer to explore the client domain. "Spike solutions" also expand technical knowledge, required to estimate a development timeframe. In XP, there is no system requirement specification, informal user stories are expected to take their place. [1]

Verification and validation are included as part of the development process in XP as unit and integration tests. Customer acceptance tests are checked by the user stories. XP is not really suited to requirements traceability. Most of the requirements are encapsulated by the code at the time. There is little documentation recording the requirements. Unit tests and integration tests are used instead to ensure requirements are valid. XP is mainly suited to projects where high reliability is not a pressing issue. XP is designed to avoid projects failing due to the lack of user involvement, this was something earlier techniques such as SSADM suffered from. [1]

RE in XP and other modern Agile methodologies have many benefits over Trad RE. Some of the shortfalls are the lack of traditional RE documentation in Agile methodologies. Agile methodologies focus on prototype modelling and the reliance on user stories as the primary requirement artefacts. [5-6] Agile requirement artefacts are developed and maintained by the actual developers. This can result in documentation which is hard to understand for other stakeholders.

Trad RE approaches this with large requirement documents that have well defined and upfront specifications. All requirement artefacts such as use cases, prototype models and non-functional decisions, are placed together so that project requirements can be reviewed at a later time.

A major trade-off for having minimal requirement artefacts in Agile RE is the challenge of conducting a formal requirement review. The assessment of requirement specifications "...can generate a considerable return on investment in terms of error cost reduction and development productivity". [3] Agile RE tried to address this by using sprint reviews, discussing any assumption with regard to functional requirements, implementation and issues encountered.

This is a fundamental issue of Agile RE as you "...cannot assess absolute completeness and correctness of requirements specifications". [3] As the requirements are altered and elaborated during the projects life cycle, Agile RE fails to take into account any functional requirements outside of the scope of the current sprint. [3]

### 3.3 The issues with today's requirements

An issue identified with both Trad RE and Agile RE is the ambiguity that can arise from capturing requirements in business language. Natural language has remained the

general choice for RE gathering and capturing. Agile RE places emphasis on short concise user stories, the level of detail on a given requirement is limited. This limited detail can lead to misinterpretation of requirements by developers, resulting in requirements not being fulfilled correctly. [16] The issues with natural language can be addressed with frameworks proposed, such as QUS (Quality User Story). [17] This framework attempts to implement a set of criteria for ensuring the quality generation of user stories.

This is important as there are over "80 syntactic variants" of accepted user stories and the variances between different developers within a team can vary hugely. AQUSA (Automatic Quality User Story Artisan) is an automated tool that uses state-of-the-art NLP to apply the QUS framework to both rate and modify user stories. [17] The use of such tools and frameworks help remove manual quality assurance steps and ensure low ambiguity when implementing user stories.

Another proposed solution to addressing the issues of natural language in RE is the use of Business Process models to aid RE elicitation. [18] Ordñez et al. argued studies have shown that graphical notional enables faster cognitive understanding and aids in the early realisation of error in requirements. The use of graphical notation during developer-client communication enables early understanding of requirements making it more likely the client is satisfied with the work done. Ordñez et al. found that the use of business process models improved communication between the analyst and the client and resulted in a more productive elicitation process.

Just in time requirements (JITR) follow the Agile principle of "deciding as late as responsibly possible", leaving the elaboration of a requirement until it is due to be implemented. [19] This approach avoids tying requirements elicitation or discovery to a particular phase. Allowing an iterative evolution of the requirements specification as the project progresses. Due to "as we need it" level of detail, the quality assurances offered by AQUSA are not applicable. [19]

However, one particular study targets JITR directly, proposing criteria that respects and takes into account the level of detail that is needed for that requirement at that particular time. [20] Ensuring that each requirement is well formed, not conflicting with one another and is not ambiguous. Whilst remaining as a lightweight user story that aids in the resolution time of requirements while avoiding the classic heavy upfront analysis. [21-22] While both these case studies deal with open source projects, bug reports and feature requests of closed source projects have been argued to be almost identical. [20] Both of these frameworks could address the issue of solely relying on sprint reviews in Agile, to ensure that a functional requirement meets minimum criteria.

The negligence of non-functional requirements, was and still is very much one of the major challenges in Agile requirements engineering. The absence of well-defined non-functional requirements documentation may result in poor quality of software, scalability issues and increased maintenance costs. [5] Very often non-functional requirements are hard to define and this results in ill-defined or poorly documented requirements. In Agile development, documenting non-functional requirements becomes challenging. Due to Agile development focusing on the working system rather than comprehensive documentation. The non-functional requirements are rarely taken into consideration in the industry. [23] Many developers are aware of the importance of non-

functional requirements but neglect them due to the lack of time and absence of support from any languages or tools.

"NFRs (Non-functional requirements) play an important role in the success of software systems. In ASD (Agile software development), existing requirements engineering practices fail shortly regarding the documentation of NFRs. For instance, user stories of ASD have limitations in specifying and documenting NFRs." [23]

Very often non-functional requirements are being avoided in the early stages of development and this results in major issues on the later stages [23]. For example, failure to consider security requirements can lead to many vulnerabilities within the system. Agile practices of documenting requirements have significant limitations for documenting non-functional requirements.

To overcome the drawbacks of Agile software development in non-functional requirements, proposals like Non-functional Requirements Model for Agile Process (NORMAP) or Non-functional Requirements Planning (NORPLAN). [21] NORMAP method links non-functional requirements to functional requirements, combining both in the single model. NORPLAN method proposes prioritization schemes such as Riskiest-Requirements-First (RRF) and Riskiest-Requirements-Last (RRL), which are optimized for Agile planning.

The proposed schemes disregard the product owner's requested priority of requirements and are based upon calculating the risks using requirements quality metrics. Furthermore, there has been research in automating the NORMAP methodology, to model non-functional requirements in an Agile environment. [24]

Research shows there is still a need for the upfront elicitation and elaboration of a projects non-functional requirements on the early stages of development life. [25] However as outlined above this approach can be combined with Agile centred methodologies in a hybrid approach that addresses the common problems with both Trad RE and Agile RE. [6][26-27]

### 3.4 Move towards the future

Recent research has shown that early involvement of end-users is important for software development and software evolution. [28] It is challenging for software companies to gather end user feedback. Collecting relevant end-user feedback may help improve usability and the overall quality of the system. There are a number of ways the end-users can provide feedback, including linguistics, such as social media comments and non-linguistic formats such as five-star style reviews present in app stores.

Another major challenge of getting feedback from the end users is the lack of motivation in providing feedback. In the last few years' social networks have become a valuable source for feedback gathering and requirements engineering. Social networks such as Facebook or Twitter, have grown exponentially in the last decade. Software companies can utilise these social media communication channels with their users for feedback and RE gathering. [28] Studies have shown that end users are more motivated to provide quality feedback on social networks. However, many software companies are still not exploiting the full potential of feedback gathering through social media. [28]

Twitter user data can be used to analyse current brand sentiment, and studies have been performed to see if it is viable to extend this to software relevant feedback. Well known and documented techniques in text processing, classification and summarization can be applied to the task of data mining this twitter dataset with a good degree of success given the limited size of the test dataset. [29] Further research points to the role of the requirements engineer incorporating the duties of a data analyst. This would allow the exploitation of existing sources of potential requirements such as product logs and bug trackers. [30]

Bug trackers are already being utilized to identify potential requirements manually, so being able to leverage automated methods to extract these requirements could reduce the time and cost involved due to automation. [19] In an agile environment this could help overcome the overhead by reducing the need for constant face to face communication with stakeholders, and overcome scalability issues in talking to a multitude of stakeholders. [5]

## 4 Limitations of Research

We have defined our limitations of research as the following. This paper does not account for absolutely all research done in the area of "Software Requirements: Past, Present, Future", as our studies are only a shallow examination of a large area of scope. Due to organisations privacy concerns, it is challenging to accurately state what RE practices were in use at the time.

## 5 Future Work

Further investigations could be performed on how agile can benefit from integrating lightweight engineering processes that can capture non-functional requirements on early stages of the development cycle.

This was noted as there is currently no accepted way of managing non-functional requirements in RE. Further research can be done in identifying and formalising ways of defining, specifying and measuring the non-functional requirements. The lack of frameworks for managing and defining non-functional requirements can lead to miscalculation of non-functional requirements.

Investigation of how machine learning techniques and neural networks can be further integrated into the process of user's feedback gathering and generating requirements. This could allow for orders of magnitude increases in requirements gathering from readily available sources. Potentially, companies can do this not only for their own projects but also that of their competitors. This could allow them to identify potential shortcomings of their competitors.

Research could also be conducted on possible ways of implementing a hybrid of both Trad and Agile RE practices. This could be in the form of a framework or tool to aid in traceability and reviews of requirements.

## 6 Conclusions

This paper discusses the evolution of software requirements engineering techniques from the not so distant past, to the techniques used in the present day. We have examined this by looking at the early core principles, the move to Agile, looking at today's RE issues and trends of note for the future.

RE principles established in the 1990s are still relevant today. Being aware of social context and the environment a system will work in, are regarded as key areas of understanding which enable quality software development. [10-11]

The elicitation models for non-functional requirements are still a challenge today. [21] There is no Agile approach to eliciting concise non-functional requirements. [23] It is the author's opinion that the use of hybrid models, based on combining Trad RE and agile RE techniques will alleviate this issue.

Natural language in requirements elicitation has been shown to cause confusion and to be insufficient in defining concise requirements. Including diagrams and graphics has been proven to facilitate better understanding amongst stakeholders. Implementation of frameworks can aid in the standardisation of user story quality. [18] It is the author's opinion that this will lead to a trend of further hybridisation of RE models.

This paper explores the potential usages of social networks for end-user's feedback gathering. We believe this is trending towards the further automation of RE. Research has proven that end-user's feedback can be a valuable source of requirements and can aid in the identification of requirements on a much larger scale than was previously possible. [28-30]

## References

1. Cohn, T., Paul, R.: A Comparison of Requirements Engineering in Extreme Programming (XP) and Conventional Software Development Methodologies. AMCIS 2001 PROCEEDINGS. 256 (2001).

2. Brooks Jr, F.P.: No Silver Bullet Essence and Accidents of Software Engineering. Computer. 20, 10–19 (1987).

3. Salger, F.: Requirements reviews revisited: Residual challenges and open research questions. 2013 21st IEEE International Requirements Engineering Conference (RE). (2013).

4. Christel, M.G., Kang, K.C.: Issues in requirements elicitation. Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA (1992).

5. Inayata, I. et al.: A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior. 51, 915–929 (2015).

6. Elshandidy, H., Mazen, S.: Agile and Traditional Requirements Engineering: A Survey. International Journal of Scientific & Engineering Research. 4, 9, 473–482 (2013).

7. Svahnberg, M. et al.: Uni-REPM: a framework for requirements engineering process assessment. Requirements Engineering. 20, 1, 91–118 (2013).

8. Nuseibeh, B., Easterbrook, S.: Requirements engineering. Proceedings of the conference on The future of Software engineering - ICSE 00. (2000).

9. Robinson, W., Pawlowski, S.: Managing requirements inconsistency with development goal monitors. IEEE Transactions on Software Engineering. 25, 6, 816–835 (1999).

10. Jackson, M.: Software requirements & specifications: a lexicon of practice, principles and prejudices. Addison-Wesley, Harlow (1995).

11. Sommerville, I., Sawyer, P.: Requirements engineering: a good practice guide. Wiley, Chichester (1996).

12. Cox, K. et al.: Empirical study of Sommerville and Sawyers requirements engineering practices. IET Software. 3, 5, 339–355 (2009).

13. Ashworth, C.M.: Using SSADM to specify requirements. IEE Colloquium on Requirements Capture and Specification for Critical Systems. (1989).

14. Alexander, I., Beus-Dukic, L.: Discovering requirements: how to specify products and services. John Wiley and Sons, Hoboken, NJ (2009).

15. Sun, Z.: User Involvement in System Development Process. Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013). (2013).

16. Lu, C.-W. et al.: A Model-based Object-oriented Approach to Requirement Engineering (MORE). 31st Annual International Computer Software and Applications Conference - Vol. 1- (COMPSAC 2007). (2007).

17. Lucassen, G. et al.: Improving agile requirements: the Quality User Story framework and tool. Requirements Engineering. 21, 3, 383–403 (2016).

18. Ordoñez, H. et al.: Eliciting Requirements in Extreme Programming (XP) Through Business Process Models. Conisoft .2015 - Congreso Internacional de Investigación e Innovación en Ingeniería de Software. 1, (2015).

19. Ernst, N.A., Murphy, G.C.: Case studies in just-in-time requirements analysis. 2012 Second IEEE International Workshop on Empirical Requirements Engineering (EmpiRE). (2012).

20. Heck, P., Zaidman, A.: A framework for quality assessment of just-in-time requirements: the case of open source feature requests. Requirements Engineering. 22, 4, 453–473 (2016).

21. Do, A.Q., Bhowmik, T.: Refinement and Resolution of Just-in-Time Requirements in Open Source Software: A Case Study. 2017 IEEE 25th International Requirements Engineering Conference Workshops (REW). (2017).

22. Bhowmik, T., Reddivari, S.: Resolution trend of just-in-time requirements in open source software development. 2015 IEEE Workshop on Just-In-Time Requirements Engineering (JITRE). (2015).

23. Behutiye, W. et al.: Non-functional Requirements Documentation in Agile Software Development: Challenges and Solution Proposal. Product-Focused Software Process Improvement Lecture Notes in Computer Science. 515–522 (2017).

24. Farid, W.M., Mitropoulos, F.J.: NORMATIC: A visual tool for modeling Non-Functional Requirements in agile processes. 2012 Proceedings of IEEE Southeastcon. (2012).

25. Bajpai, V., Gorthi, R.P.: On non-functional requirements: A survey. 2012 IEEE Students Conference on Electrical, Electronics and Computer Science. (2012).

26. West, D.: Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today. Technical Report. (2011).

27. Theocharis, G. et al.: Is Water-Scrum-Fall Reality? On the Use of Agile and Traditional Development Practices. Product-Focused Software Process Improvement Lecture Notes in Computer Science. 149–166 (2015).

28. Stade, M., Fotrousi, F., Seyff, N., Albrecht, O.: Feedback Gathering from an Industrial Point of View. 2017 IEEE 25th International Requirements Engineering Conference (RE). (2017).

29. Williams, G., Mahmoud, A.: Mining Twitter Feeds for Software User Requirements. 2017 IEEE 25th International Requirements Engineering Conference (RE). (2017).

30. Villela, K. et al.: Ubiquitous Requirements Engineering: A Paradigm Shift That Affects Everyone. IEEE Software. 36, 2, 8–12 (2019).