Open Source Software: Leveraging Software Quality in the Industrial Context

Winfried Seidel and Christoph Niedermeier

Siemens AG, Corporate Technology, CT SE 2
Otto-Hahn-Ring 6, 81730 Munich
{Winfried.Seidel,Christoph.Niedermeier}@siemens.com
WWW home page: http://www.ct.siemens.de

Abstract. For years, the software industry has been suffering from problems subsumed under the term "software crisis". While traditional development methods were not able to provide a cure, open source software (OSS) seems to be an effective approach to reduce product cycle-times and decrease development costs. Moreover, OSS has great potential to achieve high software quality and promote creation of reusable software, provided that proper organization and infrastructure are in place. In this paper, we look at different facets of software quality, and analyze possibilities and limitations of OSS methodology regarding the achievement of certain quality levels. We further consider different forms of cooperation between enterprises and OSS communities and examine the prerequisites for successful adoption of OSS methodology. We come to the conclusion that to benefit from the advantages of OSS, enterprises have to adapt their internal organization and establish a development style that is less formally organized and more strongly based on self-organization principles.

1 Introduction

Many organizations, not only in the industrial context, have to deal with a problem area that was coined earlier under the term "software crisis". To a certain extent many software projects still face problems under given constraints. Among other things the issue of quality is on permanent focus. Alongside to the discussions about shortcomings in the proprietary software world, a development took place that was in its first instance largely driven by the GNU project and the emergence of the Linux operating system [1], [7]. The quest for finding the driving success factors for the Open Source Software (OSS) model is still going on.

Therefore its seems to be a logical follow-up step to take advantage from OSS principles in software projects to exploit synergies between otherwise loosely coupled groups within an organization and possibly also improve software quality. In these days many IT groups are willing to use mature OSS software as a basis for products and services and try to cooperate with the OSS community to create higher quality software by benefiting from the underlying principles of the OSS process, like invocation of a critical mass of reviewers from peer developers and users as well.

New ways of interrelationships appear where companies cooperate with others in OSS fashion in order to create widely recognized standards for industrial application domains. Even today a huge percentage of the world-wide OSS is already developed in or with the aid of companies.

2 Current state of commercial and OSS software

Although the term “software crisis” is around for some time now in software intense organizations, it is not stressed in the public as explicitly as in earlier years. Nevertheless the problem area addressed by the term “software crisis” still exists, even if the focus may have shifted between key problem areas.

Challenges to master

Software development in the industrial context faces the following challenges:

- Product cycles are getting shorter and shorter,
- Products are getting more complex, like the number of features is increasing; complexity is difficult to manage; late requirements occur,
- Necessity to differentiate from competitors; too many components have to be controlled; effort is not focused on added value,
- Large enterprises with highly vertical internal organization experience a knowledge sharing problem.

In the following we are going to elaborate these challenges in more detail:

In a market driven environment product release-cycles are getting shorter while product development efforts should at least be kept constant or even be decreased. On the other hand the products are getting more and more complex. The number of features to be addressed by a software driven product is increasing from generation to generation, while the user interface has to hide the underlying complexity as much as possible. The effort to manage increased complexity especially in late phases of a project, like system integration, and acceptance test has to be obeyed as well. Another phenomenon is that additional requirements or requirement changes are increasingly faced late in the development cycle.

In order to differentiate from competitors many different variations of the same core product need to be handled by the executing development organization. This leads to an increased number of software configurations to be tested and maintained. Dealing with variants is not per se a business value, but adds costs in the first instance.

Often large enterprises are organized as vertical slices, where each slice represents a specific business field. Segmentation of companies, slicing into independent profit centers leads to a large extent of parallel development work and associated high costs. Even if there is potential for collaborative work on the same issue, organizational measures are very often not in place to foster a productive knowledge-sharing behavior. Driven by today’s business expectations it is quite possible that such a situation occurs. Sharing common solutions in the first place adds some overhead and

in the mid to long turn ‘returns’ are not always backed by convincing facts and numbers in the individual case.

Observations in industrial environments

Regarding the above mentioned issues, it is increasingly difficult to find a match for all these challenges at the same time in addition to delivering stable or even improved software quality. This situation is faced by developers and managers alike.

Developers usually like to design and build a solution for a certain technical problem. It is not unlikely that developers invest some part of the effort to more generic solutions than required on the actual development task. It also happens that side-effect tasks are performed where tools or tool enhancements are developed that are actually not part of the original development goal. Furthermore, in the initial phase of a project effort is spent to become familiar with already existing solutions or building blocks that could aid in developing the own solution.

From the management point of view the development process should follow a well defined process, where initially a plan is worked out which is the basis for continuous fulfillment tracking. The planning is the basis for the time schedule and the required resources and budget. With these measures project managers are able to track progress even if they don’t understand source code or architectural principles of the subject to be developed.

Observations in the OSS world

As much has been written on OSS development in contrast to proprietary software development (see [1] and [7]), here we only summarize key areas of relevance in this context:

- Continuous development process with small increments replaces waterfall model and coarse grained incremental processes,
- Large number of reviewers detect software bugs quickly and contribute fixes (e.g. consumers become “prosumers” i.e. consumers that participate in product development when working with the software),
- Critical mass of contributors and reviewers is available for software that is widely used, but not for specialized applications,
- OSS as an effective approach to reduce cycle-time and decrease design, implementation, and quality assurance costs for certain types of software, like infrastructure software: OS, compilers and language processing tools, editors, distribution middleware, development tools,
- Topics are addressed that are important for the daily work of the developers,
- Self-assignment of developers to tasks.

Especially the last item of self-assignment of tasks can be seen as a major success factor for the OSS methodology. This is very much in contrast to usual proprietary development situations where the project management authority assigns tasks to the individual developers according to budget goals and availability schedules.

Summarizing the challenges and problems to be faced in the proprietary software development arena, a serious look at the success factor of the OSS method could help organizations to become more agile and competitive.

Hypothesis #1:
The problems can't be solved by only employing better tools and processes, e.g. for quality management. Rather, it is unavoidable to break software up into re-usable components and promote reuse and thus exploit synergies between different organizations. Furthermore, adoption of OSS components reduces development cycles drastically. As indicated, this kind of development model is not suited for all kinds of software in the same way and with the same efficiency.

One of the key arguments used to defend the OSS idea is the degree of software quality that can be achieved with this method. Since software quality is a major focus and often experienced as a crucial issue in software intense industries this is at least worthwhile to look at. Therefore it is interesting to discuss the question how quality can be assessed and what the status of OSS with respect to quality really is.

3 What is Software Quality?

In many cases the term quality is a topic for debate among people arguing about it. Influenced by the discussions that took place in the European industry during the 1980ies, which was lead by the Japanese quality rush at that time, general understandings were coined. According to [20] different points of view regarding quality can be expressed:

- Transcendental view
- User view
- Manufacturing view
- Product view
- Value-based view

The transcendental view is widely based on experience and perception and it is not possible to measure anything in this view. This view is also not definable but can be recognized if one sees it. It is also not decomposable and rather a property of the whole item under inspection than its parts. The user view can be defined more precisely. It tells how much a given software or system as a whole meets the user's requirements. This is also called quality in use by ISO 9126. It can be measured by operational profiles, so-called usability tests.

The manufacturing view describes the degree to which a software package satisfies formal acceptance requirements. In practical situations this actually is heavily linked to the procedures taken for software and system acceptance testing. Under the manufacturing view quality is defined in a term of defect counts in source files and other metrics such as crash-free operational time or fault free duty cycles. Adopted from the hardware world, the term MTBF - mean time between failures – is frequently used to assess even software driven systems. However, even very well 'manufactured' software does not guarantee that it's usable later on.

The product view of quality is highly linked to the inherent properties of the software product. These properties include non-functional properties like reliability, portability, maintainability, testability, reusability, flexibility, scalability, etc. The value-based view summarizes the user, manufacturing and product view. Under this view quality is measured in terms of benefit vs. cost trade-off. This view is often used when different software alternatives are being evaluated against each other. The questions asked is: "What is the software able to deliver for me and how much do I need to invest in it"?

Idealistic view on quality and effort

Fig.1 highlights a relation between effort and quality. This curve represents a basic relation where quality improvements are cheaper in the in the initial phases and tend to increase only moderately, when the same effort is spent in later phases [19]. In this relation quality is meant to cover aspects from the manufacturing view, the product view and user's view:

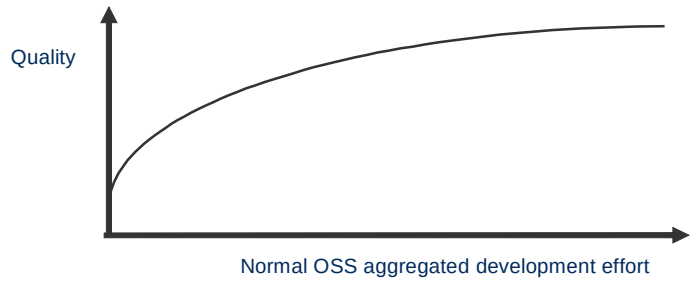


Fig. 1. General idealistic relation between quality and effort

The usual effort[1] and quality relation already reaches a remarkable level for current OSS projects as stated in [1] and [7]. Mainly the quality issue deals with the perceived view of the user community [3], [4].

In the industrial context, quality can be achieved by tackling certain aspects of a given product with objective and reliable metrics and standards. In the software case it's a matter of provable level of confidence that can be attributed to a certain component under a certain set of prerequisites, like execution environment and dependency on the service reliability of other components.

Especially in areas where mission and safety critical systems are being built, radical quality requirements exist. The subject of building so-called dependable systems has been a major topic for more than 40 years in the core computing and telecommunication industry. Obeying these two methodologies – the OSS development process where distinguished levels of quality are reported [18] and the industrial approach where specific procedures and measurement systems are in place

[1] In this context the term effort means the accumulated software design, development and test effort, which naturally also correlates to the accumulated time spent for the considered project.

to achieve a certain level of quality – it sounds obvious that some additional effort is required to gain quality above the level that is already achievable by the current OSS development methods. This relation is caught in the following diagram, again in an idealistic way:

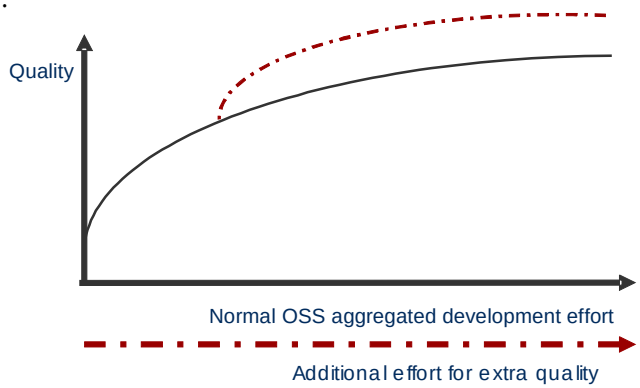


Fig. 2. Relation between quality and effort if additional effort is spent

As indicated in Fig. 2, some additional effort is required in order to improve the quality level beyond what is achievable via the currently observed OSS practices. On the source code level some partly tool driven and formal human process can be applied that in some sense is orthogonal to the lived development process. Thus for achieving higher-level error coverage it is desirable to put a qualified screening process in place where OSS is treated the same way as other software achieving high quality. The advantage can be that starting with OSS already delivers a comfortable quality level [2], [5], 18].

On one hand the quality may be improved on the code level by applying extra effort, like rigorous code inspection, especially for code that might not be under continuous public attention as the Linux kernel. On the other hand there are key areas to be addressed that are also mandatory elements of a successful quality strategy, like sufficient documentation adhering to certain documentation standards, securely archiving the development process (which might be even easier in an OSS environment, where all communication is carried out via the Internet or Intranet), and having certain steps on the timeline where provable assessment and certification steps take place.

Status Quo of quality and OSS

The OSS development process is meanwhile widely known as an effective approach to reduce cycle-time and to reduce costs for design, implementation, and quality assurance [1] – [4]. This is especially true for operating systems like Linux and system infrastructure software like web servers, compilers, editors, middleware, printing systems, file servers and so on. In this realm when talking about the benefits almost all authors writing about OSS emphasize high quality of OSS as a key advantage regarding this type of software [4], [5].

The OSS development model is a model where code is released early and often. As Raymond [7] states it, the code must run before it is released to the community and the author must be eager to accept improvements and changes to the code. This way the community is aiding the software debugging and re-factoring process in a way that was not possible to such an extent before the advent of the OSS culture. On the other hand there are also commercial projects that successfully follow a similar development method. However, the involvement of the user community is different.

Since the user has access to the source code in the OSS model she/he can 'look under the hood', monitor, and check if certain parts of the software are implemented correctly. If bugs or misconceptions are found, given the right individual capabilities, these can easily be corrected and contributed back to the originator. Moreover hints can be given to the other members of the community to stop searching for a bug or correcting a bug. However, making parallel investigations on bugs to a certain extent is not seen as a waste of effort, but even to mature the bug manifestation process on non-obvious bugs. A very important consequence of this approach is that OSS is tested in a much greater diversity of environments than any commercial software manufacturer is able to set up [7]. As example, Raymond's fetchmail project listed a number of 800 contributors in the final phase of the project. This means the degree of parallelism in handling and treating bugs is enormous.

There is evidence that the open source development model has the capability to match and even overtake commercial counterparts in a rigorous way when it comes to observing software quality. A study carried out by Reasoning Inc. showed that the LINUX TCP/IP stack (2.4.19) is the best in class with 0.1 defects per KLOC compared to commercial stacks with an average of 0.55 defects per KLOC [18]. Of course one should not extrapolate these numbers to other areas of Linux, e.g. drivers or OSS in general.

The only thing that can't be done with software quality in an OSS process: it cannot be enforced. It just happens intrinsically as a consequence of the underlying terms of operation of the OSS process itself and significantly as a consequence of the commitment of the involved actors. Looking at the basic factors influencing the software quality of an OSS project it can be observed that the maturity and the number of participants of a software project directly impacts the resulting feature set and software quality. Thus people's excellence is a major contributing factor and it turned out that major OSS projects attract many talented programmers and software architects.

Hypothesis #2:
The OSS development model has been proven to be suitable for achieving a certain, in some cases already remarkable level of software quality. Using OSS in more demanding application areas requires additional effort compared to what is usually covered in the OSS process itself, e.g. documentation according to certain standards.

4 Application of OSS methodology in the industrial context

According to the material presented above, the application of OSS principles in the industrial context seems to be a promising approach to increase software quality and, at the same time, reduce costs. Other benefits expected from application of OSS methodology are [21]:

- Knowledge sharing, free flow of ideas
- Promotion of collaboration, reuse, synergies
- Establishment of de-facto standards
- Achieving code quality above average
- Achieving a high level of safety and security
- Reduction of development costs
- Decoupling of software layers

However, high software quality and other benefits can only be achieved if certain prerequisites are met. Therefore, it is essential to investigate the premises of successful OSS development and the implications on internal processes and company culture. As many companies are already actively participating in OSS communities and thus have adopted OSS methodology also internally, there should be a lot of information around regarding these issues. Unfortunately, most of the companies involved do not disclose information about the transformation of their internal processes and culture towards OSS. Therefore, information on possible forms of cooperation between companies and OSS communities as well as on the premises for such kinds of cooperation to be successful has to be taken from the literature.

Cooperation models

In order to get a better understanding for the possible forms of cooperation between companies and OSS communities, we propose four different cooperation models that can be observed in real-life. The models range from a fairly standard way of participating in OSS projects driven by a public OSS community to the adoption of OSS methodology by a restricted community that is not open to everybody.

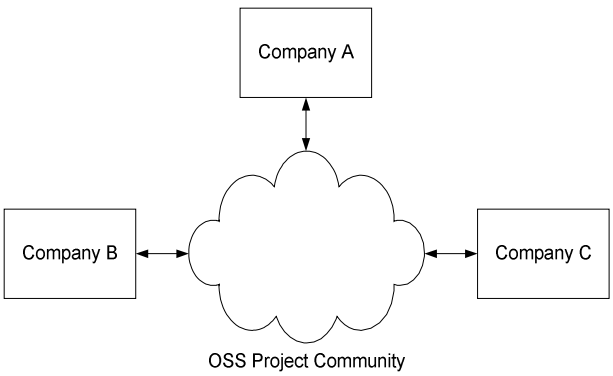


Fig. 3. Standard Model

The “Standard Model” shown in Fig. 3 represents the normal way how companies interact with OSS communities: Individual companies contribute to an OSS project without specifically collaborating between each other. Possible motivations for that kind of scenario include (among others):

- to build and sell proprietary products that are based on OSS
- to sell consulting and support to customers using OSS
- to promote open (de-facto) standards (e.g. for protocols, APIs, data formats) in order to increase the market opportunities for their proprietary products implementing those standards

A more thorough analysis of the benefits of OSS for enterprises as well as some OSS business models are presented in [1], [7], 22].

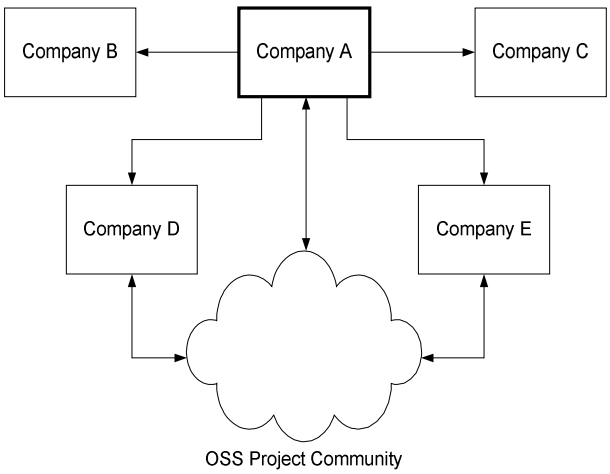


Fig. 4. Open Service Model

The “Open Service Model” shown in Fig. 4 describes a scenario where a company A is developing / extending OSS software and selling special software bundles combined with support / consulting to other companies (B, C, D, E). Company A contributes its work to an OSS community so that some of the customers (companies D, E) can participate in the further development of the OSS software they use. The added value for the customers comes from the services offered by company A rather than from the software itself. An example for this type of scenario is the former Cygnus Solutions (meanwhile acquired by Red Hat) that is selling their GNU Pro Developer Kit to customers and has contributed extensive amounts of work to GCC, GDB and other GNU tools (see [9], pp. 39-46).

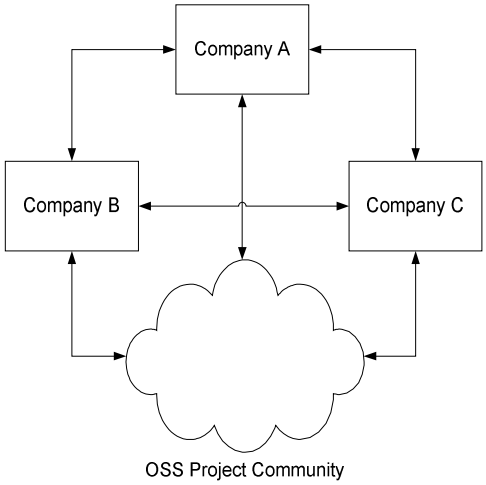


Fig. 5. Open Collaboration Model

The “Open Collaboration Model” shown in Fig. 5 relates to a scenario where several companies are collaborating with each other and contribute the result of their joint work to OSS projects. Individual companies may develop and sell proprietary products based on the OSS software. An example for this kind of scenario is the Mozilla project to which Netscape Inc. and a number of companies developing third-party plugins for the Netscape Navigator contributed their code (see [9], pp. 91-95, [23]).

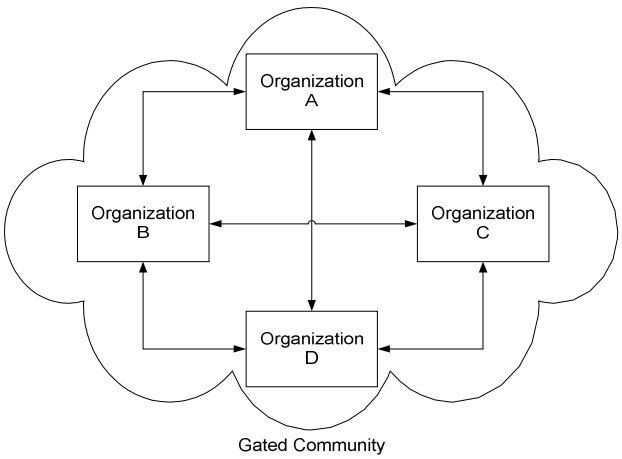


Fig. 6. Restricted Collaboration Model

The "Restricted Collaboration Model" shown in Fig. 6 describes a so-called "gated community" [10], [11]. It is characterized by a number of organizations that use OSS methodology for developing software but control exactly who is allowed to participate. Access to the community either requires paying a membership fee or being part of a larger organization. There are various examples for that kind of cooperation: The OpenPSA community is devoted to developing software for Professional Service Automation (PSA). It is open to all companies joining the PSA consortium [12]. IBM established an internal OSS community called the IBM Internal Open Source Bazaar. This community is open to IBM employees only [13]. Internal use of OSS methodology is recommended by several authors (see [14], [15], [24]), in particular to promote internal reuse of software in large enterprises consisting of a number of fairly independent profit centers.

As has been shown, there seem to be different possible models for participation of companies in OSS communities. Concerning software quality, the question arises if there is a correlation between the cooperation model of choice and the quality level that can be achieved. Which measures have to be taken to improve the quality of software in the different cooperation models? Are forms of cooperation where one or a few companies have a large influence on the direction and processes of the OSS project favoring quality? These issues are yet unsolved but should be investigated more closely in order to be able to fully exploit the benefits of OSS methodology in industrial environments.

Here, we do not want to go further into the particularities of those models, but rather stress an important aspect common to all four models:

Hypothesis #3:
The drivers of OSS projects are individual developers rather than the management. Software quality depends crucially on the skills and motivation of those individuals. Therefore, successful implementation of any of the models requires some kind a culture change inside the company, the reason for that being that the crucial success factors for OSS (trust, self motivation and minimization of bureaucracy) are not necessarily part of established company cultures.

Prerequisites and success factors for achieving quality in OSS projects

A more complete list of success factors, partly adopted from [6], includes:

- Support from the management
- Clarification of legal aspects
- Suitable communication structure / infrastructure
- Skilled and self-motivated developers
- Some key highly knowledgeable major developers
- Two-tier management structure (no hierarchies)
- Light-weight processes (no bureaucracy)
- Relaxed time-tables (no tight deadlines)
- Fun and pride (feeling of belonging to a community, pride in quality)

In particular in large companies, hierarchical management structures, overly bureaucratic processes and strict projects plans are factors that massively impair OSS style development. In order to promote the self-motivation of developers, these factors have to be mitigated. The FLOSS Developer Survey [15] revealed that the primary motivation for developers to start working in an OSS project is the desire to learn and develop new skills. This indicates that developers participating in OSS projects do not need sophisticated incentives and management hierarchies to be motivated, but the freedom to pursue their ideas.

Observations in mature high quality OSS projects revealed [6] that the following aspects regarding the development process itself seem to be favorable to achieve high quality:

- Formal specifications are existing
- Key developers can afford to work full time on a project
- Proper OSS community processes are in place (i.e. gatekeepers are in place, submissions are handled properly)

5 Conclusion

In this paper, we pointed out that the application of Open Source Software principles in the industrial context is a way to overcome the problems usually subsumed under the term "software crisis". Beside shorter production cycles and increasingly complex products, a problem present in particular in large enterprises is the existence of independent profit centers due to vertical slicing according to the business fields. Knowledge-sharing between these separate organizational units is difficult to achieve using traditional organizational patterns because the 'return-on-investment' for such sharing is difficult to prove for individual cases.

OSS style development differs from the traditional model in various aspects. Although it involves less planning and formal project management and is instead based on self-organization and self-assignment of tasks, OSS turns out to be an effective approach to reduce cycle-time and decrease implementation and quality assurance costs. However, this seems to be valid only if a large community of prosumers is participating in a project, and works particularly well for certain types of software (e.g. infrastructure software and development tools).

Proper project organization provided, the OSS development model is able to produce a remarkable level of quality. If even higher quality is desired, additional measures have to be taken. Analysis of OSS projects shows that quality depends crucially on the amount of developer expertise and peer review effort that have been invested. While some OSS projects achieve average quality comparable to commercial software of the same kind, others produce much higher quality (e.g. the Linux TCP/IP stack).

Application of OSS methodology in the industrial context promises a number of benefits, but the implications of OSS on internal processes and company culture have to be understood in order to be able to successfully run OSS projects. We proposed four different cooperation models that range from single companies participating in an OSS project community over certain forms of collaboration between different

companies and OSS project communities to so-called gated communities where access to the community is strictly controlled. All these forms of cooperation require a culture change in enterprises, in particular in large ones, in order to ensure the crucial success factors for OSS, namely trust, self motivation and minimization of bureaucracy. As OSS projects are usually highly self-organized and developers are self-motivated, high software quality is likely to be achieved if a reasonable level of management and supporting infrastructure is provided and negative incentives as tight deadlines and discouragement of participation are avoided.

Further investigation is required to come up with provable measures in order to assess the quality of given software components, regardless whether they are developed in OSS projects in shared project in vertical enterprises or other large organizations. Using OSS components in safety critical, mission critical or high availability environments will therefore be a challenge for those projects. Beside the high quality of software that is already often perceived in OSS projects, experiences from rigorous quality measures should be applicable to OSS components as well. This concludes that more research as well as industrial involvement is required on the quality aspect of OSS.

Acknowledgements

This work was supported by the German Federal Ministry of Education and Research (BMBF) in the project "NOW – Utilization of Open Source Software in Business and Industry", grant number 01ISB04C. The authors thank Prof. Dr. Thomas Wieland for fruitful discussions and hints.

References

1. Feller, J., Fitzgerald, B.: Understanding Open Source Software Development. Addison Wesley Professional (2001)
2. Open Source Quality, Research project, University of Berkley. http://osq.cs.berkeley.edu/
3. Halloran, T. J., Scherlis, W. L.: High Quality and Open Source Software Practices. Meeting Challenges and Surviving Success: 2nd Workshop on Open Source Software Engineering, ICSE 24, Orlando, USA (2002) http://opensource.ucc.ie/icse2002/HalloranScherlis.pdf
4. Feller, J., Fitzgerald, B.: A Framework Analysis of the Open Source Software Development Paradigm. In Orlikowski, W., Weill, P., Ang, S., Krcmar, H. (Eds) Proceedings of the 21st Annual International Conference on Information Systems, Brisbane, Australia (2000) http://afis.ucc.ie/bfitzgerald/oss-icis00.pdf
5. Schmidt, D. C., Adam Porter, A.: Leveraging Open-Source Communities to Improve the Quality & Performance of Open-Source Software. 1st Workshop on Open Source Software Engineering, ICSE 23, Toronto, Canada (2001) http://www.cs.wustl.edu/~schmidt/PDF/skoll.pdf
6. Gauthier, C.-A.: Open Source Software Quality. CASCON 2000, Toronto, Canada. (2000) http://seg.iit.nrc.ca/slides/gauthier01s.pdf

7. Raymond, E. S.: The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary. O'Reilly & Associates (2001)
8. Open Source Case for Business. Open Source Initiative (2003) http://www.opensource.org/advocacy/case_for_business.php
9. DiBona, C., Ockman, S., Stone, M.: Open Sources. Voices from the Open Source Revolution. O'Reilly (1999) http://www.oreilly.com/catalog/opensources/
10. Reilly, T. O.: Gated Communities. O'Reilly Online (2000) http://tim.oreilly.com/opensource/gated.html
11. Perens, B.: Gated Communitites. Linux.com (2000) http://linux.omnipotent.net/article.php?article_id=9939
12. OpenPSA. Professional Service Automation Community. http://www.openpsa.net/
13. McMillan, R.: Friendly, Big and Blue. Linux Magazine (2000) http://www.linux-mag.com/2000-10/ibm_01.html
14. Ambler, S. W.: Reuse through internal open source. Software Development Online (2000) http://www.sdmagazine.com/documents/s=737/sdm0012p/0012p.htm
15. Vaughan-Nichols, S. J.: It's Tool Time. eWeek (2001) http://www.eweek.com/print_article/0,3668,a=11619,00.asp
16. Glott, R.: Free/Libre and Open Source Software – Developer Survey. Egovos conference, Washington DC (2002) http://www.infonomics.nl/FLOSS/workshop/presentations/FLOSS@egovos_developer/index.htm
17. McMillan, R.: The Great Dictator. Linus Torvalds: The Benevolent, Brilliant Keeper of the Kernel. Linux Magazine Online (2002) http://www.linux-mag.com/2002-12/linus_01.html
18. Reasoning Inc. How Open-Source and Commercial Software Compare: A Quantitative Analysis of TCP/IP Implementations in Commercial Software and in the Linux Kernel (2003) http://www.reasoning.com
19. Musa J. D.: Software Reliability, Measurement, Prediction, Application: McGraw Hill (1987)
20. Garvin, D. A.: Japanese Quality Management. Columbia Journal of World Business (1984)
21. McConnell, S.: Open-Source Methodology: Ready for Prime Time?, IEEE Software vol. 16, no. 4, pp. 6-8 (1999)
22. Hecker, F.: Setting up Shop: The business of Open Source Software, (2000) http://www.hecker.org/writings/setting-up-shop.html.
23. Reis, C. and Pontin, R.: An Overview of the Software Engineering Process and Tools in the Mozilla Project (2002) http://opensource.mit.edu/papers/reismozilla.pdf.
24. Dinkelacker, J. and Garg, P.K.: Corporate Source: applying open source concepts to a corporate environment, In: *Making Sense of the Bazaar: Proceedings of the 1st Workshop on Open Source Software Engineering*, 2001. http://opensource.ucc.ie/icse2001/dinkelackergarg.pdf