# Quality Criteria for Just-in-Time Requirements: Just Enough, Just-in-Time?

Petra Heck
Fontys Applied University
Eindhoven, The Netherlands
Email: p.heck@fontys.nl

Andy Zaidman
Delft University of Technology
Delft, The Netherlands
Email: a.e.zaidman@tudelft.nl

***Abstract*—Just-in-time (JIT) requirements drive agile teams in planning and implementing software systems. In this paper, we start with the hypothesis that performing informal verification of JIT requirements is useful. For this purpose we propose a framework for quality criteria for JIT requirements. This framework can be used by JIT teams to define 'just-enough' quality criteria. The framework also includes a time dimension such that quality criteria can be defined as 'just-in-time'. We demonstrate the application of this framework to feature requests in open source projects and explain how it could be customized for other JIT environments. We present our results for feature requests in open source projects, to show that there is a difference between creation-time quality and just-in-time quality. As this is ongoing research, we also list several points for discussion and future work.**

## I. Introduction

Requirements verification "ensures that requirements specifications and models meet the necessary standard of quality to allow them to be used effectively to guide further work" [1]. For traditional up-front requirements there are many guidelines, frameworks and standards (e.g. BABOK [1], Volere [2], IEEE-830 [3]) that define this 'standard of quality': requirements should be complete, unambiguous, specific, time-bounded, consistent, etc.

Just-in-time (JIT) requirements are "initially sketched out with simple natural language statements" [4], only to be fully elaborated when being developed. Moreover, JIT requirements engineering (JIT RE) heavily relies on face-to-face communication [5] and thus JIT requirements are not necessarily fully elaborated in written form. As such quality criteria like *complete* or *specific* might not always hold for JIT requirements. This observation has led us to investigate the quality criteria that apply to JIT requirements. Do the ones defined for traditional up-front requirements still hold? Which new ones are identified? Our goal is to guide practitioners on how to perform informal verification of JIT requirements.

Section II explains why in many situations it is useful to perform informal verification of JIT requirements. Section III presents our framework for quality criteria for informal verification of JIT requirements. It describes the application of our framework to feature requests in open source projects. Section IV highlights points for discussion around informal verification of JIT requirements.

## II. Informal Verification of JIT requirements

On the one hand there is the idea that JIT RE practices solve the initial 'vagueness' of JIT requirements not by documenting according to standards, but by e.g., face-to-face communication or prototyping [5]. On the other hand there is the observation that this JIT RE practice poses a challenge because many situations (e.g., distributed teams, large teams, complex projects) require documented JIT requirements that are fully elaborated in writing [5]. As soon as JIT teams cannot rely on face-to-face communication the 'correctness' of written documentation becomes more and more important.

An example of such a situation are the feature requests in open source projects that we studied in earlier work [6], [7]. Because of their distributed and on-line nature, open source projects document these requests for new or enhanced functionality and even the discussions that take place during the entire life-cycle of the feature request, mostly in issue trackers. Several quality problems for issue reports in open source projects have been investigated by e.g. Herzig et al. [8] and Bettenburg et al. [9].

As part of our own ongoing research [10] we have interviewed eight agile practitioners in person. The practitioners all answer 'yes' when asked if 'written JIT requirements should fulfill quality criteria' (as opposed to the claim made by Leffingwell [11] that user stories are throw-away artefacts). According to the practitioners, good quality agile requirements help the understanding within the team and are important for traceability or accountability towards the rest of the organization. When asked for quality criteria they apply in their daily practice they mentioned: unambiguous, uniform, atomic, SMART (see [12]), INVEST (see [13]), easy to modify.

Our findings until now support our claim that in many situations it is useful to verify quality criteria of JIT requirements. The next section presents our framework for quality criteria for JIT requirements.

## III. JIT Quality Criteria for Open Source Feature Requests

The way of working in JIT environments encompasses two things for requirements documentation:

1) **Just-enough** Only document what is needed for developing a good quality software product, the rest is solved by e.g. face-to-face communication or prototyping.

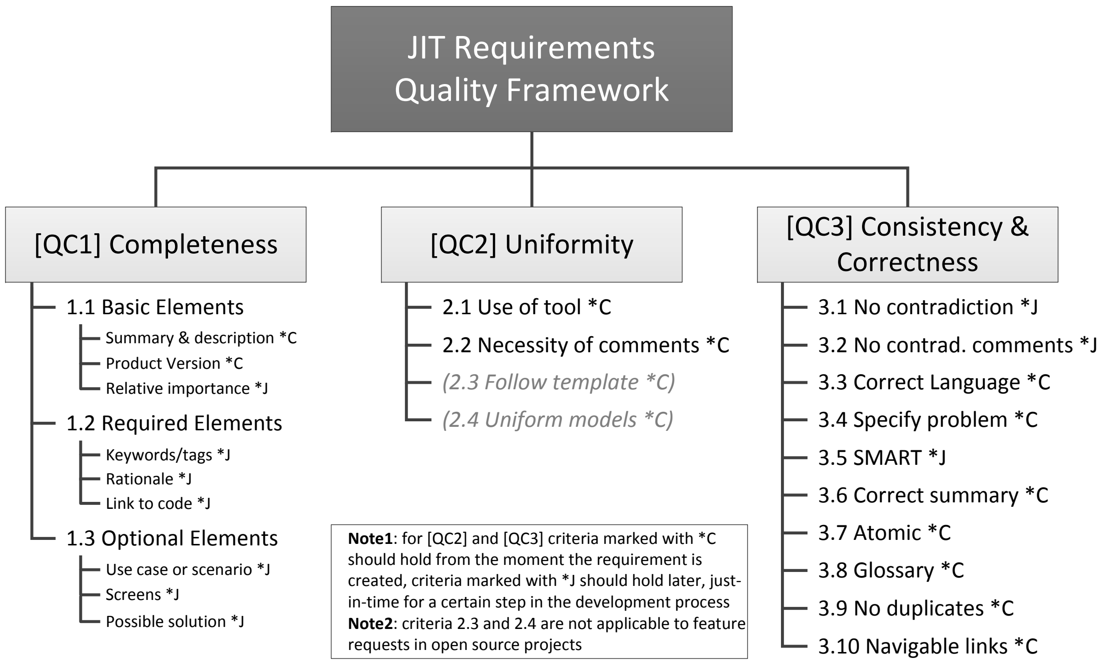

Fig. 1. JIT Requirements Quality Framework

2) **Just-in-time** Do not document things far before they are implemented, because there is a risk that they change in the mean time (or are even no longer needed).

What constitutes just-enough and just-in-time is different for each environment. Do we have a customer on-site or off-site? Is the team globally distributed? What is the size of the team? Is there a separate test team or are testers part of the team? Does the software contain complex algorithms or is it life-critical? Because of this variation in JIT environments we deem it necessary to firstly develop a framework in which we can structure quality criteria for different types of JIT requirements. We have instantiated this framework for feature requests in open source projects and applied it to three existing open source projects.

## A. A Framework for JIT Quality Criteria

We developed a first version of a framework for JIT quality criteria [10] that was an evolution of a framework developed for traditional requirements [14]. According to us the division in three quality areas (Quality Criteria QC) also holds for JIT requirements. JIT requirements also benefit from all elements being present and specified in a uniform way. Together with consistency and correctness this lowers the amount of time needed for discussion and the chance of rework (e.g. caused by misinterpretation) later in the process.

**[QC1]** Completeness w.r.t. Elements. All elements of a single requirement should be present. We consider three levels: basic, required, and optional. In that way we can differentiate between requirement elements that are mandatory and elements that are nice to have.

**[QC2]** Uniformity. The style and format should be standardized. This leads to less time for understanding and managing the requirements, because all team members know where to look for what information or how to read certain models attached to the requirement.

**[QC3]** Consistency & Correctness. The JIT requirements should be consistent and correct.

The overall QCs should hold for each type of JIT requirements. The QCs are detailed into specific criteria [QCx.x] for each type of JIT requirements. In that way each team can add ‘just enough’ quality criteria for their specific environment.

As said JIT requirements are “initially sketched out with simple natural language statements” [4], only to be fully elaborated when being developed. This leads us to introduce a second dimension in our quality framework: the notion of time. For each of the quality criteria we indicate when it should hold:

***C** Creation-time. This criterion should hold at soon as the requirement or the requirement part is created.

***J** Just-in-time. This criterion does not necessarily have to hold when the requirement (part) is created. However, it should hold at a later moment, just-in-time for a certain step in the development process. This could be further detailed by specifying which step is the latest moment for the criterion to hold, such that more than the current 2 time-points are specified.

In that way each team can specify which quality criteria should be there from the beginning and which quality criteria should be there just-in-time. Figure 1 shows the instantiation of the framework for feature requests. For a detailed explanation of the quality criteria, see [10].

Customizing the framework for other types of JIT environments would mean that the team decides which of the specific quality criteria (QCx.x) hold for their situation and which new ones should be added. The resulting list of quality criteria can be used as a checklist for the team to decide when a JIT requirement is 'Ready' [15]. An example are the quality criteria [QC2.3] (Follow template 'As a <role>, I can <activity> so that <business value>') and [QC2.4] (if attachments are added they should use the same modeling language) in Fig. 1 that we recommend for user stories in agile environments. Both quality criteria are usually not applicable to open source feature requests.

## B. *Quality of Open Source Feature Requests*

We have asked 83 last-year software engineering students to apply our framework to 570 feature requests from three open source projects: 200 feature requests from ArgoUML[1], 80 from Mylyn Tasks[2] and 290 from Netbeans[3]. For this purpose we have developed a scoring algorithm to calculate percentages for each of the quality criteria. For details of the experiment and the scoring algorithm see [10].

The experiment gave us valuable insights into feature request quality in open source projects. The aforementioned interview with agile practitioners also confirmed that the framework could be valuable as a 'checklist' or 'definition of done' for requirements in JIT environments.

The feature requests in the experiment are all in status "Closed". The participants have rated all quality criteria for the feature requests (both *C and *J). We define this as the 'final quality state' of the feature request. An interesting point is to look at the 'initial quality state' (only *C criteria) of the feature requests.

*1) QC1 *C:* The *C criteria for QC1 are to have a summary, description and product version. These three fields are mandatory in the issue tracker that was used in all three projects. Therefore all feature requests that were considered in the experiment score 100% on 'initial QC1 quality'. However, for 'final QC1 quality' they score on average 100% for basic elements (relative importance is also a mandatory field), but only 54% for required elements (QC1.2). This means that on average almost half of the required elements are missing.

*2) QC2 *C:* All QC2 criteria are *C. As such 'initial QC2 quality' can not be distinguished from 'final QC2 quality'.

*3) QC3 *C:* When we only average the seven creation-time criteria for [QC3] we see a clear difference in scoring:

| | | Mylyn Tasks | ArgoUML | Netbeans | TOTAL |
|---|---|---|---|---|---|
| **Creation-time** | **[QC3] *C** | 83% | 80% | 84% | **82%** |
| **Just-in-time** | **[QC3]** | 80% | 75% | 78% | **77%** |

This is mainly due to the fact that at creation-time we would allow feature requests to not be SMART (QC3.5, see [12]), but just-in-time before development starts the feature request should be clear. A lot of feature requests (on average 46%) were judged as 'Not SMART at all'.

Our experiment demonstrates an example of how the quality of the feature requests at creation-time is different from the 'just-in-time quality' of the feature request. Our claim is that it makes sense to make this distinction when discussing quality of JIT requirements, as we do in our framework. As said, further investigation is needed to distinguish more different time-points in our framework than just *C and *J.

[1]`http://argouml.tigris.org`
[2]`http://projects.eclipse.org/projects/mylyn.tasks`
[3]`http://netbeans.org`

## IV. Discussion

Future work is to further validate the framework, e.g. in industry settings or in agile environments with user stories. We conclude this paper by highlighting some open questions and points for discussion.

## A. *Oral versus Written JIT Requirements*

As said, JIT environments rely heavily on face-to-face communication. In open source projects, because of their distributed nature, this is usually not possible. Instead these projects use other means of informal/undocumented communication like mailing lists, chat sessions, personal emails. We have studied feature requests in open source projects from the documented discussion in the issue tracker. An open question is how much of the discussion about the feature request actually took place outside of the issue tracker. Did we have a complete picture of the feature requests we considered by looking at the issue tracker only? And more in general for JIT requirements: how much of the requirement is elaborated in oral form and how much is elaborated in written form? What should be reasons to document oral communication? What would be good means for doing this? These questions could be answered e.g. by a more in-depth study of mailing lists in open source projects, by interviewing more JIT practitioners or by closely following a JIT team in their way of working.

## B. *OSS versus CSS*

According to [16] "closed source software bug reports and feature requests and the process for managing them look much like those for open source software". This would mean that our findings and list of quality criteria for feature requests in open source projects (OSS) would also be applicable to feature requests in closed source projects (CSS). A detailed comparison of OSS feature requests to CSS feature requests could confirm this.

## C. *Why Early Verification?*

We claim that informal verification of JIT requirements is useful. We did however not investigate how much time or effort the use of our framework would constitute for the team. As said, there is some evidence that early verification of requirements quality contributes to a higher software product quality but what exactly is this contribution in JIT projects? After all, incorrect requirements should be spotted early on because of short iterations and should be easier to correct because of high customer involvement. The practitioners we interviewed however stated that, at the least, early verification helps to save time and effort in implementing the requirements. This stems from the fact that even if the work can be redone in a next iteration to correct wrong implementations, it still pays off to do it right the first time. This corresponds to the findings of Fitzgerald et al. [17]: there are many failures in open source feature requests that correspond to effort being wasted. If our framework is incorporated as a 'checklist' into the natural process of the team, e.g. in their 'Definition of Done', we think that it should not constitute a big effort. Experienced team members should not even need the checklist to produce good quality requirements. However the checklist then still holds value as 'contract' of what the team sees as good quality requirements. This contract should be a 'living' item: if requirements problems are surfacing in the team process, the checklist should be updated with additional checks to prevent this type of problems from occurring.

## D. 'Just-in-Time Quality'

Our observation is that a quality framework for just-in-time requirements should include a time dimension. In the current version of our framework we only distinguish between 'creation time' and 'just-in-time'. Creation time is a clearly defined moment: the creation of the requirement or the requirement part. However, what just-in-time means for quality of JIT requirements is not strictly defined. 'Just-in-time' can be just before development starts or just before the requirement is communicated to the customer for the first time. We would like to better understand what constitutes 'just-in-time' for different JIT environments by collecting experiences from different JIT teams. Does it indeed make sense to distinguish initial quality from just-in-time quality by including a time dimension? If so, which time-points should we consider?

## E. Feature Requests versus User Stories

Ernst and Murphy [4] mention two types of JIT requirements: features and user stories. In our research we studied open source feature requests because of their on-line public availability. We did however interview practitioners that work with user stories, studied literature on user stories and have personal experiences in working with user stories. According to Leffingwell [11] user stories are "throw-away artifacts" and only serve to guide conversations, so it would not make sense to spend much time on their verification. However, we saw many cases were user stories are not only documented with one brief sentence but also include more detailed specifications. In addition, many practitioners confirmed that user stories in their practice are not thrown away. This led us to belief that our framework is also applicable to situations were teams work with user stories. As said before the use of the framework should be an integrated part of the agile team process. When and how would it be useful to perform informal verification of user stories in agile projects?

## F. Specification by Example (SBE)

Specification by example [18], also known as example-driven development, executable requirements or acceptance test-driven development is a way of working in which requirements are not documented as abstract statements but as a set of examples (e.g. 'Given ... when ... then ...'). SBE or similar methods are often used in iterative or agile development. If the requirements are specified in such a way, the need for informal verification seems to be less: the structure of the example is given and the stakeholders can easily validate the requirements by validating the examples. Until now we did not consider this type of JIT projects in our research. Questions are: Is it indeed possible to document all JIT requirements in this way? Which quality criteria apply to SBE? In which situations does informal verification add value in SBE?

## REFERENCES

[1] IIBA, "A guide to the business analysis body of knowledge (BABOK Guide)," *International Institute of Business Analysis (IIBA)*, 2009.

[2] J. Robertson and S. Robertson, "Volere: Requirements specification template," Technical Report Edition 6.1, Atlantic Systems Guild, Tech. Rep., 2000.

[3] IEEE, "IEEE recommended practice for software requirements specifications," *IEEE Std 830-1998*, 1998.

[4] N. A. Ernst and G. Murphy, "Case studies in just-in-time requirements analysis," in *Int'l Workshop on Empirical Requirements Engineering*. IEEE, 2012, pp. 25–32.

[5] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in Human Behavior*, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S074756321400569X

[6] P. Heck and A. Zaidman, "An analysis of requirements evolution in open source projects: Recommendations for issue trackers," in *Int'l Workshop Principles of Software Evolution (IWPSE)*. ACM, 2013, pp. 43–52.

[7] ——, "Horizontal traceability for just-in-time requirements: the case for open source feature requests," *Journal of Software: Evolution and Process*, vol. 26, no. 12, pp. 1280–1296, 2014. [Online]. Available: http://dx.doi.org/10.1002/smr.1678

[8] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," Universitaet des Saarlandes, Saarbruecken, Germany, Tech. Rep., August 2012.

[9] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *Int'l Symp. on Foundations of Software Engineering (FSE)*. ACM, 2008, pp. 308–318.

[10] P. Heck and A. Zaidman, "A quality framework for agile requirements: A practitioner's perspective," Software Engineering Research Group, Delft University of Technology, Tech. Rep. TUD-SERG-2014-006.

[11] D. Leffingwell, *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise*, 1st ed. Addison-Wesley Professional, 2011.

[12] G. T. Doran, "Theres a smart way to write managements goals and objectives," *Management Review*, vol. 70, no. 11, pp. 35–36, 1981.

[13] B. Wake, "INVEST in good stories, and SMART tasks," http://xp123.com/articles/invest-in-good-stories-and-smart-tasks/, 2003, [Accessed Nov-2013].

[14] P. Heck, M. Klabbers, and M. C. J. D. van Eekelen, "A software product certification model," *Software Quality Journal*, vol. 18, no. 1, pp. 37–55, 2010.

[15] K. Power, "Definition of ready: An experience report from teams at cisco," in *Agile Processes in Software Engineering and Extreme Programming*, ser. Lecture Notes in Business Information Processing, G. Cantone and M. Marchesi, Eds. Springer International Publishing, 2014, vol. 179, pp. 312–319. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-06862-6_25

[16] T. Alspaugh and W. Scacchi, "Ongoing software development without classical requirements," in *Int'l Req. Engineering Conference (RE)*, 2013, pp. 165–174.

[17] C. Fitzgerald, E. Letier, and A. Finkelstein, "Early failure prediction in feature request management systems: an extended study," *Requirements Engineering*, vol. 17, no. 2, pp. 117–132, 2012. [Online]. Available: http://dx.doi.org/10.1007/s00766-012-0150-7

[18] G. Adzic, *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications, 2011.