Detlef Hühnlein et al. (Eds.): Open Identity Summit 2015,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2015 55

Quality Management in Open Source Projects – Experiences from the Open eCard Project

Daniel Nemmert[1] Hans-Martin Haase[1] Detlef Hühnlein[1] Tobias Wich[1]

Abstract: Open Source Software (OSS) has immensely increased in popularity over the years and it is well known, that software with public access to the sources is on average less error prone than closed source software, especially if the project is supported by a large community which peer reviews the sources [Kua02]. For new and smaller projects however there is no large community yet and hence achieving and maintaining sufficient product quality is challenging. Against this background the present paper discusses aspects of product quality management for OSS in general and shares the experiences gathered in the Open eCard project, which has developed an ISO/IEC 24727 based eID client.

Keywords: Open Source, Quality Management, electronic identification (eID).

1 Introduction

Open Source — as a software development model — enables free access to the source code of software published under an appropriate license[2]. Such projects may be used as provided or forked and modified to fit individuals needs. Open Source Software (OSS) has been a success story for over a decade [MP12, CAHM04]. Particularly for Open Source projects with large communities, one of the reasons for this success may be seen in the fact that severe bugs are usually found more reliable and sooner than they would be detected otherwise [Kua02].

For new and highly specialized projects however there is no large community yet and hence achieving and maintaining sufficient quality is challenging. For the Open eCard project for example, which has developed an ISO/IEC 24727 based eID client it is rather unlikely to acquire a large community, because smart card development is still a highly specialized field as smart cards may usually not be used by citizens for individual purposes. Therefore in the scope of the Open eCard project, a development process has been implemented that uses ISO standards with regard to (software) quality as a frame of reference to compensate for the low amount of input from the community.

2 Related Work

Open Source development is not a new topic for scientific research. There have been a number of publications examining the product quality of Open Source projects compared

[1] {daniel.nemmert, hans-martin.haase, detlef.huehnlein, tobias.wich}@ecsec.de, ecsec GmbH, Sudetenstraße 16, 96247 Michelau

[2] See `http://opensource.org/licenses` for example.

to closed source projects over the years, with other studies examining which factors help an Open Source project to succeed. The subject of most studies tends to focus on projects with larger communities or projects that are developed for a specialized topic that nevertheless tries to solve a pressing need for many people, leading to large amount of feedback albeit the actual development team may be quite small. As a consequence there seems to be a distinct lack of research concerning projects with very small communities and a low amount of feedback from third parties.

After reviewing other empirical research on the scope of OSS, Crowston et al. [CWHW12] come to the conclusion that although OSS is rapidly increasing in popularity and adoption as a software development method, there is still a need to examine Open Source development processes and their "socio-technical work practices."

Aberdour [Abe07] reviewed studies of OSS with the goal to better understand how to improve the software quality of OSS *and* closed source software projects. He draws several conclusions (e.g. creating a sustainable community, testing processes and successful strategies for project management) which will also be investigated further in this paper.

Midha and Palvia [MP12] examine the intrinsic and extrinsic success factors of an OSS project in the first three years of its existence. In their paper they analyze popular assumptions on what contributes to the success of an OSS project. Their work helps to explain the reasons for the low participation by third parties for highly specialized and complex projects. Their research supports the hypothesis that high complexity and focus on niche markets are factors that result in low involvement from the outside.

Jennifer Kuan [Kua02] predicts that the source code of new Open Source projects will surpass the software quality of their closed source counterparts. In her analysis she finds evidence that this prediction holds up for many Open Source projects.

In 2006, Martin Fowler [Fow15] conducted central rules for the software development practice of Continuous Integration (CI), which is used in software development processes popular in the Open Source community, which are inspired by agile methods. Miller [Mil08] provides an example for CI being employed in a closed source project within Microsoft.

Hoffmann [Hof13] provides a comprehensive treatment of aspects related to software quality. He not only covers quality management processes (maturity models and software development methodologies) and software testing but also related topics, such as software bugs, quality assurance and code analysis.

Schneider [Sch08] provides comments on the ISO 9241-110 standard, which addresses software usability.

One of the central standards regarding product quality management is the ISO 9001 [ISO08] standard. It is applicable to virtually every industry and is therefore one of the most often used standards for certification concerning product quality management. Applying the standard to software development, however, proved to be problematic due to the focus of the standard on the manufacturing industry. While the production of goods is the focal

point for product quality management in ISO 9001 (which is barely existent in the software industry), with the design phase beforehand being relatively small compared to the production phase, software development almost completely consists of design and engineering. Therefore the additional guideline ISO/IEC 90003 [ISO14b] was developed at ISO, providing recommendations for the usage of ISO 9001 regarding software development. With ISO/IEC 90003 being only a guideline and not an imperative standard, every additional information provided in the standard is not a strict requirement for a successful certification. However, an auditor may nevertheless ask why certain points from the standard are not implemented in a product quality management system for a software company.

Another standard regarding quality is the ISO/IEC 25000 [ISO14a] series known as "SQuaRE" (**S**oftware product **Qua**lity **R**equirements and **E**valuation) which replaces the old ISO/IEC 9126 [ISO01] and ISO/IEC 14598 [ISO99] standards. The standard offers a framework within which the product quality of software can be evaluated.

The ISO standard defines several procedures and processes to establish a quality management system. The development scenario in the standard is expected to be a typical closed source environment with development being conducted inside a company, ignoring the unique factors that exist in an Open Source context. This leads to the question how ISO 90003 can be applied to OSS development that is driven by a company.

On top of the ISO standards concerning product quality, there are also maturity models to be investigated, which focus in the measurement of the quality of the implemented processes. The following standards are not yet part of the product quality management system of the Open eCard project, but will be analyzed at a later time. One example would be ISO 15504, which is also known as "SPICE" (**S**oftware **P**rocess **I**mprovement and **C**apability d**E**termination). ISO 15504 on its own does not provide a fully defined maturity model but provides a requirements catalog for a fully featured model instead. There are two ISO standards that fulfill the requirements provided by ISO 15504 of which each has a slightly different scope: ISO/IEC 12207 (software life cycle processes) and ISO/IEC 15288 (system life cycle proceses). Besides those two standards, additional viable models that are compliant to ISO 15504 are the Capability Maturity Model (CMM) and Capability Maturity Model Integration (CMMI) by the Software Engineering Institute (SEI).

3 Quality Management in the Open eCard Project

Since the very beginning of the Open eCard Project, one of the goals was to assure a high level of quality through testing and well-defined development guidelines.

The Open eCard App is being developed for as many different platforms as possible. Besides versions for several desktop operating systems (Linux, Windows and MacOSX), there is also an App for Android. Although the platform independence alleviates most of the problems that would be present with more platform dependent languages, there are still enough factors left that may cause complications. Those potential incompatibilities include for example the different versions of Java Development Kits and Runtime Environments — which differ not only between completely different operating systems but also

for example between different distributions of Linux — or the different level of support or quality of the drivers that are necessary to access smart cards.

Additionally, the Open eCard App not only has to be secure from an information security point of view, it also has to follow usability best practices to make the software ”feel safe” to use for users without a strong technical background in information technology. This implies that it is not only necessary to have good and secure code, but to also make the app as user friendly as possible.

3.1 Overview

The Open eCard project orientation is geared towards a process that is aligned with several ISO standards concerning quality in general and standards concerning software quality in particular. One of the most important standards regarding quality management is without a doubt the ISO 9001 standard [ISO08] — and the guideline ISO/IEC 90003 [ISO14b] which expands on and explains ISO 9001 in terms of software development.

The basis for the quality management system of this ISO standard is a ”Plan-Do-Check-Act” (PDCA) cycle as shown in Fig. 1. The model takes the requirements from the customers as input and starts a cycle of continuous improvement starting with the product realization. After a full cycle the product will either be released or the cycle continues until the customer is satisfied with the product. ISO/IEC 90003 provides guidelines for the application of this standard to software.

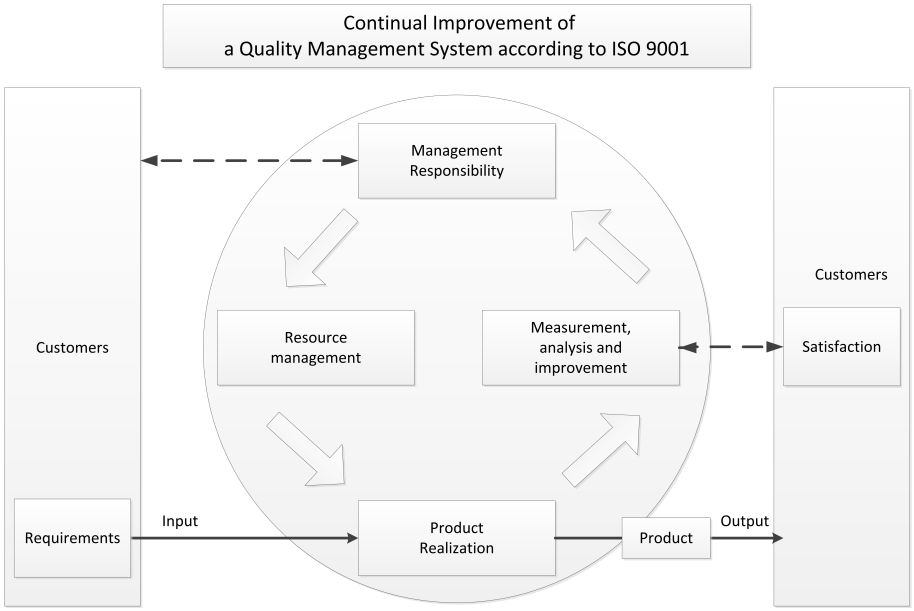

Fig. 1: Process-based Quality Management System according to ISO 9001[ISO08]

The Open eCard Project employs an adapted version of the classic ISO Quality Management System (see Fig. 2) to better reflect the specific requirements within the project. The used system is less strict about the different phases of the PDCA-process with most phases overlapping in some way. This overlap exists as a result of the different phases being partly interconnected with each other. Community interaction, and as a result the recruitment of new developers or testers (contributors) can be a part of the first phase or the first two phases if a new team member is already working on a feature by himself and does so before and after he has been recruited making a smooth transition into the development phase possible. The only isolated phase in the life cycle is the collection of feedback and marketing, which is carried out after a new release and before any new work on the Open eCard App itself begins. After a new release, feedback from users is collected, evaluated and converted into new features, patches or new or refined requirements. In this phase there will usually be announcements for major new versions or related news.

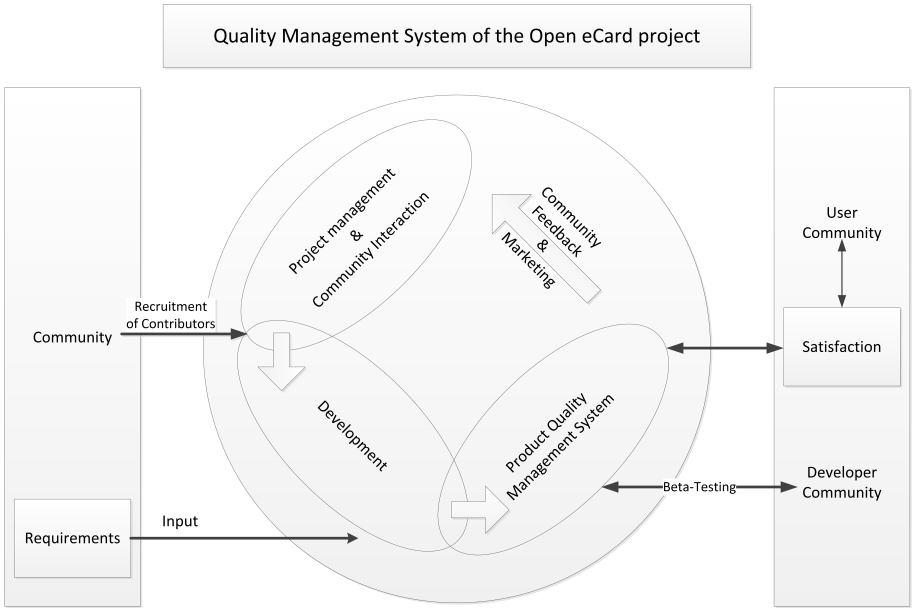

Fig. 2: ISO 9000 inspired QMS as used in the Open eCard project

3.2 Project Management & Community

One of the most important goals for any OSS project is to have a large *and* active community. Aberdour [Abe07] summarizes some of the studies done in terms of OSS quality and proposes that there are certain factors that are common in most successful and high-quality OSS to some degree. The bigger the community, the more new features can be implemented and the more bugs are found — increasing the chances to find and fix potentially devastating vulnerabilities faster than they would be found in a closed source project. This assumption comes with a caveat, though. Projects with rather large communities and the respective quantity of developers require a very good community management to achieve

this goal. Otherwise the sheer size is still useful for finding bugs, but can be a limiting factor that slows down development and deteriorates the general quality of the code base. In addition, a good documentation of everything related to the project is not only essential for closed source projects, but even more so for an OSS project. A good documentation facilitates the participation by other interested developers. Another important success factor is having just the right degree of modularity, which, when done right, facilitates the introduction of new features. While modularity is generally a good thing, "high modularity can lead to greater complexity; therefore, administrators need to keep modularity within an acceptable range." [MP12]

3.2.1 Core Team

The core team consists of the project manager, a maintainer as well as core members acting as developers and testers. Interested parties can gain access to the git-based development repository at GitHub[3] and may easily become tester or developer. Using the example of a new feature the different roles fulfill different tasks in the development process. A ticket for the feature will be created and assigned to a developer. The developer will then implement a solution for the new feature in a given time. Developers are encouraged to provide unit test facilities with their contribution, which will be integrated in the CI system. When the first version of the new feature is implemented, one or more testers (depending on the complexity of the new feature and the available personnel) review the new code and the new functionality with the goal to find any outstanding bugs. After the testing is finished the maintainer will be responsible for the final quality assurance and the integration of a contribution into the corresponding development branch. If any bugs are found in the review phase, a ticket will be created and if possible assigned to the original developer and the process starts from the beginning. The project manager is responsible for making informed decisions about the future direction of the project and assign priorities and new tasks to the rest of the team. Although the number of contributors is quite low, this must be strictly adhered to, because it gives the project a clear structure for each new release and avoids potential chaos in the submission of new features and subsequently in the release of new versions.

3.2.2 Community Add-ons

The Open eCard App also offers the possibility to develop add-ons to the software, enabling everyone to implement new features, without breaking the rest of the core software. Add-on capabilities also make it easier for other companies to develop their own, possibly very specific, additions to the software increasing the potential adoption rate. A company or a group of individuals could for example develop a plugin that enables the Open eCard App to be started before the login to the operating system to use the app for system authentication. In the best case a group of developers will then share their plugin with the original project.

[3] See `https://github.com/ecsec/open-ecard`.

3.2.3 Release Cycle

The best thing that can happen to an OSS project is having an active community. To ensure that, the core team of the project must make sure that release cycles aren't too long. Of course it does not make sense to release weekly new versions without any notable changes. Every OSS project has to try and find the right balance between "exciting" new releases and a release cycle that is short enough to keep its community interested in contributing to the project.

3.3 Development

> "Quality management system planning at the organizational level may include the following: [...] b) defining the work products of software development, such as software requirements documents, architectural design documents, detailed design documents, program code, and software user documentation; [ISO14b]"

3.3.1 Requirements

An initial set of requirements for the Open eCard App has been specified in [KPS+13, WHP+13]. Over time these requirements have been refined and are roughly as follows: The Open eCard App as to support multiple platforms (e.g. Windows 7/8.1 or Mac OS X 10.7 - 10.9 and popular Linux distributions) and a multitude of different cards[4] and especially demonstrate conformity with Technical Guideline TR 03124 [Fed15b, Fed15a] of the Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik, BSI). Conformity to this technical guideline is especially important in order to ensure the full compatibility with the German eID card (Personalausweis) and may be formally assured by a corresponding certification of the BSI.

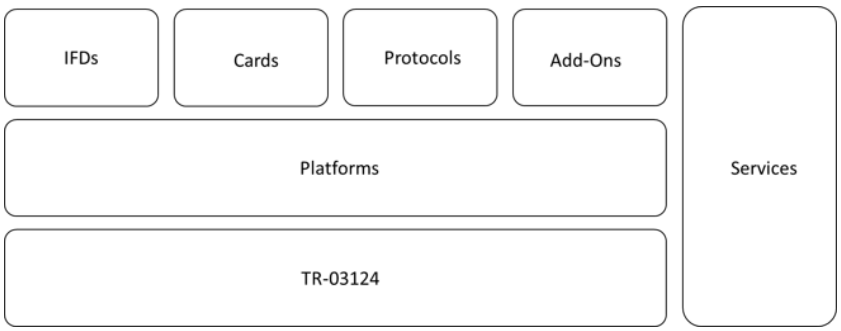

Fig. 3: Overview of requirements of the Open eCard App

At the very beginning of the project there were less planned modules than shown in figure 3. This latest iteration of the specification is a result of a constant evaluation of feedback

[4] A list of the currently supported cards can be found at: `https://www.openecard.org/ecards/`

by the surrounding community and internal evaluations by the core development team. As a result of the ongoing input of ideas and critique the scope of the App is much more complex today. This is an ongoing process that repeats itself for every new version of the App as can be seen in figure 2.

3.3.2 Development Guidelines

Another very important factor for the success of an OSS project is to have well-defined guidelines regarding a variety of development aspects. Without maintaining strict guidelines, having a potentially large community will almost certainly lead to disaster or at least to a huge amount of additional work for the members of the core team, whose time would be better spent on implementing new core features than formatting and rewriting code or rejecting bad code. Defining as strict as possible development guidelines as early as possible leads to a lot less work later on in the project.

In the Open eCard project there are guidelines for the development process, code-style, usage of the repository, and a general guide for developers. Besides those specific guidelines the central rules for general quality assurance governing the area of code commits are:

- Only atomic commits.
- Untested code must not be committed.
- Undocumented code must not be committed.

The developer guide[5] right now consists of two parts: One part provides an overview on how add-on development is realized, giving information for example about the relevant packages and its content. The other is a full documentation of the API of the project in form of a JavaDoc[6]. Again: Having reasonable modular extensibility and good documentation is critical to encourage participation in an OSS project.

The Open eCard project uses the term ”add-on” as a generic description for any type of component that enhances the functionality of the application and makes use of the publicly available API, which is limited to certain features from the full API. This model can be compared to the add-on model popular browsers like Firefox or Chrome are using. As explained in Wich et al. [KPS+13], an add-on can either be an extension or a plug-in, with both fulfilling different purposes. Extensions are directly included into the user interface and can be executed and interacted with by the user. An example for this would be an add-on that provides the functionality to change the pin of a smart card. Plug-ins, on the other hand, are dependent on the context the user utilizes the add-on in. Performing an authentication to a service using a particular smart card, for instance, requires a plug-in which is capable of providing such functionality.

[5] See `https://dev.openecard.org/projects/open-ecard/wiki/Developer_Guide`.
[6] See `https://ci.openecard.org/job/Document/javadoc/`.

Also there is a short explanation on how to begin with add-on development and what to consider before and while developing add-ons for the software.

As mentioned above, it is important for an OSS project to have strict code-style guidelines. If an OSS is adopted by a wide user base, strict code-style guidelines not only help to ”make a good impression” on other users or potential co-developers but also ensure a high readability of the code, which in turn leads to more feedback, which may encourage others to contribute. During the early stages of the Open eCard project, those guidelines have already existed, but weren't enforced in a very strict manner. Which led, in some parts, to code, that is not up to par to the code-style guidelines. As a consequence code-style is one of the most strictly enforced guidelines in the project now. It also facilitates the work of the core team, since newer developers (attracted to the project for various reasons) have access to information on how to format their code fitting to the general style of the project. Furthermore badly formatted code can be instantly rejected based on those guidelines.

The Code-Style guidelines[7] determine how well-formed code has to be formatted for the Open eCard project. In them, general rules are defined governing for example the use of whitespaces, indentations, line endings, file encoding, the language of the code and documentation or other relevant documents. In addition to the general rules, there are more specific rules for document types relevant to the project.

The Open eCard project uses a revision control system, as it is highly recommended for every software development project. The repository is public and hosted on GitHub[8]. Only the maintainer has write permission to this master repository, but every user with git development access, which is provided manually on a case-by-case basis, gets his own repository for development. The development model is similar to the traditional model that is used for the development of the Linux kernel. Additionally, there is the possibility for every interested party to open issues in the central project management and issue tracking tool[9]. The Open eCard Wiki[10] contains more detailed information about the whole process to ease the setup process for less experienced contributors. There are also rules on the correct formatting of the commit messages and other general rules.

3.3.3 Release Cycle

> ”Processes, activities and tasks should be planned and performed using life cycle models suitable to the nature of a software project, considering size, complexity, safety, risk and integrity.” [ISO14b]

The release cycle of the Open eCard project follows several rules. The release versions will be numbered according to Semantic Versioning [SV115]. At the beginning of a release

[7] See `https://dev.openecard.org/projects/open-ecard/wiki/Code-style`.
[8] `https://github.com/ecsec/open-ecard`
[9] access to the Redmine based ”Open eCard Development Center” can be gained by registering at: `http://join.openecard.org`
[10] See `https://dev.openecard.org/projects/open-ecard/wiki/Wiki`.

cycle the project manager assigns tasks to individual developers who then develop the new feature for the upcoming release and he also announces a targeted release date.

To ensure a stable release version, the code is frozen at least two weeks before the end of the cycle by the maintainer. From this point onward, only bug fixes are allowed to be committed to the branch of this release cycle. New features are committed to the branch of the next release. During this two or more week period the release manager and the testers need to test the code thoroughly. If bugs are found, the responsible developer is informed and the testers and developers agree on a person who will be responsible to correct the error. Fixing a bug should not take longer than two working days.

When the tests are completed and all detected errors are corrected, the maintainer is then responsible to build a release version and distribute it to the users. If a bug is found in the current release version, the maintainer and possibly the project manager decide whether its impact is big enough to warrant a bug-fix release or if it is sufficient to correct the error in the following scheduled release version. In case of a bug-fix release the testing stage is repeated: the maintainer contacts the respective developer to fix the bug and gets the new version reviewed by the testers. As soon as the error is fixed the maintainer builds the bug-fix release and again distributes it to the users.

3.4 Quality Assurance

ISO 9000 defines quality assurance as a ”part of quality management (3.2.8) focused on providing confidence that quality requirements will be fulfilled.” [ISO05] In this sense quality assurance is the process which has the objective to guarantee that a product works as intended. The most common and obvious way to achieve this goal is to define and conduct tests. In most software projects it is possible to automate many, if not all, tests required to enable the desired quality level, with the remaining test cases being carried out manually.

Because quality should be the goal of every OSS project that aspires to be successful, quality assurance, and as a consequence testing procedures, must be an important focal point. With limited resources being the most problematic aspect of many OSS projects (apart from projects led by bigger companies) establishing formal testing procedures — like automated tests and continuous integration — is not feasible for every project. Another possibility to improve the quality of the code base is to let the code be reviewed by external parties that have absolutely no connections to the project and do not turn ”a blind eye” to some errors.

The Open eCard project employs a mixture of manual and automated tests in the form of Continuous Integration (CI) and acceptance testing where automation is not feasible. In the scope of this project, a high level of product quality is achieved, when all defined features work as specified and can withstand both phases of the testing procedure.

It is very important to carefully plan and implement a suitable testing procedure right at the beginning of the project. Jenkins, for example, offers the possibility to check if

submitted code follows the defined code-style guidelines. If the code-style check surpasses a defined threshold in terms of warnings, the maintainer is notified via e-mail, whose work is immensely facilitated in consequence of this.

3.4.1 Continuous Integration

> ”Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.” [Fow15]

Fowler postulates eleven best practices for CI including points such as to maintain a single repository or to automate the build among others. By implementing those guidelines a project can create an efficient development process, that leads to much less integration problems, which is especially important for OSS with multiple contributors, but is also already useful if there is more than one developer working on the software.

Of course a project does not have to fulfill all those best practices at once. Every single step that is used in a software project, mitigates the risk of ending up in an ”integration hell” [Fow15]. The implementation of Continuous Integration is easier than might be expected, because there are very powerful Open Source CI systems available on the market, which may be customized to any needs a specific project might have.

Miller [Mil08], for example, comes to the conclusion that ”teams moving to a CI driven process can expect to achieve at least a 40% reduction in check-in overhead when compared to a check-in process that maintains the same level of code base and product quality.” This reduction alone spares a lot of time not only for bigger projects, but also helps relatively small teams to become more efficient.

The CI process is on the one hand supported by the concept of Mocking and on the other hand by the Open Source CI tool Jenkins[11] Mocking facilitates the testing process by providing mock data required for the testing procedure without the need to have an infrastructure in place that emulates the production environment. As a result, using mock data not only reduces the development costs, it also speeds up the development process because there is no maintenance or setup time involved — or at least less than for e.g. setting up an appropriate database. It is also possible to automate the creation of mock data, eliminating the time needed for creating classes that provide the data necessary to run the tests. Jenkins is one of the most popular CI tools, owing its popularity mainly to the widespread adoption by the Open Source community [Wie11]. The tool enables a development team to implement an extensible CI system facilitating the development process immensely.

[11] See `https://jenkins-ci.org/`.

Most of the testing is done by automated build-testing through a CI-system. In the development process a developer pushes his changes to a central version control system, which in turn informs the CI-system about the presence of changes. The CI-system then conducts the defined tests. If any of the tests fail, the developer is informed so that he can review and edit his changes. If all tests pass the CI-system informs both the developer and the release manager of which the latter then checks the changes and decides about whether to accept or reject the changes.

Both aspects are used in the Open eCard project, albeit limited by external systems that are part of the scope of the project (see section 3.4.2).

3.4.2 Acceptance Testing

The main focus of the acceptance tests is achieving a certification according to the technical guideline TR-03124 Part 2 [Fed15a]. The conformance with the guideline is being verified by using the official test suite issued by the BSI. This is, of course, an extremely specialized certification, which focuses on the conformity with the requirements defined in TR-03124 Part 1 [Fed15b]. Alternative certifications, which are not yet part of the requirements, would include Common Criteria (ISO/IEC 15408 [ISO09]) and NIST FIPS 140-2 [NIS01].

As mentioned earlier, external systems make an almost complete automation of testing very challenging, if not impossible. External systems in this case are primarily the various different smart cards and their integration into the software. Although standards like the ISO/IEC 7816 series related to electronic identification cards with contacts provide guidelines for various characteristics of smart cards the standard itself leaves room for proprietary manufacturer specific aspects which leads to vastly different implementations of the standard. The multi-part standard ISO/IEC 24727 aims to provide normative guidelines for the interoperability between different identity tokens and applications. The standard is an important step towards a more harmonized smart card environment in the future, with governments (e.g. in Germany) already adopting ISO/IEC 24727 for the implementation of their electronic identity card and the related infrastructure. This standard will ease supporting different smart cards easier in the future.

Besides testing different smart cards, the stability of the GUI is tested along with the general behavior of the App. As a last, but nevertheless important, step the cryptographic functionality of the App is being reviewed by experts to limit the possibility of faulty — and as a consequence potentially vulnerable — implementations.

3.5 Community Feedback & Marketing

A large and active community is arguably one of the most import factors which contributes to the success of an OSS project [Abe07]. The community for a successful OSS project is, according to Aberdour, like an onion with the large user base (which uses the software and

notices bugs) as the first layer, a smaller group of bug reporters as the second layer, with the next layer consisting of contributing developers and at the center a core team which leads the project.

The structure of a typical Open Source community is in most cases similar to the onion model described by Aberdour [Abe07]. The community in this model consists of a relatively small core team that is responsible for the main work done on the project. In the case of the Open eCard project this idealized model does not really apply, because not only the project manager and the maintainer would be part of the core team, but also the developers and testers, which are also active developers for the project. Testers are also responsible for bug reports, but so is every member of the team. Having no large and active community surrounding the Open eCard project makes it difficult to apply the most popular community models. As a consequence most development work is done within the core team itself.

An interested developer commonly engages with an OSS project, because the software is missing a feature that is important to him. This engagement can take on multiple forms: from the creation of a ticket or post on the forums, to coding the feature by himself or as part of the OSS team with the goal of implementing in the software, to forking the project and building his own version [Kua02]. This leads, over time, to a more and more ”feature complete” and robust software.

The community of the Open eCard project consists almost entirely of universities and other potentially interested companies. The low participation by private developers may be attributed to the unfortunate introduction of the German eID card and a lack of opportunities to use it for authentication purposes online, leading to a very small potential user base that on top of it all is mostly limited to the universities conducting research in this field and the companies that have financial interest in developing an own solution for card based authentication at the moment. There simply are no features a private developer could need badly enough for him to contribute to the project.

For users that are not interested in being part of the development team there is the possibility to provide feedback via a central issue tracker and via e-mail[12].

Nevertheless, the necessity for making this project open to the public is very high. Making this project Open Source and the code reviewable by everyone imposes a high level of trust in the security of the application, which is essential to the future adoption rate of the application and related eID systems. An application like the Open eCard App that deals with potentially highly sensitive information needs a high level of trust from the general public in order to be widely used for authentication purposes.

4 Conclusions

Implementing a basic product quality management for an OSS project requires less work than one might expect and hence is recommended for any project. The Open eCard project

[12] feedback@openecard.org

has demonstrated that it is even feasible for rather small projects to implement a quality management system as defined by ISO 9001 and ISO/IEC 90003. All factors described above — apart from the costly certification according to [Fed15b] — are possible to implement in almost every OSS project without extensive public funding or funding via relatively new channels like Kickstarter or Indiegogo. The resources required to realize the ISO 9001 inspired quality management system and the automated testing environment are within reasonable limits and should be affordable for almost every OSS project.

After the formal finalization of the certification according to BSI TR-03124 [Fed15b, Fed15a] a future goal may be the systematic examination and improvement of the Open eCard App with respect to usability, which may especially consider ISO 9241-100 [ISO10] and related guidelines.

References

[Abe07] Mark Aberdour. Achieving quality in open-source software. *IEEE Software*, 24(1):58–64, 2007.

[CAHM04] Kevin Crowston, Hala Annabi, James Howison, and Chengetai Masango. Effective work practices for software engineering: free/libre open source software development. In *Proceedings of the 2004 ACM workshop on Interdisciplinary software engineering research*, pages 18–26. ACM, 2004.

[CWHW12] Kevin Crowston, Kangning Wei, James Howison, and Andrea Wiggins. Free/Libre open-source software development: What we know and what we do not know. *ACM Computing Surveys (CSUR)*, 44(2):7:1–7:35, 2012.

[Fed15a] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eID-Client – Conformance Test Specification. Technical Directive (BSI-TR-03124), Version 1.2, Part 2, 2015.

[Fed15b] Federal Office for Information Security (Bundesamt für Sicherheit in der Informationstechnik). eID-Client – Specifications. Technical Directive (BSI-TR-03124), Version 1.2, Part 1, 2015.

[Fow15] Continuous Integration, 2015. `http://martinfowler.com/articles/continuousIntegration.html`, Stand: 16.04.2015.

[Hof13] Dirk W. Hoffmann. *Software-Qualität*. Springer-Verlag, Berlin Heidelberg, 2nd edition, 2013.

[ISO99] ISO. 14598-1:1999 Information technology – Software product evaluation – Part 1: General overview. ISO 14598-1:1999, International Organization for Standardization, 1999.

[ISO01] ISO. 9126-1:2001 Software Engineering – Product Qualiy – Part 1: Quality model. ISO 9126-1:2001, International Organization for Standardization, 2001.

[ISO05] ISO. 9000:2005 Quality management systems Fundamentals and vocabulary. ISO 9000:2005, International Organization for Standardization, 2005.

[ISO08] ISO. 9001:2008 Quality management systems – Requirements. ISO 9001:2008, International Organization for Standardization, 2008.

[ISO09] ISO/IEC. 15408-1:2009 Information technology – Security techniques – Evaluation criteria for IT security – Part 1: Introduction and general model. ISO 15408-1:2009, International Organization for Standardization, 2009.

[ISO10] ISO/TR. 9241-100:2010 Ergonomics of human-system interaction – Part 100: Introduction to standards related to software ergonomics. ISO/TR 9241-100:2010, International Organization for Standardization, 2010.

[ISO14a] ISO. 25000:2014 Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – Guide to SQuare. ISO 25000:2014, International Organization for Standardization, 2014.

[ISO14b] ISO. 90003:2014 Software engineering – Guidelines for the application of ISO 9001:2008 to computer software. ISO 90003:2014, International Organization for Standardization, 2014.

[KPS+13] Raik Kuhlisch, Dirk Petrautzki, Johannes Schmölz, Ben Kraufmann, Florian Thiemer, Tobias Wich, Detlef Hühnlein, and Thomas Wieland. An Open eCard Plug-in for accessing the German national Personal Health Record. In Detlef Hühnlein and Heiko Roßnagel, editors, *Proceedings of Open Identity Summit 2013*, volume 223 of *Lecture Notes in Informatics*, pages 82 – 94. Gesellschaft für Informatik eV (GI), 2013.

[Kua02] Jennifer Kuan. Open source software as lead users make or buy decision: a study of open and closed source quality. *Stanford Institute for Economic Policy Research, Stanford University*, 2002.

[Mil08] Ade Miller. A hundred days of continuous integration. In *Agile, 2008. AGILE'08. Conference*, pages 289–293. IEEE, 2008.

[MP12] Vishal Midha and Prashant Palvia. Factors affecting the success of Open Source Software. *Journal of Systems and Software*, 85(4):895–905, 2012.

[NIS01] NIST. 140-2: Security Requirements for Cryptographic Modules. ISO 14598-1:1999, National Institute of Standards and Technology, 2001.

[Sch08] Wolfgang Schneider. *Ergonomische Gestaltung von Benutzungsschnittstellen. Kommentar zur Grundsatznorm DIN EN ISO 9241-100:2010*, volume 2. Beuth Verlag GmbH, 2008.

[SV115] Semantic Versioning 2.0.0, 2015. `http://semver.org/`, Stand: 16.04.2015.

[WHP+13] Tobias Wich, Moritz Horsch, Dirk Petrautzki, Johannes Schmölz, Thomas Wieland, and Detlef Hühnlein. An extensible platform for eID, signatures and more. In Detlef Hühnlein and Heiko Roßnagel, editors, *Proceedings of Open Identity Summit 2013*, volume 223 of *Lecture Notes in Informatics*, pages 55 – 68. Gesellschaft für Informatik eV (GI), 2013.

[Wie11] Simon Wiest. *Continuous Integration mit Hudson*. dpunkt.verlag, 2011.