Software Requirements Specification in Global Software Development – What’s the Difference?

Frank Salger
Capgemini
Carl-Wery-Strasse 42
81739 Munich
frank.salger@capgemini.com

Abstract: Global software development (GSD) is becoming the standard approach in the software industry. In software projects, the quality of the software requirements specification (SRS) is a crucial success factor. However, little is known about how to write high quality SRS within GSD projects. The question is: “What notation of quality shall we employ for SRS that are used in GSD projects?” In this paper, we use the IEEE Standard 830 as a basis to answer this question. This standard defines several quality attributes of ‘good SRS’. For each of the quality attributes, we explain its specific role in GSD and point out means to achieve compliance with it. We argue that the negative impact in not complying with the quality attributes is exacerbated in GSD. Combining these results with our previous work on inspections, we derive our main conclusion: The quality attribute which has to be treated most carefully in GSD is ‘external completeness’. The paper includes a concise checklist to assess SRS in GSD projects. The results thus provide tangible benefits to practitioners. We also point out research questions with high relevance for the industry.

1 Introduction

It is well known that requirements engineering (RE) has a strong influence on project success or failure [SGI03, DO05]. This influence becomes even stronger in global software development (GSD) where teams in geographically different sites work together as one team in order to create complex software systems. In fact it is often concluded that requirements engineering is one of the main challenges in GSD (see e.g. [PAE03]). This conclusion seems to be reasonable as the software requirements specification (SRS) usually constitutes the pivotal work product by means of which much of the communication between the different sites which are involved in GSD takes place [HPB05, HM03]. Taking into account the increased importance of offshoring and outsourcing in modern software development [WFV06, He07a] it is surprising that the question: “What notation of quality shall we employ for SRS that are used in GSD projects?” has not been explicitly addressed yet.
Now, why is RE so difficult in GSD projects? This question can be answered along the main differences of GSD to co-located projects. These specific GSD-challenges are:

**Geographical distribution:** Geographical distribution impedes ‘whiteboarding’, that is, standing next to each other, drawing models and the like at the white board to clarify specification issues or search for flaws in requirements. It also hinders to use ‘rich’ communication, in particular mimics and gestures. Probably the biggest impact of geographical distribution might come from missing ad hoc meetings at the water cooler, at the coffee machine or at lunch. It is there, where lots of information is exchanged, even without the team members noticing it [He07b]: By now, it is well known that we cannot write down all information that is necessary to build a system in a SRS and that large portions of the missing information is transferred by means of such unplanned “ad hoc meetings” (see, e.g., [Da05]). **Different language:** Using different languages involves repeated translations. This ‘language hopping’ opens a wide space to introduce all kinds of ambiguities. **Different cultures:** The effect of different cultures is often explained along Hofstede’s cultural dimensions [Ho10]. For example, Indians are considered to be less assertive than Germans. This might lead to situations, where an Indian programmer does not point out flaws of a SRS, in order not to let the German requirements engineer ‘loose his face’. **Time zone difference:** Although time zone difference can be advantageous for several disciplines like testing (see, e.g. [HH07]), other disciplines are negatively affected by reducing the overlap of common working hours. With respect to RE, there simply might be less time available for discussions between the onshore requirements engineer and the offshore developer. **Different engineering backgrounds:** Different engineering backgrounds might entail usage of different processes, methodologies and tools. Apart from slowing down the lead time to productive work, different terminologies can seriously affect a distributed team by opening the door for ambiguities and misunderstanding [ED01], [B et al 09].

Regarding the immense challenges that are newly introduced in GSD settings, surprisingly little research has been conducted on the topic of requirements engineering in GSD. While there exist some valuable research results on process related aspects of writing software requirements specifications in GSD projects (see, e.g., [G et al 08]), the author is not aware about discussions, what “quality” of a SRS would mean in GSD projects. To tackle this research gap, we here leverage the IEEE Standard 830 - Recommended Practice for Software Requirements Specifications [IEEE98], a well known and largely accepted definition of ‘quality’ of a SRS. Among other information, the standard provides a characterization of good SRS. There, a SRS is considered to be of high quality if it satisfies a number of quality attributes, i.e., if it is:

- **Correct:** “An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.” [IEEE98]

- **Unambiguous:** “An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation.” [IEEE98]
• Complete: In [IEEE98], three aspects of ‘completeness’ are distinguished: A SRS is considered complete, if and only if a) All significant requirements are captured (called ‘external completeness’ hereafter). b) The definition of the responses of the software to all realizable classes of input data in all realizable classes of situations is given (called ‘internal completeness’ hereafter). c) Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure are given (also subsumed under ‘internal completeness’ hereafter).

• Consistent: “Consistency refers to internal consistency. If an SRS does not agree with some higher-level document, such as a system requirements specification, then it is not correct.” [IEEE98]

• Ranked for importance/stability: “An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.” [IEEE98]

• Verifiable: “An SRS is verifiable if, and only if, every requirement stated therein is verifiable. A requirement is verifiable if, and only if, there exists some finite cost-effective process with which a person or machine can check that the software product meets the requirement.” [IEEE98]

• Modifiable: “An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.” [IEEE98]

• Traceable: “An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.” [IEEE98]

In the remainder of this paper we investigate the question, what role the above described quality attributes play in GSD projects. We provide a qualitative valuation which quality attributes are even more crucial to be obeyed than in co-located projects. This valuation is derived from various project dig-outs, numerous interviews with senior managers and architects, as well as literature research and empirical research conducted at Capgemini [F et al 07]. Single aspects of our results can already be backed by existing research literature. The main contribution of this work thus is a consolidated overall view on SRS quality in global software development projects based on the IEEE-standard 830. We believe that the article is beneficial for practitioners to get a concise overview on the specifics of writing good quality SRS in global software development projects. It also contributes to fill an existing knowledge gap identified in [DE06]: “[…] more information has to be gathered to establish a connection between quality assurance techniques and certain quality attributes.”

The next section describes the context of this work: The company, the system type considered and one of the GSD projects which where the basis of this research. In section 3 we scrutinize the quality attributes that are explicitly stated in the IEEE-standard with respect to its role in GSD.
We also discuss quality assurance techniques that help to comply with these attributes. Further, we reframe the discussion into a concise checklist which we hope will serve the practitioner to set-up or assess his own GSD projects. In Section 4, we give a short summary, draw our conclusions, indicate our future work and point out some research questions with relevance to the industry.

2 Context

Capgemini employs about 95000 people in more than 30 locations around the world. With a global network of more than 40,000 experts, Capgemini’s Technology Services (TS) is dedicated to helping clients design, develop and implement technical projects of all sizes through the creation of project architecture, software package implementation, application development, consulting in IT technologies and innovative solutions. For more information: http://www.capgemini.com/services-and-solutions/technology/. The business information systems developed at Capgemini Germany support core business processes of our customers (and as such are critical to the business of our customers) but are usually not critical to life. This paper focuses on custom solution development projects of large scale business information systems. Business information systems can consist of several hundred use cases, user interfaces and domain entities. Complex concepts for print output, batch processing, archiving, authorization and multi-tenancy are almost always necessary to meet the requirements of the customer. We now shortly describe one such custom solution development project. This case study will subsequently be used to explain how specific techniques have been to achieve a high quality SRS.

Project Alpha: In this project Capgemini Germany was engaged to build the retirement management application of a large German insurance company and the migration of the data of the legacy application landscape into this new system. The project started in 2003 and the last customer release went into production mode in 2007. In the average, about 100 developers worked in this project. The project leveraged offshoring of the design, the implementation and testing. The project was distributed over three sites: a) Munich, Germany, where requirements engineering, analysis & design, implementation, testing and deployment took place. b) Zurich, Switzerland, where analysis & design, implementation and testing took place. c) Wroclaw, Poland where design, implementation and testing took place. The final SRS consisted of several hundred use cases and user interface specifications, as well as complex cross-cutting concepts, e.g. for multi-tenancy or the time dependent handling of business domain objects. A major challenge was the very strict performance requirements on data-intensive batch processes. Moreover, business rules where very complex since numerous legal exceptions had to be considered. Instead of using a commercial requirements management tool, the SRS was maintained in a custom-build tool. This allowed us to extensively generate documentation and configuration code from the SRS and helped to keep the SRS and the design in sync.
3 Quality of SRS in Global Projects

In this section we investigate how the above discussed quality attributes are affected by GSD. We will argue that meeting all quality attributes becomes even more important in GSD than in co-located project setting. For each single quality attribute, we discuss why it becomes more important in GSD. We then succinctly propose techniques how to achieve the quality attribute in GSD, based on our experiences. Some of these techniques are well known and should also be applied in co-located projects. However, we also highlight a couple of techniques that are less known and especially helpful in GSD. We describe the techniques as “checks”. The checks are independent of each other and can be used separately. At this moment, we have no empirical data about interrelation of the techniques. E.g., we cannot yet say, whether one technique is more effective than another in order to satisfy a specific quality attribute. Thus, we suggest to use all proposed techniques in order to increase the likelihood of achieving the respective quality attribute. We cannot claim yet, that the list of proposed techniques is complete.

Quality Attribute: Correct, Unambiguous, Complete, Consistent

Why more important: We discuss these four quality attributes together because they are uniformly affected by the GSD-challenges: First, geographical distribution impedes the intensive communication necessary to resolve these kinds of defects in SRS. Even working interactively in front of a whiteboard, it can be daunting to resolve flaws in SRS - let alone by using only electronic communication tools. Writing SRS in a different language than ones native language is very challenging and often leads to ambiguities in SRS. These issues often lead to delays and quality problems with the resulting system. Time zone differences leave less time for the onshore requirements engineer and the offshore developer to discuss issues with a SRS. It also entails asynchronous working which makes it more difficult to quickly reveal ambiguities.

Check:

Constructive quality assurance applied? Correctness and completeness largely escape classical appraisal and failure-based quality assurance techniques as suggested by the results from other researchers [Da05] as well as our own previous work [SEH09]. It is thus especially important in GSD to embrace systematic constructive quality assurance methods in order to arrive at correct and complete SRS baselines from which to start offshore development. Project Alpha used Joint Application Development sessions to gain a common understanding of the system to be built.

Incremental development used? Incremental development and continuous integration are mentioned as a technique to tackle the problem with correctness and completeness of SRS: Increments allow to ‘fail early’ and gather early feedback based on tested increments. Missed core requirements often lead to substantial rework of architectures, design and large portions of low-level code. Our experience is that such massive changes are much more difficult to be executed in GSD as compared to co-located projects. Early increments often point out missed requirements and can even substitute detailed specifications to some extend.
In project alpha was developed in four major increments and several minor increments. This approach especially helped to evolve the cross-cutting concepts of the SRS, and avoided that teams of later increments made the same mistakes as their team mates in previous increments.

Test cases derived, checked and supplied to the offshore team? Deriving test cases for SRS often point out flaws, omissions and ambiguities: It is simply not possible to derive test cases for underspecified requirements. In project Alpha, we heavily used this technique to improve the quality of the SRS. The most valuable improvement was gained by having the end users specify (acceptance) test cases, where the requirements engineer conducted walkthroughs on the SRS based on these test cases. This approach surfaced a couple of serious flaws in the SRS. This technique alongside with handing over the test cases to the offshore team seems to be especially promising in GSD be acceptance test driven development [SE10, SSC04].

Quality Attribute: Ranked for importance/stability

Why more important? These quality attributes are indirectly impacted by geographical distribution. Put it simple, this kind of distribution makes it more difficult to manage a project (see, e.g., [Cu08]). This in turn can make it necessary to implement management techniques like earned value, which in turn is based on a concept of ‘value’, or ‘importance’. Further, some authors also advise not to offshore the development of business critical functionality (see, e.g., [BK05]), which should also be an aspect of the definition of importance. A same argument is often used with respect to stability: Due to hindered communication in GSD, it seems to be unreasonable to offshore the development of highly unstable functionality, as instability necessarily leads to frequent communication. Communicating changes and the necessary re-planning are more difficult to handle due to distance [S et al 06].

Check:

Business domain itself stable? Which parts of the SRS will be offshore? Are these parts related to highly volatile business subject matter? If so, the project should be able to answer how they will manage frequent changes to SRS and how they will cascade these changes to the offshore development team. Although the business domain of project Alpha was highly complex due to numerous legal exceptions, it was quite stable. Thus, stability of the business domain did not pose any extra challenges.

Change request management process appropriate? Highly related to the first check, defined processes for change request (CR) management are always especially important in GSD. In particular, it is necessary to implement a process which can handle the many issues that come up during fine design and programming [S et al 06]. In project Alpha, a multi-level CR management process was implemented. Minor changes (triggered from then development team or the user group) could be authorized by the project managers (client side and project side) themselves, but all significant changes of course had to pass the executive change management board.
SRS baselined and signed-off? It is always important to define clear SRS baselines to start development from, and always difficult to judge when to draw the baseline [Wi03]. However, in GSD a good baseline becomes even more important as every change or update to the requirements package must be propagated to a globally distributed site. Using modern requirements engineering tools with full access of all sites, this problem is to some extent mitigated. But still, it remains challenging to communicate changes of requirements or packages thereof remotely. In project alpha, parts of the SRS (so called ‘work packages’) where first validated by the client. After that, the responsible requirements engineer together and the offshore development team conducted a so called ‘work package handover checkpoint’ to ensure that the work package was complete from a developers point of view.

Quality Attribute: Verifiable

Why more important? In the end, it is the client who will verify that the system satisfies the SRS as stated. To do so, he of course must understand the SRS. However, a dilemma arises, as “representations that improve the requirements specification for the developer may be counterproductive in that they diminish understanding to the user and vice versa.” ([3], see page 5). Now, some authors argue that SRS must be more detailed in GSD settings in order to support the work of the offshore developer {attrition, less business context} (see, e.g., [S et al 06]). Basically, this is an indirect consequence of geographical distribution as they cannot participate in the requirements workshops and miss the information shared in ad hoc meetings. Thus, the dilemma is exacerbated in GSD, as adding even more ‘developer oriented’ information further diminishes the probability that the client understands the SRS. Hence, in principle, the treatment of the quality attribute ‘verifiable’ must be rethought in GSD. Luckily, most often, ‘inverifiability’ is a consequence of under-specification. This can be reduced by adding detail which is required by client and developer alike.

Check:

Test cases derivable? Testing is the prime technique to verify requirements. Testing relies on test cases. If these are not derivable, a requirement is not verifiable. In project Alpha, the requirements engineer started to derive test cases already during the software requirements specification phase. Moreover, the business subject matter experts reviewed these test cases which also helped to improve the specification [SE10].

Weak phrases avoided? Verifiability is closely related to absent ambiguity. And a lot of ambiguity can be avoided if so called weak phrases are eliminated. In project Alpha, a rather mathematical and semi-formal approach was taken to specify business logic. This helped to avoid weak phrases.

Precise quantification used? Especially for non-functional requirements, precise quantification is necessary. One possibility to achieve precise quantification is to use highly structured (yet practical) languages, like, e.g. Planguage [Gi05]. In project Alpha, the technical chief architect reviewed all non-functional requirements and ensured that they were precise enough to derive a valid software architecture.
Quality Attribute: **Modifiable**

**Why more important?** Of all quality attributes discussed in this article, this is the one which is least impacted by GSD. Nevertheless, *different engineering backgrounds* triggered discussions in some of our projects. Sometimes, the offshore team wanted all information related to a use case be bundled together with the use case. This eased the task of allocating work to the developers. However, this would have resulted in redundancy (e.g., copying business rules to incorporate them into different ‘use case bundles’). We refrained from this approach as this clearly would have made the SRS less modifiable, accepting a potential reduction in development efficiency. If we consider the ability to propagate modifications to the according stakeholders than *geographical distribution* also impact this quality attributes, as distribution reduces the developer awareness of requirement modifications [Da07].

**Check:**

Sufficient tool support employed? In most nontrivial software projects there are too many requirements to be left unmanaged. Specifically in GSD, tools should support the notification of requirements changes to the involved people. Vendors of requirements management tools acknowledge the trend towards GSD and take provision to update their tools to these settings. Project Alpha used a requirements management tool. But, this tool did not support direct notification mechanism to flag requirements changes to developers. However, the role of a ‘topic responsible’ was introduced, who was responsible to propagate the changes of requirements of his topic to the team. In this project, this approach proved to be effective as well.

Quality Attribute: **Traceable**

**Why more important?** Traceability plays an important role in GSD [L et al]. Traces between artefacts on the same level of abstraction are important from a developer point of view. For example, a clear mapping of the domain objects referenced by the use cases onto the logical data model might be helpful for the offshore developers. But traces between higher level requirements and, e.g., use cases are also especially helpful for offshore developers as they usually did not participate in the requirements workshop and thus miss the overall view on the business subject matter. As ‘higher level’ requirements (like features or business processes) often provide more overview that the ‘lower-level’ uses cases, a developer can use the traces to gain additional information on the business subject supported by the use case he implements.

**Check:**

Traces between high level requirements and SRS exist? Check that the constituent parts of the SRS can be traced back to the high level requirements, and vice versa. For example, it is usually necessary to trace high level features down to the use cases that ‘implement’ them. In project Alpha, a lightweight excel-based tool was used to implement traces between high level requirements (business processes) and system-level use cases. This proved to be sufficient, as the mapping between business processes and system-level use cases was usually a one-to-many relation.
Traces between SRS primitives exist? The appropriateness of this kind of tracing directly impacts developer efficiency: As he implements the functionality he will have to navigate through the SRS in order to get all necessary information. Especially important are the traces between use cases and user interfaces, as well as between use cases and business rules. In project Alpha, a specific naming scheme was used to map SRS primitives to each other (like, e.g., use interfaces to use cases). Again, this proved to be sufficient due to the rather simple mapping between the different primitives.

Traces between SRS and downstream artefacts can be set up? Already when a SRS template is made up one should think about traceability to downstream artefacts that will be spawned by the SRS (like design artefacts, test cases, etc.). The critical decision is, to what level of detail unique identifiers (IDs) will be used. For example: Will use cases be uniquely identified? Or will steps of use cases be uniquely identified? In GSD, IDs play an important role, as most communication happens over telephone and email. There, it is a tremendous advantage if one can clearly point about what he is talking using IDs. In project Alpha, IDs where used down to the level of use case steps and business rules. This allowed to map fine-grained functional parts of the specification onto design primitives, which proved to be valuable especially in the long maintenance phase.

Although project Alpha applied various of the discussed techniques, it still faced considerable requirements engineering related challenges. Probably the most critical challenge was the issue of knowledge transfer: Although a rather detailed and granular SRS has been created, the offshore developers did sometimes miss information necessary to implement the system. Sometimes the developers missed even more details for specific complex requirements. On the other hand side, they sometimes missed the overall context. We believe that answering the question of right granularity of SRS is especially critical in GSD projects. In Table 1, we present a checklist condensed from the discussion above. We believe that this checklist can be valuable to assess GSD projects – also by practitioners of other companies. In the table, “!” flags a challenge which is exacerbated by global distribution, and “*” flags aspects which might help to address the respective challenge. These aspects should be carefully assessed in GSD projects to assure project success.

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Why more important?</th>
<th>Check!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Correct Unambiguous Complete Consistent</td>
<td>! Distance impedes communication which makes resolving non-compliance to all these quality attributes more difficult ! Non-native language can introduce ambiguities ! Time zone differences leave less time so resolve potential issues</td>
<td>* Constructive quality assurance applied (like, e.g., joint application development, prototyping)? * Incremental development used? * Test cases derived and checked?</td>
</tr>
<tr>
<td>Ranked for Importance/ Stability</td>
<td>! Distribution makes project management more difficult and demands for advanced techniques like, e.g., earned value analysis</td>
<td>* Business domain stable? * Change request management appropriate? * SRS baselined and signed-off by</td>
</tr>
</tbody>
</table>
| Verifiable | ! Instable business domain triggers frequent changes to SRS, whose propagation is more difficult in distributed teams | customer before development starts?  
* Allocation of all ‘work packages’ for offshore development rationalized? |
|---|---|---|
| Modifiable | ! If a requirement is not verifiable it most often is underspecified. This will lead to problems during development, which makes lots of communication necessary. But communication is impeded in GSD. | * Test cases derivable?  
* Weak phrases avoided?  
* Precise quantification used (e.g., Planguage)? |
| Traceable | ! Distribution limits awareness of offshore developers for modifications  
! Distribution makes propagation of modifications more difficult | * Sufficient tool support employed?  
* Offshore developer have direct access to requirements management tool?  
* Traces between high level requirements and SRS exist?  
* Traces between SRS primitives exist?  
* Traces between SRS and downstream artefacts can be set up? |

Table 1 SRS Checklist for Global Software Development

4 Summary, Conclusion and Future Work

We now briefly summarize the results of this paper before we draw our conclusions. By indicating our future work, we will close our discussion. **Summary:** This paper offers preliminary answers to the question “What notion of quality shall we employ for SRS that are used in global software development (GSD) projects?” We based our discussion on several quality attributes that characterize high quality of a SRS as suggested by the IEEE-standard 830 – Recommended Practice for Software Requirements Specifications. For each of those quality attributes, we explained how the specific GSD challenges exacerbate the relevance of complying with the quality attribute. Further, we discussed various techniques which help to achieve compliance with the quality attributes. We also explained how these techniques where applied within a large GSD project executed at Capgemini CSD GSA. Finally, we derived a concise checklist which can be leveraged by practitioners to set-up or assess their GSD-projects. After having discussed the effect of GSD on the quality attributes of the IEEE-standard, we can now put these results in the context of our previous work and the results of other researches and draw our major conclusions. **Conclusion:** In our previous work, we argued that not all quality attributes are equally effectively checked by inspections [SEH09]. Basically we found that the quality attributes “internal completeness”, “consistency”, “understandability”, “verifiability”, “modifiability”, “traceability” and “ranked for importance/stability” can be very effectively checked by inspections. Less effectively but sufficiently addressed is the quality attribute “correctness”. However, only insufficiently addressed is the quality attribute “external completeness”.

180
The main reason is that omissions of requirements are very difficult to spot for any reviewer: A peer reviewer often has the problem of being ‘routine-blinded’. In turn, a project external reviewer is not as intimately familiar with the business domain as to judge completeness [Da05]. As we here consider the SRSs of custom business information systems, the project team members are sometimes the only experts in the company for the according business subject matter. Apart from that, customer needs constantly change [Da05]. Not only appraisal-based quality assurance techniques fail to assess ‘completeness’. To worsen the situation, other quality assurance techniques like testing do often not catch requirements omissions, as (acceptance) tests run only against those requirements which have been gathered and which are actually written down within the SRS.

But requirements omissions are not only hard to spot. They are also the type of requirements defects, which are most expensive to correct. We thus conclude that ‘external completeness’ is probably the most important quality attribute to strive for in GSD. From our project experience we did not experience that completely new quality attributes for SRS show up in the context of GSD. On the other hand, we have argued that compliance to the SRS quality attributes becomes even more important in GSD, and that failure to do so exacerbates the risk of project failure – even more than in co-located projects. Thus, requirements engineering in GSD seems not to be different from what a software engineer has to do. The difference is that high quality requirements engineering is even more critical in GSD than in co-located projects!

Future Work: In our future work we are going to investigate the question, how SRS should look like in agile GSD projects, taking into account challenges like long maintenance phases of business information systems. For our near term work we plan to provide an empirical basis for the (qualitative) results offered in this paper. Subsequently, we also want to extend this study on the quality characteristics which are only implicitly mentioned by the IEEE-standard 830. Finally, we want to formulate a research question of which we believe that the answers would be highly beneficial to the industrial practice: Writing SRS is always a balancing act of keeping the SRS readable for both, the customer and the developer. It is sometimes argued that SRS in GSD must be more detailed than in co-located projects. An interesting research question would thus be: “Does GSD intensify the problem of keeping a SRS readable for all involved stakeholders?” And – if yes – how to tackle this issue?

References