Electronic Communications of the EASST
Volume 33 (2010)

Proceedings of the Fourth International Workshop on Foundations and Techniques for Open Source Software Certification (OpenCert 2010)

Component Certification as a Prerequisite for Widespread OSS Reuse [1]

George Kakarontzas, Panagiotis Katsaros, Ioannis Stamelos

20 pages

Guest Editors: Luis S. Barbosa, Antonio Cerone, Siraj A. Shaikh
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

[1] This work is partially funded by the European Commission in the context of the OPEN-SME "Open-Source Software Reuse Services for SMEs" project, under the grant agreement no. FP7-SME-2008-2 / 243768

Component Certification as a Prerequisite for Widespread OSS Reuse [†]

George Kakarontzas[1], Panagiotis Katsaros[2], Ioannis Stamelos[3]

[1] gkakaron@teilar.gr
[2] katsaros@csd.auth.gr
[3] stamelos@csd.auth.gr
Aristotle University of Thessaloniki, Greece

Abstract: Open source software is the product of a community process that in a single project may employ different development techniques and volunteers with diverse skills, interests and hardware. Reuse of OSS software in systems that will have to guarantee certain product properties is still complicated. The main reason is the many different levels of trust that can be placed on the various OSS sources and the lack of information for the impact that a reused OSS component can have on the system properties. A prerequisite for promoting widespread reuse of OSS software is certification at the component level in an affordable cost. This work addresses the main technical issues behind OSS component certification by formal and semiformal techniques, as well as the incentives that raised the need for the OPEN-SME European funded project. OPEN-SME introduces an OSS software reuse service for SMEs, in order to address the problem that OSS evolves by volunteers that follow different development processes. We discuss the requirements relating to OSS software reuse based on the findings of a survey. Then we present the OPEN-SME tool-set and approach for OSS reuse and finally we show how the provision of verifiable certificates can provide assurance that an OSS component conforms to one or more anticipated requirements, necessary for reusing it in a system.

Keywords: component certification, open source software reuse, component-based development

1 Introduction

Trusted software components are defined as "reusable software elements that possess specified and guaranteed property qualities" [Mey03]. This definition by Bertrand Meyer emphasizes two aspects that we need to consider. The first one is that elements should be 'reusable'. The second is that someone should guarantee their properties in relation to quality (e.g. security). Reusability may be considered as an umbrella property that embeds many other properties including quality related ones. Quality on the other hand is a multi-faceted concept with many different and often incompatible views [Gar84, KP96]. In software, numerous quality models have been established in an attempt to capture essential quality aspects and product characteristics that contribute to

[†] This work is partially funded by the European Commission in the context of the OPEN-SME "Open-Source Software Reuse Services for SMEs" project, under the grant agreement no. FP7-SME-2008-2 / 243768

these aspects. For example the ISO-9126 quality model [ISO01] defines software product quality as a combination of six characteristics, namely functionality, reliability, usability, efficiency, maintainability and portability, which are further sub-divided in sub-characteristics (e.g. the understandability and learnability sub-characteristics of usability). However none of the quality models had established a unanimous consensus and criticism exists even for the ISO quality model, which enjoys the status of an international standard. For example a survey for the ISO quality model revealed ambiguities in the structure of the quality model although it also provided evidence for its (partial) validity [HSC04]. Furthermore software quality can be generally viewed from the perspective of process-based approaches to quality, such as CMMI and ISO-9001, which assume that by improving the process of software development eventually better quality products will follow, and product-based approaches to quality which measure or verify software characteristics to objectively conclude quality related issues. These two general approaches also generate criticism and none is unanimously accepted. For Open Source Software (OSS) quality however, and although process-based approaches are valuable, we cannot hope that they can affect the processes followed by open source projects, since the participation in these projects is mostly volunteer-based. It is therefore important to focus on product characteristics that provide the opportunity to objectively conclude an OSS product's quality. Finally, studies have established that the quality of OSS is comparable to closed source software; however this may be happening for different reasons. For example [Abe07] observes that intense bug reporting in OSS projects in tandem with rapid release cycles results in decreased defect density. Modularity, documentation, improved tools and processes are also very important since they increase participation effectively contributing to the so-called many-eyeballs effect: *"given enough eyeballs, all bugs are shallow"* [Ray01].

The OPEN-SME project [OPE10] introduces a set of methodologies, associated tools and business models centered on SME Associations, which will enable software development SMEs to effectively introduce Open Source Software Reuse practices in their production processes. In this context, software reuse is regarded as the sharing of software modules across different development teams, organizations, and diverse application domains. The potential benefits from the adoption of Software Reuse practices by software SMEs could provide substantial competitive advantages against large players by improving productivity, increasing competitiveness, and facilitating entrance to new markets. A prerequisite for the effective reuse of software modules however is the trustworthiness of these modules. To establish trustworthiness, SME Associations which are representatives of software development SMEs provide a number of services centered on the reuse of OSS software effectively acting as certification authorities for their SME members. The OPEN-SME project emphasizes trustworthiness of software components through product-based approaches to quality because, as we already mentioned, process-based approaches are not suitable for OSS software reuse.

In the rest of this paper in Section 2 we present a survey which we conducted in order to elicit requirements for the trustworthiness of OSS reusable components. Based on the results of this survey we then present the OPEN-SME process in Section 3. In Section 4 we discuss the details of the OPEN-SME tools and processes related to OSS component certification. Next in Section 5 we will review some of the most prominent approaches to OSS software certification and quality. Finally in Section 6 we conclude.

2 A survey of issues for the trustworthiness of OSS software

In order to form the requirements of the OPEN-SME project a questionnaire was sent to a number of Small and Medium Enterprises. The respondents were 41 experienced developers, analysts and designers, of the SMEs participating in the OPEN-SME project. The questionnaire was quite extensive since the OPEN-SME project aims at developing and support a reuse process and tools, however a number of questions were specifically addressing quality issues that are directly related to the trustworthiness aspect of OSS reuse. In this section we present some of the most important findings of this survey which helped in shaping the OPEN-SME processes and tools.

In the question 'What do you consider the most important artifact to reuse' the respondents were given the options to rate in a scale from 1 to 5 (1 being unimportant and 5 being extremely important) the reuse importance of requirements, documentation, design, code and test suites. The result is shown in Figure 1. As can be seen, the most important reuse artifact is source code whereas the less reusable artifact are requirements. The importance of code as a reuse artifact signifies that tools and processes should concentrate in establishing trust at this level, since developers are more likely to be interested in directly reusing source code modules. This emphasis on source code is also evident in a number of commercial tools that aim in architecture reconstruction from source code and re-modularization such as Structure 101 [STR10] and Lattix [Lat10]. The main reason for this emphasis is that design documents and other artifacts are often outdated and inaccurate. The source code is considered therefore the definitive trustworthy artifact to examine, in order to obtain knowledge for a system for the purposes of maintenance, evolution and reuse.

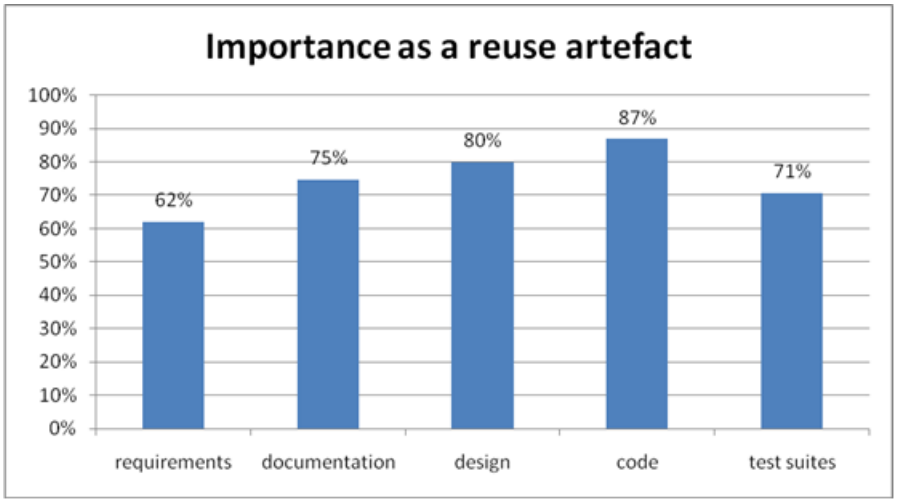

Figure 1: Relative importance of several artifacts as reuse artifacts

Another important finding of the survey is that the vast majority of the respondents said that their organization supports the reuse of OSS software. On the question "Does your organization encourage reuse of Open Source Components?", the vast majority of the respondents (80%) said that their organization supports OSS reuse, but there was also a small percentage (15%) that said

that their organization discourages the reuse of OSS. A small percentage (5%) was not sure if their organization supports the reuse of OSS or not (Figure 2).

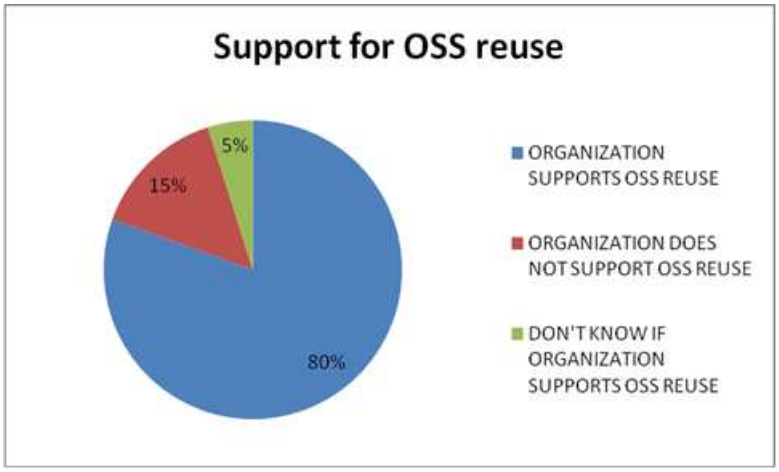

Figure 2: Support for OSS reuse

The most important factors preventing OSS reuse according to the respondents were the lack of documentation (80%) and the uncertainty on the quality of OSS components (76%). Barriers for OSS reuse are also the difficulty in searching and retrieving OSS components (66%) and licensing (59%). The reluctance to use components that were not developed in-house (i.e. the so-called 'Not invented here syndrome') is only preventing OSS reuse by 46% according to the respondents (Figure 3). It is noticeable that two of the factors that are considered very important for trust, namely the knowledge embodied in documentation and the quality uncertainty are also the most important factors preventing the reuse of OSS components.

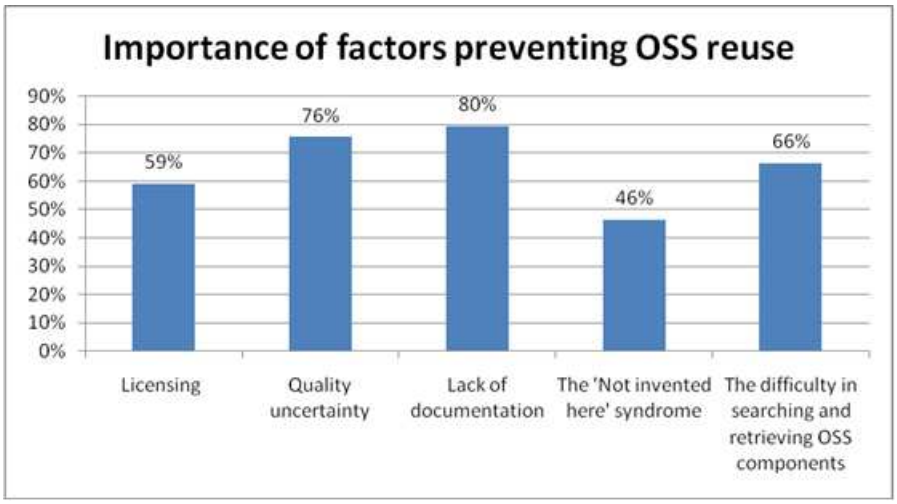

Figure 3: Relative importance of factors preventing OSS reuse

A question that demonstrated a significant gap for the reuse of OSS software regarded the source of reusable OSS components. The respondents were asked to determine the source of reusable components: 'The source for reusable code you usually use is (select as many as appropriate, use a scale between 1 to 5 to indicate the most important sources)" and the options were In-house legacy code repositories, Publicly available open source code repositories (e.g. SourceForge, Google Code, etc.), Specialized Open Source Software search engines (e.g. Koders, Krugle, etc.) and Classical search engines. The responses (Figure 4) demonstrate that developers do not view the specialized OSS search engines as important as other sources of reusable software. Another question was 'Have you used specialized Open Source code search engines (e.g. Google code search, Koders etc.)?' in which the respondents that said that they have used specialized OSS search engines were only 12%.

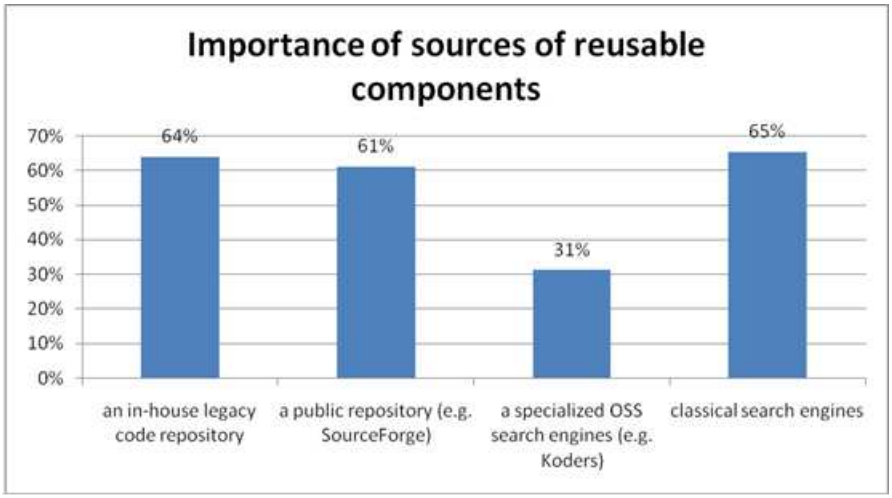

Figure 4: Relative importance of sources of reusable components

Since specialized OSS search engines are not widely used it is interesting to determine what are the most important services that reusers would expect from a repository. On the question of the importance of services that should be provided by a reuse repository (Figure 5), developers view as more important the services related on the description of the reusable components (e.g. version management, and dependency management) than reputation-based services (e.g. repository usage and numbers of downloads of a component). This signifies again the importance of tangible evidence on characteristics and properties of the source code.

Collectively this survey demonstrates that in order to improve the reuse of OSS software third-party reuse service providers are needed. In the context of the OPEN-SME project the reuse service providers are the SME associations. In a different context however these services can be provided by independent corporations. More specifically:

- Source code based analyses that provide undisputable evidence for the quality of OSS components are very important, since they can provide quality metrics and guarantees that can be used as descriptors of OSS components in repositories to improve knowledge and found trust. Quality metrics however are not a replacement for documentation. The lack

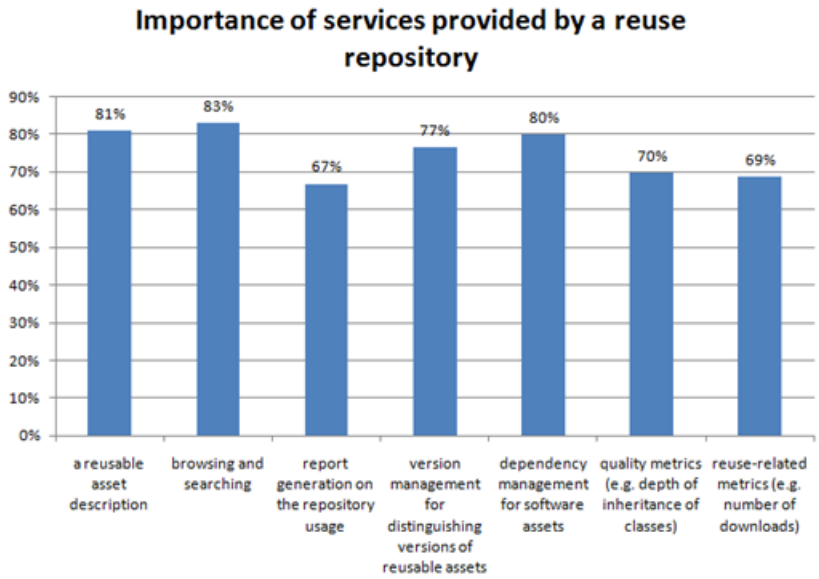

Figure 5: Relative importance of services provided by reuse repositories

of documentation can be counterbalanced by both automatic and semi-automatic means. Examples of automatically produced documentation are the verifiable security properties or the diagrams that can be produced by analyzing the source code. Textual descriptions of software however, are not produced automatically.

- As the survey demonstrated, although source code is entirely different than text, reusers often resort to classical search engines for their reuse needs. This signifies a lack of concrete advantages of specialized OSS search engines in relation to classical search engines. We believe that source code analysis can be used to improve this situation. In fact there are several European research projects (some of which we will review later in Section 5) such as FLOSSMETRICS and SQO-OSS, that already successfully followed this approach. These research efforts concentrated on the collection of metrics from the source code mostly automatically, attempting to analyze thousands of projects. However current state-of-the-art prevents certain useful information to be extracted entirely automatically. For example feature location identifies the components that participate in a use case, a very important piece of information for the reuser, and cannot be entirely automated. It can be however supported by analysis tools in such a way that the engineer is not required to have a prior knowledge of the system under analysis [RHR09]. The OPEN-SME approach complements the entirely automated approaches with the semi-automatic collection of analyses results for a smaller number of projects, targeting specific domains and specific projects.

- The importance of documentation and quality metrics for the establishment of trust in OSS reuse overshadows other factors such as the reluctance to use software not developed in-house. Therefore trustworthiness is not an issue related so much with the origin of the

reused software as it is related to the provision of evidence for the software itself. This fact in tandem with the widespread support for OSS reuse evidenced by the responses to the survey, suggests that there is large potential impact for reuse support services in relation to OSS. In fact the 15% of the respondents who said that their organization does not support OSS reuse, were not concerned with the quality of OSS software, but rather restricted in reusing OSS software due to legislation relating to public sector projects in certain countries, that required commercial support for the provided software.

3 The OPEN-SME process and tools

Trustworthiness is a multi-faceted concept and for reusers to trust software developed from OSS communities an independent service is required in order to provide the missing link between reusers and developers. Software component certification is preferably based on objective measurable product qualities and is carried independently from the original developers of a software system. As the survey from the previous section demonstrated, the automatic collection of metrics from thousands of projects is probably not specific enough for the purpose of reuse which requires more information in a more targeted domain of applications. Besides this, many existing projects already provide results in the area of automated collection of quality metrics (see Section 5 for an overview). OPEN-SME therefore is based on the assumption that a service provider positioned between the OSS projects and the reusers of OSS components provides services of both automated and semi-automated source code analysis, certification based on these analyses and packaging of the results in a familiar format to the reusers. The overall architecture of the OPEN-SME approach is depicted in Figure 6. As can be seen in Figure 6, the role of the reuse facilitator service is carried out by SME Associations. The reusers (i.e. Software Development SMEs) do not seek reusable components directly from existing OSS project repositories and OSS search engines, but rather use the service provided by the SME Associations.

As can be seen in Figure 6, existing OSS search engines and repositories lie outside the scope of the OPEN-SME project. They provide the initial input to the system, which analyzes, generates metadata and packages existing OSS artifacts for reuse. The *domain engineering process* is performed by the *reuse engineer*, an expert operating the OPEN-SME toolset. Systematic software reuse is divided in activities or processes related to building reusable assets and activities and processes related to reusing these assets in the context of a software application development. The latter processes are referred to as application engineering processes. The processes which concentrate on producing reusable assets are collectively referred to as domain engineering processes or methodologies. The authors in [MMYA02] define domain engineering as “the set of activities involved in developing reusable assets across an entire application domain, or family of applications”. In domain engineering a number of applications are identified and their similarities and variabilities are analyzed to produce a domain model. Then the model is designed and implemented. Concrete artifacts of the implemented model are then reused in a number of applications.

The domain engineering process in the context of the OPEN-SME project is different than a typical domain engineering process. The main differences are the following:

1. Domain analysis and design is carried out using as exemplar applications Open Source

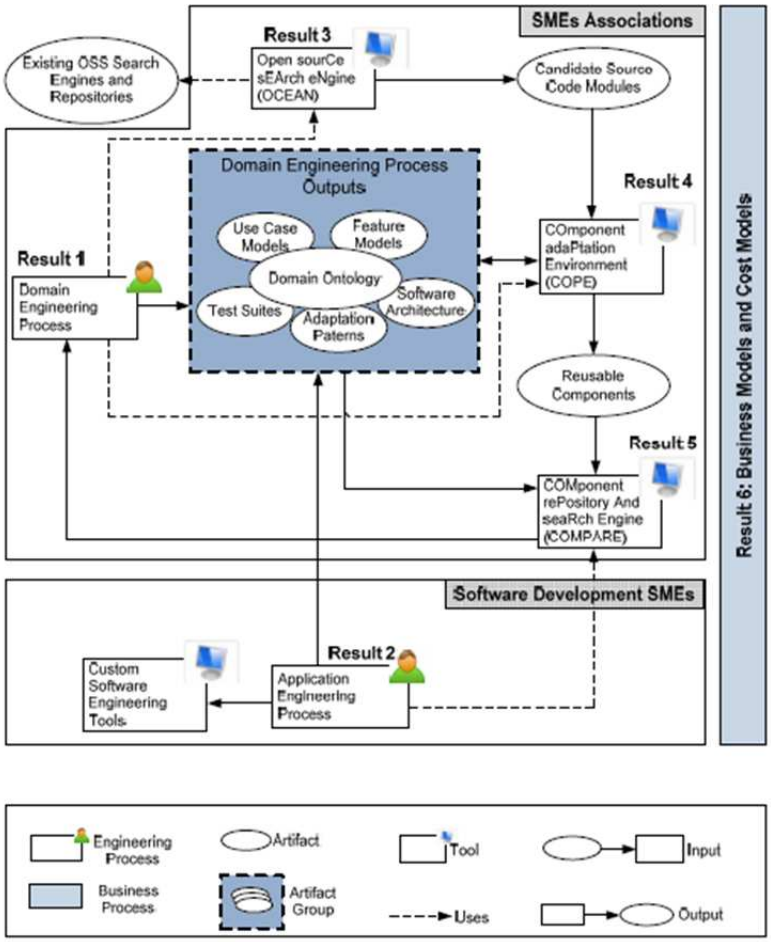

Figure 6: OPEN-SME generic approach and project results

Software (OSS) applications that are available through the OSS search engine (OCEAN). During analysis and design these exemplars will provide inspiration for the concepts of the "unifying" reference architecture. The architecture design (and later the implementation) do not strive to create a common reference architecture (e.g. as in product lines) that will be used as a platform (i.e. framework) for the SMEs applications. This is both undesirable and infeasible. It is undesirable because SMEs wish to differentiate each other, and a common framework would undermine their competitiveness since it would prevent (to a certain extent) innovations. It is infeasible because in practice the SMEs will already have developed their own frameworks and incorporating the technical differences of all these different frameworks to a "unifying" framework is a very difficult if not impossible undertaking. Instead the role of the reference architecture is to provide a detailed description for the common domain concepts and their organization to a typical architecture. This typical architecture plays an important twofold role: (a) It provides a knowledge artifact

enabling the reusers to learn the different concepts and their relationships, and relate these concepts to their own products, and (b) It also provides a classification framework for reusable components.

2. Domain implementation does not implement the domain concepts from scratch. The main input for the implementation of the domain concepts of the reference architecture are the components that are discovered with the analysis of the OSS projects using the Component Adaptation Environment toolset (COPE). These components may in fact differ from the concepts as they appear in the typical architecture. But this does not signify a problem since in the OPEN-SME project the typical architecture does not play the role of the framework that is reused as-is, but rather the role of a classifying framework as we explained earlier. From a reuser perspective what is important is that reusable components are sufficiently related to their classifying concepts from the typical architecture.

Application engineering on the other hand is performed by the SME developers who use their in-house development processes as usual blended with reuse practices similarly to the approach proposed by [CCL06]. The important difference with the OPEN-SME approach, as opposed to reusing OSS software directly from their project repositories, is that the application developers use a domain-specific component repository (i.e. COMPARE) which contains the OSS artifacts after analysis by the SME Associations' reuse engineers. The metadata provided by this analysis step help in increasing trust and consequently improving the *reuse readiness* [NAS10] of existing OSS artifacts.

The tools that the reuse engineers use to perform their analysis, certification and packaging are the following three:

1. *OCEAN*: OCEAN (Open Source Search Engine) will provide unified access to information already available in several OSS search engines. Several OSS search engines exist (e.g. FOSSOLOGY, Google Code Search, Koders, FLOSSMOLE, SQOOSS, etc.) and make it possible to find open source software that satisfies certain conditions, such as software that is written in a specific programming language, having a specific license, contains certain keywords, satisfies certain quality metrics and so on. OCEAN is a *meta-search engine* that is intended to collect information from diverse search engines and provide this information under a unifying framework.

2. *COPE*: COPE (Component Adaptation Environment) is designed to support the following:

   (a) It will provide an environment for the enactment of the domain engineering process of OPEN-SME including feature modeling capabilities and the definition of architectural elements.

   (b) It will provide an environment for static and dynamic analysis of existing OSS projects. The analysis goals include the following: (a) Automatic generation of metadata in addition to those provided by the OCEAN search engine, (b) Semi-automatic identification of the features of software so that it is easier to relate these features to the feature model of the domain, and (c) Provision of input for possible adaptations that can be used in order to adapt the existing identified components to the reference architecture.

(c) It will integrate existing Model Driven Development (MDD) tools that will be used for the adaptations mentioned in (2c) as well as the definition of generic and reusable adaptation patterns.

(d) It will allow reasoning through an appropriate knowledge base. The knowledge base will provide facts for the discovered features of OSS code and the reuse engineer will be able to query the knowledge base. This knowledge base will be also used to enhance the COMPARE repository and search engine with semantic search capabilities.

(e) It will integrate existing safety analysis tools for the analysis of safety properties of the components which is especially significant for safety-critical systems.

(f) It will provide an environment for testing and collection of test results.

(g) Finally it will integrate the above mentioned features through a comprehensive user interface allowing the reuse engineer to perform the above mentioned activities as efficiently and intuitively as possible.

3. *COMPARE*: COMPARE (Component Repository and Search Engine) will support the following activities:

(a) It will provide an environment to assist reusers to search and discover the assets (software artifacts, technical documents, test suites, metamodels) produced by the Domain Engineering Process.

(b) Provide advanced search capabilities for components to the reusers using diverse criteria such as execution framework, programming language, licence, desired functionality, etc.

(c) Store and use all the available information about the role of a component in the domain architecture in order to achieve higher precision and recall.

(d) Use the ‘semantic’ distance of a component from certain specified criteria (e.g. it’s distance from the definition of a prototypical entity from the domain ontology, or the number of tests that the component passes from a given test suite).

(e) Provide the asset retrieval services in such a way that the artifacts are retrieved in a holistic manner.

(f) Support the communication of information, between the asset consumers (re-users) and the asset producers (in our case the reuse engineers), such as requests/orders, bug reports, advertisements, etc.

(g) Provide adequate data models for structuring this kind of communication in order to be rapidly processed.

(h) Allow re-users to obtain information about the verification and certification attributes of a component and certifiers to provide such information.

In the following section we will discuss the tools and processes in relation to component certification.

4 OSS components' certification

Certification as a means for establishing trust aims to provide objective evidence for a product quality in the form of an authoritative statement that suggests contractual obligations and legal implications. Lessons learned from recent attempts [SK04, HALM08] showed that software certification comes at a high cost that the current OSS community is doubtful, if they will ever spent. OPEN-SME introduces an independent reuse service that opens a perspective for economies of scale in OSS reuse through the packaging of OSS software in ready-to-use components. The same service has the potential of an independent party for certifying the packaged OSS code at an affordable cost for the SME members. Certification of OSS components should take place in the context of the system where the components are to be reused. In the OPEN-SME process, a partial description of the system context is provided by the typical domain-specific architecture that prescribes certain quality guarantees for the OSS components. OPEN-SME certification concerns the aforementioned contextual qualities, as well as the implementation specific component qualities that can be ascertained independently of their use context (e.g. absence of buffer overflows). The aim of trusted quality guarantees for the OSS components is the only technically feasible and economically viable alternative to the concept of trusted components, which is an illusive goal when certification is limited in the source code and the system context is only partially specified.

Certification is supported by the COPE toolset and is connected to the provision of verifiable evidence for the guaranteed qualities, as accompanying component assets in the COMPARE repository. It is inherently a normative product certification [Wal04], since it basically aims to establish evidence that the OSS code (product) conforms to some typical domain-specific architecture, which is the established norm. Depending on the quality property that is certified, evidence may be produced either by observation and measurement (empirical certification) or by formal and semi-formal means. Observation and measurement is typically employed in coverage-based testing, whereas semi-formal certification refers to static program analysis. When there is need for the more costly formal certification, a feasible alternative at an affordable cost may be the use of a certifying model checker [DKFW10] over the program slice, which is related to the property of interest.

OPEN-SME certification is procedural rather than fully mechanized, from the point of view that it is based on the role of a trained certifying agent, i.e. the reuse engineer. At the same time, the process is based on mature tool support for testing, static analysis and formal verification, fully interoperable with the COPE toolset for packaging the developed component assets. Certification is not driven by some standard that defines how compliance is enforced, but it is guided by a domain-specific OSS reuse de facto standard, which is the underlying typical architecture. Table 1 summarizes the main characteristics of the OPEN-SME certification profile, which were discussed in the preceding paragraphs.

4.1 On the adoption of a software assurance classification system

A software assurance classification system determines the level of confidence required for certifying that the software fulfills the anticipated qualities. Certification costs increase with increasing the required confidence for a software or a software component. The system used for

Characteristic	Comment
Normative certification	Conformance to a domain-specific typical architecture.
Trusted product	OSS source code of the reusable components.
Trusted qualities	Contextual qualities specified in the domain-specific architecture and local component qualities.
Formal and empirical certification	Combines observation and measurement (testing) with semi-formal and formal certification (static analysis and model checking).
Procedural certification	Carried our by a trained reuse engineer with appropriate tool support.
Absence of definition for compliance to a specific standard	Based on a de facto standard: the domain-specific typical architecture.

Table 1: OPEN-SME certification profile

software assurance classification depends on the certified quality and implies certain tasks in the certification process for establishing the required level of confidence. For a security certified software we adopt the Common Criteria [ISO99] classification system with seven *evaluation assurance levels (EALs)* as shown in Figure 7. For a safety critical software the level of confidence placed on the individual components depends on their *safety integrity levels (SILs)* that are assigned by an appropriate SIL allocation algorithm [PWR+10].

A quality guarantee for an OSS component is certified by the reuse engineer at a specific assurance level. The certificate (tests, design information etc.) is packaged together with all other component assets and is stored in the COMPARE repository. The reuse engineer may utilize assets provided by the OSS project that have to be validated, but if a certain degree of assurance is to be achieved this may incur some re-engineering effort. Validation of existing assets or production of new certification assets are supported by appropriate verification and validation tools and the COPE toolset. The assurance level at which a software component is certified is reflected in the packaged assets.

4.2 Certificate life cycle and change management

OPEN-SME certificates will be verifiable by the component re-user, under the condition that the re-user has access to appropriate tool support. Under this requirement, test-based certificates take the form of test suites and oracles for a certain code coverage level (verifiable condition). Static analysis certificates are represented by code annotations that express the certified requirements. These annotations depend on the used programming language (e.g. Java JSR-305 annotations), as well as the tool support required for verifying them. Finally, a formal certificate includes code annotations (preconditions, postconditions and invariants), a state-based behavior representation

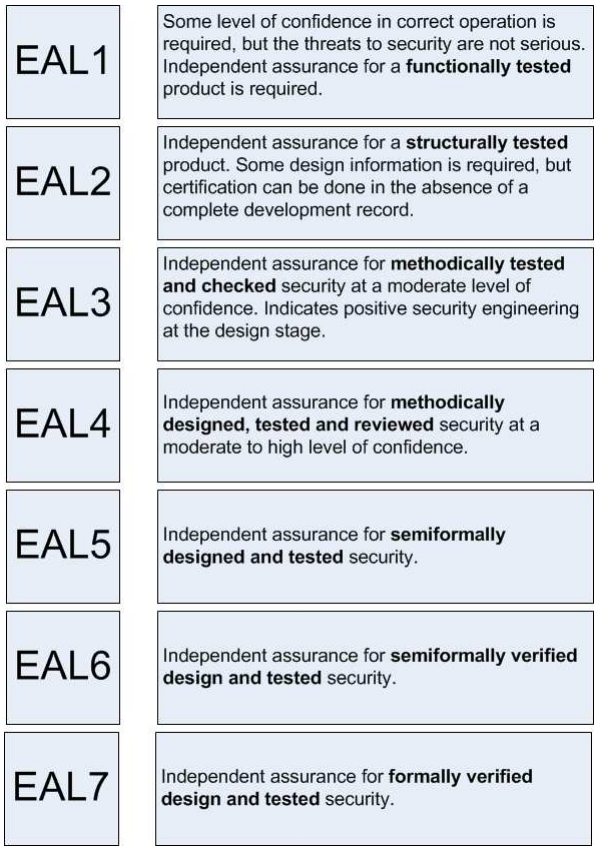

Figure 7: Common Criteria Assurance Classification

of the certified subject and a property specification expressed in a formal logic system.

Authenticity of certificates is assured by code file signing, so that the re-user can verify that the code has not been altered or corrupted since it was signed. For this purpose, OPEN-SME builds on the GPG Signing and Encrypting technology, which is supported by the COMPARE repository and the OPEN-SME Application Engineering process.

The certification process has to cope with the fast and agile OSS development processes and this implies a number of requirements for certificate handling and life cycle and version management. Toward this aim, COMPARE will provide functions for tracking the dependencies between issued certificates for a domain architecture. Also, the COPE toolset will be able to automatically recompute certificates for new component releases and relate them to a history of certifications for the system level quality guarantees.

4.3 Certification by testing

In OPEN-SME, certification by testing is supported by third-party tools. Both functional and structural testing processes are available and the reuse engineer selects the one prescribed by the aimed assurance level. In functional testing, which is also called model-based testing the reuse

engineer certifies the absence of design errors, some of which may be not possible to detect by structural testing. Examples of such quality guarantees are the absence of deadlocks, livelocks or that a forbidden application state cannot be reached.

The System Under Test (SUT) is represented by a *model program* derived by (i) data abstraction from the SUT's variables, (ii) behavioral abstraction from the SUT program statements and (iii) environmental abstraction, where the environment effects are replaced by non-determinism. A finite state machine (FSM) is then generated that instantiates all possible "runs" of the model program. An exploration algorithm searches the FSM for specification or design errors. It is important that exploration is limited only to appropriate scenarios for the purpose of the certification, such as to avoid a possible state space explosion. The model-based testing approach that is integrated to the OPEN-SME certification process is the one described in [JVCS07]. However, this is not enough in order to conclude the process. Certification also includes a precise definition of a conformance relation between the SUT and the model program. The software is certified only when the SUT conforms to the model program, which can be shown by the conformance testing approach in [FT07] for component-based systems. An important challenge for the OPEN-SME process is to effectively deploy a truly compositional approach that will allow to take into account individual specifications for the components' behavior (test-by-contract paradigm).

In structural testing, components will be tested in isolation from the rest of the system using stubs and drivers in place of the interfacing components. The SUT is certified with the same test case design technique (branch/decision coverage) for all interacting components or else, if there are components that have been already certified with different techniques then the validity of the certification depends on the subsumption relations between the used techniques. OPEN-SME tool support will also automate structural testing starting with the generation of test oracles from the component contracts.

Verifiability of the testing certificates will be assured by code coverage metrics and measurements for the certified components.

4.4 Certification by static analysis

Static program analysis became recently an attractive alternative, because of the advent of efficient and accurate enough analyses. These analyses are easily accessible in mature static analysis tools with good extensibility prospects. They are particularly effective in local component qualities like the absence of dangerous vulnerabilities.

Contextual properties that can be certified include temporal safety guarantees that are checked by typestate tracking, as well as high level security properties like information flow guarantees that are checked by taint propagation analyses. However, compositional certification is still an open research problem [CC02] and there is no reported success story in the current state of practice.

5 Related Work

A number of projects attempted to provide quality indicators for OSS software and to improve the trustworthiness of OSS. In this section we review the most important of these projects.

5.1 SQO-OSS and Alitheia core

Alitheia-core [GS09] was developed as part of the Software Quality Observatory for Open Source Software (SQO-OSS) European Community's Sixth Framework Programme. Alitheia-core is an extensible platform for quality analysis of software projects which integrates with a diverse set of revision control systems (e.g. SVN, GIT etc.), bug tracking systems (e.g. Bugzilla, Mantis etc.), mailing lists, Wiki documentation systems etc. that are used in the development of open source software projects. The need to interface with all these systems required a common representation of metadata that eliminates the need of handling multiple data formats at the higher layers of the system. The architecture of Alitheia-core is a three-tier architecture with a Results and Presentation tier, a System Core tier and a Data Mirroring, Storage and Retrieval tier. The Data Mirroring, Storage and Retrieval tier is responsible for storing project metadata and metric results and uses an object-relational mapping technology for transaction management and mapping of runtime types to database data types. For performance and scalability there is a preprocessing phase each time a project is registered with the system for the extraction of metadata and the storage of these metadata to the database layer. A variety of quality plugins which are implemented as OSGi components are then used for accessing these metadata and calculating different metrics which are necessary for quality analysis. Researchers interested in using the system for different quality analyses than the ones provided out of the box, can develop their own quality analysis plugins. To ease this extensibility and mask the OSGi details from the developers of extension plugins, the system provides a skeleton plugin that the developers can extend to build their own. Plugins can have activation types to enable the update of stored values when corresponding artifacts change and scope which determines the set of artifacts that are used for the metrics that a plugin calculates. Furthermore plugins can have dependencies to handle composite metrics whose values depend on the values of other metrics. Dependencies determine the execution order of the respective plugins which calculate the dependent metrics. Several metrics are already provided, but as we already mentioned the system is specifically designed to allow for easy integration of more metrics as required by software engineering research.

5.2 The QualiPSO Approach and Toolset

Another EU funded project in the context of the 6th FP is QualiPSO . QualiPSO [BLM+10] is an attempt to establish a methodology and develop the respective tools for the trustworthiness of OSS software. The general approach is that trustworthiness can be established by subjective and objective criteria. Since trustworthiness is a multi-faceted quality there are several elementary quality characteristics that need consideration such as 'As-is Utility', 'Functionality', 'Reliability', 'Security' and so on. For each such quality property the subjective model evaluates it by collecting the users' opinion. On the other hand for the objective model a number of metrics that can be collected automatically from tools are used. Statistical evaluations carried out in the context of the project established that the subjective (user perceived) quality of trustworthiness is linked with reliability, usability, interoperability, efficiency and documentation and that other factors such as the popularity and the development language are insignificant. Similarly there are significant correlations between trustworthiness and reliability and measurable (objective) characteristics such as the size of the code base and its complexity. A number of tools were

developed to support the project's goals:

- *Spago4Q* is a platform that integrates quality measurements collected by different tools and provides a quality indication visually to its users. Measurements tools are used by Spago4Q via extractors. These extractors are used to trigger the analysis by the lower-level tools or to collect the results of existing measurements from the database. Spago4Q is free software distributed under the GNU Lesser General Public License (LGPL).

- *MACXIM* is a tool that statically analyzes the source code of a project and extracts several metrics. Currently it supports the Java language. MACXIM separates the parsing phase from the analysis phase. Parsing generates an abstract representation of the source code which is saved as XML in a database. Queries (in XQuery or in Java) are then used to extract metrics from the XML abstract representation of the source code in the database. These metrics can be visualized either directly by MACXIM own interface or through Spago4Q which integrates it. The tool computes 70 metrics at application, package, class and method level. The tool is free software distributed under the GNU Lesser General Public License (LGPL).

- *JaBUTi* is a testing coverage tool that operates on the Java bytecode language and can be used without the program source code. It analyzes the test suite of a Java program against control and data flow testing criteria and produces six metrics that can be used for the evaluation of the reliability and correctness of an OSS product. JaBUTi is partially integrated to the Spago4Q platform and is free software distributed under the GNU General Public License (GPL).

There is also a Goal-Question-Metrics (GQM) tool which can be used for the application of the GQM methodology, but this tool is not yet publicly available or integrated in the Spago4Q platform. Also the Spago4Q platform integrates a number of other tools, such as StatSVN and StatCVS for statistical analysis of software repositories, FOSSology for license information and JUnit for collecting test status data. PMD and Checkstyle for the analysis of possible bugs, dead code, code duplications and other problems have been integrated in the MACXIM tool.

5.3 FLOSSMETRICS

FLOSSMETRICS [FLO10] is another European funded project with the central goal of producing quantitative analysis results for thousands of open source projects and making this data publicly available for different studies of OSS including quality studies. Other goals of the project include the provision of several high-level analyses of the results (e.g. evolution analysis of OSS projects), a guide for Small and Medium Enterprises and the development of several OSS tools suitable for the retrieval and analysis of OSS project data. Currently in the Melquiades website [Mel10] there is data available for 2,630 OSS projects. This data can be accessed by various means including the following: (1) Using a REST API client scripts can access the data directly, (2) Direct access to Melquiades database is also possible but limited (after request), (3) Database dumps can be downloaded directly. The Melquiades database is MySQL and the dumps are compressed images of the files produced by the mysqldump command. The dumps can be for individual projects and also aggregated dumps for all projects but for different repositories.

The kinds of repositories that the system aggregates are Software Configuration Management systems, Mailing lists and Issue tracking systems, (4) A variety of metrics is also available including code metrics (e.g. source code lines, cyclomatic complexity etc.), (5) Researches can also use a number of predefined queries to analyze the data such as the total number of commits, the number of commits per time unit etc.

In order to get this data FLOSSMETRICS developed a number of OSS tools that access the data and these tools are also available for further extension and development.

- *SVSAnalY* : The purpose of this tool is to retrieve information from CVS or Subversion repositories and store this information in a database. The database schema is divided in two parts. The first part contains information retrieved directly from the repository log (e.g. files, people involved, branches, tags etc.). The second part (i.e. extensions) contains additional information such as metrics for source code files. CVSAnalY is free software and can be redistributed and modified under the terms of the GNU General Public License.

- *Bicho* : Bicho is Bug Tracking System (BTS) analyzer. As with CVSAnalY this is a command-based tool which retrieves data from the bug tracking system and inserts it in a database. The database schema contains several tables: a table of bugs, a table comments found in the BTS about each bug, and a table of attachments found in the BTS about each bug etc. Bicho is also free software under the GNU General Public License.

- *MLStat* : Mailing list analyzer tool. This is another command-line tool with the purpose of extracting information from a mailing list and storing it in a database. The database schema includes information on messages in the mailing list, the people and their association to messages etc. This program is also free software distributed under the GPL.

5.4 Software Certification Success Stories

Two of the most notable examples of software certification include the works reported in [SK04] and [HALM08].

In [SK04] the authors conducted a Common Criteria [ISO99] based certification of Linux. Their conclusions from this work are that:

- At a low level of evaluation assurance it is possible to certify an OSS like Linux in about four months

- There are outstanding difficulties with the lack of adequate documentation in OSS

The results of this project are promising for the goals of OPEN-SME, since OPEN-SME targets components and not complete systems (e.g. Linux). Therefore it is expected that the certification process for components will require less time and fewer expenses. Furthermore the cost factor of certification and the lack of documentation seem to indicate that an asynchronous service in respect to the reuse activity is essential, since it can provide missing data and documentation related to the certification task and amortize the cost through a new business model.

In [HALM08] the authors used a formal certification process for certifying the core of a system. Their approach used three partitions of the system form which only one (which is less than

10%) required formal verification. This reduction of the code to be verified resulted in fewer expenses. Their approach is based on a high-level state machine specification of the behavior of the software and a formal specification of the property. The code is annotated and then partitioned based on the property to be specified and only part of the code is taken into account. More specifically the code is partitioned into three partitions: (a) Event, (b) Trusted and (c) Other Code. The high-level specification and the property are translated to the language of a mechanical prover and it is subsequently proved that the specification satisfies the property. Then a mapping is developed from the preconditions and the postconditions of the high level specification to those of the annotated code and it is demonstrated that this partition of the code (i.e. the Event partition) is a refinement of the high level specification and that the trusted and Other Code partitions are benign.

6 Conclusions

In this paper we presented the approach of the OPEN-SME EU funded project [OPE10] which is a project aiming to support the reuse of OSS software components from software development SMEs. We started by reviewing a survey related to OSS reuse which guided the design of the OPEN-SME toolset. The OPEN-SME approach processes and tools as well as the project's approach to software certification were also discussed.

Bibliography

[Abe07] M. Aberdour. Achieving Quality in Open Source Software. *IEEE Software* 24:58–64, 2007.
doi:http://doi.ieeecomputersociety.org/10.1109/MS.2007.2

[BLM+10] V. del Bianco, L. Lavazza, S. Morasca, D. Taibi, D. Tosi. The QualiSPo approach to OSS product quality evaluation. In *Proceedings of the 3rd International Workshop on Emerging Trends in Free/Libre/Open Source Software Research and Development*. FLOSS '10, pp. 23–28. ACM, 2010.
doi:http://doi.acm.org/10.1145/1833272.1833277

[CC02] P. Cousot, R. Cousot. Modular Static Program Analysis. In *Proceedings of the 11th International Conference on Compiler Construction*. CC '02, pp. 159–178. Springer-Verlag, London, UK, 2002.
http://dx.doi.org/10.1007/3-540-45937-5_13

[CCL06] I. Crnkovic, M. Chaudron, S. Larsson. Component-Based Development Process and Component Lifecycle. In *International Conference on Software Engineering Advances (ICSEA'06)*. P. 44. IEEE, 2006.

[DKFW10] K. Dräger, A. Kupriyanov, B. Finkbeiner, H. Wehrheim. SLAB: A Certifying Model Checker for Infinite-State Concurrent Systems. In *TACAS*. Pp. 271–274. 2010.
http://dx.doi.org/10.1007/978-3-642-12002-2_22

[FLO10] FLOSSMetrics Consortium. FLOSSMetrics Final Report. Technical report, March 2010.

[FT07] L. Frantzen, J. Tretmans. Model-based testing of environmental conformance of components. In *Proceedings of the 5th international conference on Formal methods for components and objects*. FMCO'06, pp. 1–25. Springer-Verlag, 2007. http://dx.doi.org/10.1007/978-3-540-74792-5_1

[Gar84] D. Garvin. What Does 'Product Quality' Really Mean? *Sloan Management Review* 26:25–43, 1984.

[GS09] G. Gousios, D. Spinellis. Alitheia Core: An extensible software quality monitoring platform. In *IEEE 31st International Conference on Software Engineering (ICSE 2009)*. Pp. 579–582. May 2009. doi:10.1109/ICSE.2009.5070560

[HALM08] C. Heitmeyer, M. Archer, E. Leonard, J. McLean. Applying Formal Methods to a Certifiably Secure Software System. *IEEE Transactions on Software Engineering* 34:82–98, 2008. doi:http://doi.ieeecomputersociety.org/10.1109/TSE.2007.70772

[HSC04] J. Ho-Won, K. Seung-Gweon, C. Chang-Shin. Measuring Software Product Quality: A Survey of ISO/IEC 9126. *IEEE Software* 21:88–92, 2004. doi:http://doi.ieeecomputersociety.org/10.1109/MS.2004.1331309

[ISO99] Evaluation Criteria for IT Security, parts 1 to 3. 1999.

[ISO01] Software Engineering Product Quality Part 1 - Quality Model, 1st ed. 2001.

[JVCS07] J. Jacky, M. Veanes, C. Campbell, W. Schulte. *Model-Based Software Testing and Analysis with C#*. Cambridge University Press, 2007.

[KP96] B. Kitchenham, S. Pfleeger. Software Quality: The Elusive Target. *IEEE Software* 13:12–21, January 1996. doi:10.1109/52.476281

[Lat10] Lattix Website. http://www.lattix.com, November 2010.

[Mel10] Melquiades Data Website. http://melquiades.flossmetrics.org/, November 2010.

[Mey03] B. Meyer. The grand challenge of trusted components. In *Software Engineering, 2003. Proceedings. 25th International Conference on*. Pp. 660 – 667. May 2003. doi:10.1109/ICSE.2003.1201252

[MMYA02] H. Mili, A. Mili, S. Yacoub, E. Addy. *Reuse-Based Software Engineering: Techniques, Organization and Controls*. Wiley, 2002.

[NAS10] NASA Earth Science Data Systems - Software Reuse Working Group. Reuse Readiness Levels (RRLs). Technical report, April 2010.

[OPE10] Open-SME Website. http://opensme.eu, November 2010.

[PWR+10] Y. Papadopoulos, M. Walker, M.-O. Reiser, M. Weber, D. Chen, M. Törngren, D. Servat, A. Abele, F. Stappert, H. Lonn, L. Berntsson, R. Johansson, F. Tagliabo, S. Torchiaro, A. Sandberg. Automatic allocation of safety integrity levels. In *Proceedings of the 1st Workshop on Critical Automotive applications: Robustness and Safety*. CARS '10, pp. 7–10. ACM, New York, NY, USA, 2010. doi:http://doi.acm.org/10.1145/1772643.1772646

[Ray01] E. S. Raymond. *The Cathedral and the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2001.

[RHR09] A. Rohatgi, A. Hamou-Lhadj, J. Rilling. Approach for solving the feature location problem by measuring the component modification impact. *IET Software* 3(4):292–311, 2009. doi:10.1049/iet-sen.2008.0078

[SK04] K. Shankar, H. Kurth. Certifying Open Source-The Linux Experience. *IEEE Security and Privacy* 2:28–33, 2004. doi:http://doi.ieeecomputersociety.org/10.1109/MSP.2004.96

[STR10] Structure101 Website. http://www.headwaysoftware.com, November 2010.

[Wal04] K. C. Wallnau. Software Component Certification: 10 Useful Distinctions. Technical report, Software Engineering Institue, September 2004.