Requirements Quality Defect Detection with the Qualicen Requirements Scout

Henning Femmer

Technical University Munich and Qualicen GmbH
Munich, Germany
henning.femmer@qualicen.de

Abstract

Our group worked on both redefining quality in RE, as well as simple methods for quality defect detection, their potential and limitations. This report summarizes the main challenges from our industrial perspective, which are: Precision, relevance, process, and summarization.

1 A Brief Team History

Our team was founded by four PhDs and a Professor from the department of informatics at the Technical University Munich (TUM). At the chair for software & systems engineering of Manfred Broy, we conducted bilateral research motivated by the needs of our industrial partners, i.a. Munich Re, Daimler AG, TechDivision, and Wacker Chemie AG. At that time, we experimented with combining the existing source code quality analysis toolkit ConQAT with NLP techniques in order to detect quality issues in system tests [HJE+13] and requirements [FMJ+14]. As our experiments lead to promising results, the companies started asking for productive systems instead of experimental academic prototypes.

At this point we founded the HEJF GBR, which was shortly afterwards succeeded by the Qualicen GmbH[1]. Qualicen is now a quickly growing company with currently eleven employees, located in Garching near Munich, Germany. Qualicen does test and requirements engineering coaching, consulting and tooling. Our customers come from various domains, including automotive, aerospace, healthcare, and insurance.

2 Technical Outcome: The Qualicen Requirements Scout

The main technical outcome of our group is the Qualicen Scout. Qualicen Scout searches for quality findings in requirements and system tests written in natural language. It is based on the continuous source code analysis tool Teamscale[2]. To detect quality findings, the Scout is attached to a requirements data source, such as a PTC Integrity or DOORS NG requirements database, an SVN or a GIT repository. The Scout then continuously pulls for new versions of the requirements and thus creates a full history of all automatically detectable quality defects. Whenever an author updates the requirements, the system immediately analyzes whether this update introduced new findings or fixed existing findings and reports this back to the team.

With this information, Qualicen Scout users serves three basic use cases:

Rapid Feedback: The largest possible benefit you can get from automatic tools is, when the engineers creating artifacts are notified straight after they made a mistake. Not only is fixing the defect the cheapest, but the learning effect is also much stronger. Therefore, we support various RE tools, such as PTC Integrity, IBM DOORS NG or Microsoft Word with plugins for rapid feedback (see Fig. 1).

[1] http://www.qualicen.de

[2] https://cqse.eu/en/products/teamscale/landing/

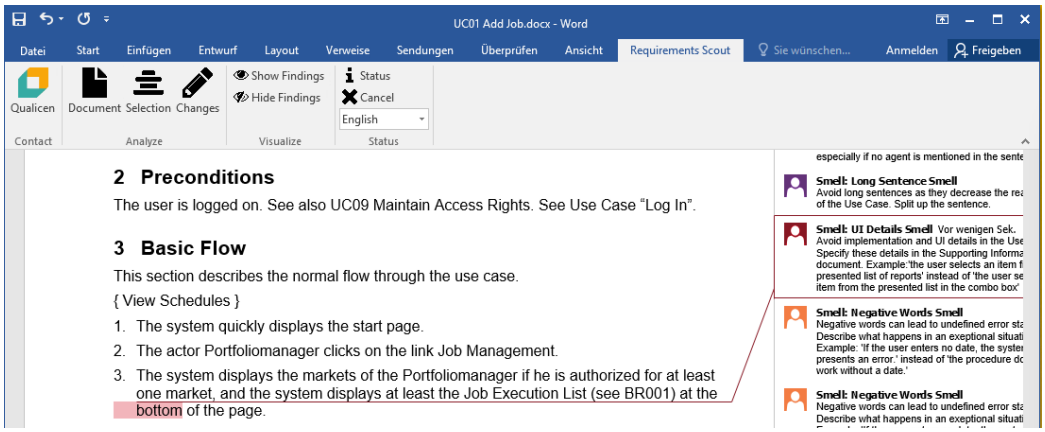

Figure 1: Author plugins (e.g., in Microsoft Word) provide rapid feedback to requirements engineers.

Tool-supported audits & reviews: One of the most common situations where we apply our tools is for reviews or tool-supported audits. These can be in one of two situations: Either this is the first time we look at the requirements, or we want to understand how things changed since the last review. For the former, we use similar perspectives as the ones described in the first use case. For the latter, we have a specific delta perspective that describes which files, sections, metrics, and findings have been changed between two point in time. In addition, the delta perspective provides the reviewer with an analysis whether the engineer has worked in a specific section but has ignored a certain defect. In tool-supported audits we found that it is of utmost importance to combine the findings found by the tool with the findings found by a manual review of an expert.

Trend analysis: Lastly, there are multiple roles in the RE process that do not so much care about individual quality defects, but more about the trend. Especially in situations with heavy reuse (e.g., in the automotive industry) it is unrealistic to expect people to iterate through all findings for all existing requirements. Instead, quality engineers and project leads assume that over time the quality improves. For this, one can use metrics such as number of findings or findings density[3] and visualize their trend over time. These roles are interested in trends and have a more dashboard-like viewpoint onto the system (see Fig. 2).

Of course, the scout offers various necessary features, such as creating custom dashboards for a team, notifying users about changes, customizing the analyzed criteria or hiding (*blacklisting*) false positive findings. In the following, we explain our past research, which includes the types of defects that Qualicen Scout detects.

3 Academic Outcome: Past Research

In the past, we worked specifically on RE and system test quality. For a summary on the works on system test quality, please refer to the work by Hauptmann [Hau16]. In the following, we summarize the RE specific outcomes. They are separated into three main questions: First, what is RE artifact quality as a concept? Second, to which extent can we automatically detect quality defects? And third, where are the (theoretical and practical) limitations of automatic defect detection?

3.1 What is RE Artifact Quality?

Regarding the first question, we base our view onto the fact that RE artifacts are just a means and not an end [FMM15, FV18]. As such, the definition of a *high quality* artifact depends on its purpose. The purpose of RE artifacts can be of different types (see [Fem17, p.12ff] for details), but it usually breaks down into the simple (and simplified) question: Which quality factors (properties of the artifact) can make the artifact more efficient or effective to use[4]? We call this paradigm *Activity-based RE artifact quality models* or *ABRE-QM*.

The ABRE-QM paradigm allows us to operationalize quality through impact on effectiveness and efficiency in usage. To understand the impact of certain quality factors, we analyzed the impact of passive voice on understanding requirements [FKV14] and creating good test cases based on requirements [MFME15, BJFF17].

[3] Probability that a random word is subject to a finding

[4] As a consequence, the quality meta-model model then is a slight modification of the Quamoco quality model [WLH+12].

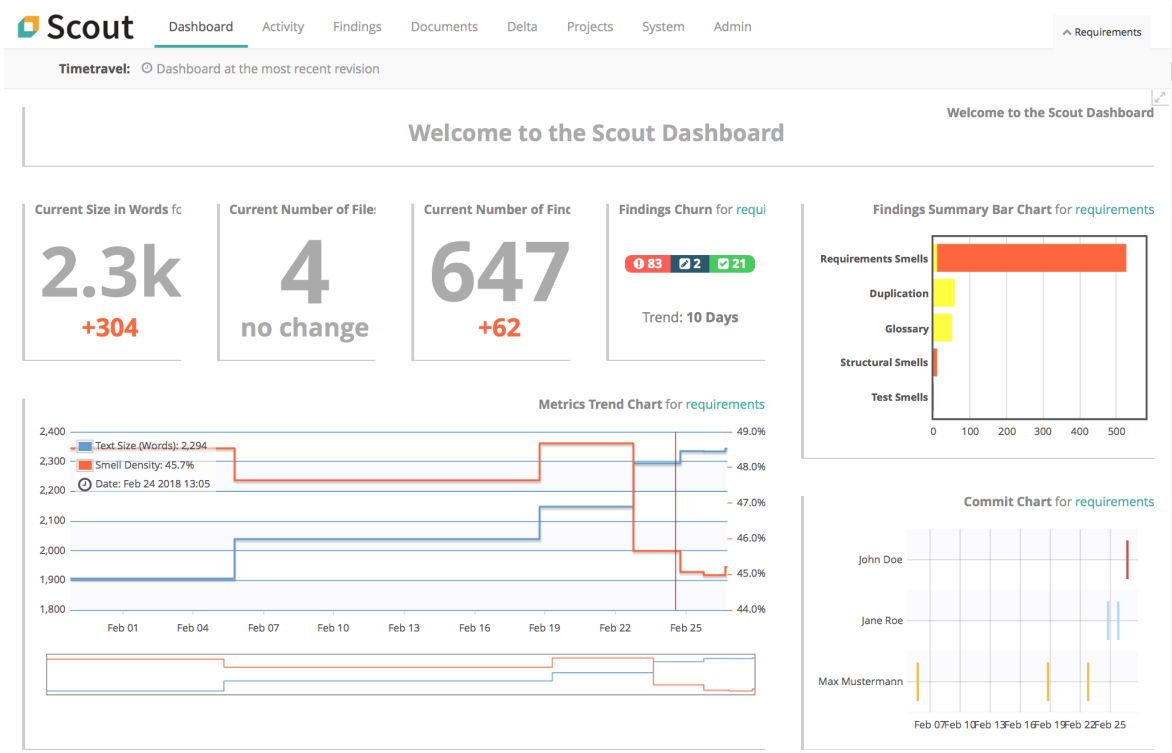

Figure 2: The dashboard shows trends to quality engineers and team or department leads.

3.2 Which quality factors can we automatically detect and how?

In [FMWE17] and [FUG17], we gained a first understanding to which extent we can and cannot automatically detect quality defects. Based on the notion of code smells [FB99], we refer to these automatically detectable types as *requirements smells*.

Types of Smells. We differentiate the types of defects, depending on the scope of information that is accessed. The defect can relate to a word (*lexical smells*), the grammar of a text (*grammatical smells*), the structure of the text (*structural*), or the semantics (*semantical smells*). Of these, the last category is a bit imprecise, since most smells come with a semantic problem that they address and a syntactic (for an automated system analyzable) mechanism for detection. Accordingly, smells in the semantic category have to be broken down to lexical, grammatical or structural aspects in order to be automatically detectable.

Methods. To detect these smells, we apply the common types of NLP mechanisms: Word- and Sentence Splitting, Morphologic analysis, Lemmatization (and sometimes stemming), POS tagging, and syntactic parsing. We also experimented with shallow semantic parsing and dependency analysis. Our tool chain relies on the framework DKPro [dCG14], which enables to switch NLP tools without intensive effort. Interestingly, as we found in [FUG17], the three most useful mechanisms are not the strong weapons of NLP, but simple mechanisms, such as dictionaries, regular expressions, and formatting information.

However, as we found in our work: Our main effort is not spent in detecting defects, but in improving the precision. For that, we make heavy use of context-based filtering mechanisms, similar to the ones proposed by Krisch and Houdek [KH15].

3.3 Which quality factors can we not automatically detect and why?

We found the following reasons, why certain quality factors cannot be automatically detected: The quality factor refers to *stakeholder or domain knowledge*, to the *semantic understanding* of natural language, to the *scope or goal* of the system, to the development *process*, or that the quality factor itself is *vague or subjectively defined*. When we analyzed a large guideline by an industrial partner, we found that, surprisingly, the main challenge was not the technical limitations of the NLP. Instead >80% of undetectable rules were actually undetectable due to the vague or imprecise rules [FUG17].

4 Challenges from an Industrial Perspective

In the following, we want to summarize the main challenges in the area of automatic detection of quality defects in RE from our industry-focussed perspective.

The Precision Challenge: From our perspective and in contrast to a popular opinion in academia, the main challenge for creating user acceptance is precision. We see two reasons for this: First, in our experience, if users receive a certain percentage of incorrect findings, they less and less tend to respect even the correct findings. So, if users receive more and more false positives they will stop using the system altogether. This is different with recall. In our experience, users are more willing to invest into manual effort for finding additional defects than for ignoring suggested defects. Second, these automatic detection mechanisms quickly turn into metrics and assessments for teams (as in the trend analysis use case described above). For an assessment, however, teams tend to more openly accept a tool, when the tool misses half of the defects, but each finding is correct (the metric is more optimistic than reality), than if the tool finds all defects but only every second finding is actually a defect. In our opinion, this is because in the recall-over-precision case, the metric lets a team look worse than it actually is.[5]

If an automatic detection lacks precision, the reason is either the NLP or the rules based on the NLP. For the former, academia has to understand that what is considered reasonably good in the NLP community is not sufficient for most automatic defect detection tasks. For example, POS detection is widely considered a solved problem, while we often still struggle with incorrect POS tags. Second, our main effort nowadays is adapting rules to a new context. The main challenge here is to either identify a universal rule set or create systems that automatically adapt to the context (e.g. based on user feedback).

The Relevance Challenge: The second challenge, which is also widely recognized in academia, is to create a tool that detects relevant issues. For this, we still have to customize the tool to new customers, since every team comes with different styles and consequently also different (or new) quality factors. Only two ways out of this variation problem exist: Either tools will automatically adapt to the various contexts, or the RE language will become more similar between the various teams. Currently, we see progress on both ways.

The Process Challenge: Since there are quality factors that cannot be automatically detected, quality assurance needs combinations of manual and automatic methods. First thoughts on combining automatic and manual QA into a more efficient QA process can be found in [FHEM16].

The Summarization Challenge: Lastly, practitioners need easily accessible information on what is good and bad in which context. In academic lingua, what we need is a common theory for RE artifact quality: A plethora of NL-based quality factors exist. Starting from lexical issues, such as weak words, to various types of ambiguities and potentially harmful constructs such as passive voice or nominalizations. Which of these factors exist? What is their impact? In which context does this impact apply? The work on the ambiguity handbook [BKK03] is a great step in that direction. As a next step, we need to index this information and make it more accessible and more maintainable than small studies in individual research papers. One idea to start this can be as simple as a wiki-page with a list of quality factors and their assumed consequences.

5 Summary

With the need for higher speed, lower cost, and higher quality in software engineering, there is an urgent need for automatic support in both analytical and constructive quality assurance of RE artifacts. While the pull from industrial customers is evident, there is still plenty of work left in order to make automatic NLP-based QA checks in RE as natural as spell checkers. To us, the question is not whether requirements engineers will use automatic tool support for QA, but only when the precision-recall relation is good enough for wide-spread user acceptance.

Acknowledgements

This work was performed within the project Q-Effekt; it was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01IS15003 A-B. The author assumes responsibility for the content. Thanks to Maximilian Junker for his thoughts and review.

[5] Nevertheless, of course, we are not arguing for ignoring the recall. We rather want to motivate teams to not constrain their research by the 100%-recall-assumption, and motivate teams to give tools into the hands of practitioners. Only then can you find out whether the approach is accepted by users.

References

[BJFF17] Armin Beer, Maximilian Junker, Henning Femmer, and Michael Felderer. Initial investigations on the influence of requirement smells on test-case design. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 323–326. IEEE, 2017.

[BKK03] Daniel M. Berry, Erik Kamsties, and Michael M Krieger. From contract drafting to software specification: Linguistic sources of ambiguity. Technical report, University of Waterloo, 2003.

[dCG14] Richard Eckart de Castilho and Iryna Gurevych. A broad-coverage collection of portable NLP components for building shareable analysis pipelines. In *Workshop on Open Infrastructures and Analysis Frameworks for HLT*, OIAF4HLT, pages 1–11, 2014.

[FB99] Martin Fowler and Kent Beck. *Refactoring: improving the design of existing code.* Addison-Wesley Professional, 1999.

[Fem17] Henning Femmer. *Requirements Engineering Artifact Quality: Definition and Control.* PhD thesis, Technische Universität München, 2017.

[FHEM16] Henning Femmer, Benedikt Hauptmann, Sebastian Eder, and Dagmar Moser. Quality assurance of requirements artifacts in practice: A case study and a process proposal. In *PROFES*, pages 506–516. Springer, 2016.

[FKV14] Henning Femmer, Jan Kučera, and Antonio Vetrò. On the impact of passive voice requirements on domain modelling. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 21:1–21:4. ACM, 2014.

[FMJ+14] Henning Femmer, Daniel Méndez Fernández, Elmar Juergens, Michael Klose, Ilona Zimmer, and Jörg Zimmer. Rapid requirements checks with requirements smells: Two case studies. In *RCoSE*, pages 10–19. ACM, 2014.

[FMM15] Henning Femmer, Jakob Mund, and Daniel Méndez Fernández. It's the activities, stupid! A new perspective on RE quality. In *International Workshop on Requirements Engineering and Testing*, RET, pages 13–19. IEEE, 2015.

[FMWE17] Henning Femmer, Daniel Méndez Fernández, Stefan Wagner, and Sebastian Eder. Rapid quality assurance with requirements smells. *Journal of Systems and Software*, 123:190–213, 2017.

[FUG17] Henning Femmer, Michael Unterkalmsteiner, and Tony Gorschek. Which requirements artifact quality defects are automatically detectable? A case study. In *AIRE*, pages 1–7. IEEE, 2017.

[FV18] Henning Femmer and Andreas Vogelsang. Requirements quality is quality in use. *To Appear in IEEE Software*, 2018.

[Hau16] Benedikt Hauptmann. *Reducing System Testing Effort by Focusing on Commonalities in Test Procedures.* PhD thesis, Technische Universität München, 2016.

[HJE+13] Benedikt Hauptmann, Maximilian Junker, Sebastian Eder, Lars Heinemann, Rudolf Vaas, and Peter Braun. Hunting for smells in natural language tests. In *International Conference on Software Engineering*, ICSE, pages 1217–1220. IEEE, 2013.

[KH15] Jennifer Krisch and Frank Houdek. The myth of bad passive voice and weak words: An empirical investigation in the automotive industry. In *RE*. IEEE, 2015.

[MFME15] Jakob Mund, Henning Femmer, Daniel Méndez Fernández, and Jonas Eckhardt. Does quality of requirements specifications matter? combined results of two empirical studies. In *International Symposium on Empirical Software Engineering and Measurement*, ESEM, pages 144–153. ACM, 2015.

[WLH+12] Stefan Wagner, Klaus Lochmann, Lars Heinemann, Michael Kläs, Adam Trendowicz, Reinhold Plösch, Andreas Seidl, Andreas Goeb, and Jonathan Streit. The quamoco product quality modelling and assessment approach. In *International Conference on Software Engineering*, ICSE, pages 1133–1142. IEEE, 2012.