Trustworthy Software Development

Sachar Paulus[1], Nazila Gol Mohammadi[2], and Thorsten Weyer[2]

[1] Department of Economics, Brandenburg University of Applied Sciences,
14770 Brandenburg an der Havel, Germany
paulus@fh-brandenburg.de
[2] paluno – The Ruhr Institute for Software Technology,
University of Duisburg-Essen, 45127 Essen, Germany
{nazila.golmohammadi,thorsten.weyer}@paluno.uni-due.de

Abstract. This paper presents an overview on how existing development methodologies and practices support the creation of trustworthy software. Trustworthy software is key for a successful and trusted usage of software, specifically in the Cloud. To better understand what trustworthy software applications actually mean, the concepts of trustworthiness and trust are defined and put in contrast to each other. Furthermore, we identify attributes of software applications that support trustworthiness. Based on this groundwork, some well-known software development methodologies and best practices are analyzed with respect on how they support the systematic engineering of trustworthy software. Finally, the state of the art is discussed in a qualitative way, and an outlook on necessary research efforts and technological innovations is given.

Keywords: Software development, Trustworthiness, Trust, Trustworthy software, Trustworthy development practices.

1 Introduction

In the last years, many attempts have been made to overcome the issue of insecure and untrusted software. A number of terms have been used to catch the expectation on how "solid" a piece of software should be: secure, safe, dependable and trusted. Only in recent years literature related to (secure) software developments has seen the introduction of socio-technical systems (STS) (for more details, see [1]). This concept allows to distinguish between the actual trust that users of software put into the functioning / delivery of the software in questions on the one side, and the trustworthiness of the software, i.e. properties (we call them attributes) that justify the trust that users put "into" the software. Whereas trust should primarily be the subject of the "maintenance" of the relationship between the user and the software in use ("in operations"), trustworthiness is primarily acquired during the development process of the software and can mostly only be "lost" later on.

The software creation process, neither, has been addressed adequately both in theory and practice until recently regarding topics like trust, trustworthiness or similar, except either purely theoretical approaches (such as formal proofs or other forms

B. De Decker et al. (Eds.): CMS 2013, LNCS 8099, pp. 233–247, 2013.

of verification (e.g. [2]) or on a functional level only (using e.g. security patterns [3]). As such, an analysis of existing software development practices / methodologies with a specific view on trustworthiness is new to the field. This research has been carried out as part of the OPTET project, and the results will be presented in this paper in adequate detail. As an overview publication, it summarizes results of other very recent publications [1].

This paper is structured as follows: in a first section, we define the notions of trust and trustworthiness and introduce the concept of trustworthiness attributes. The next section presents the analysis of the different development methodologies and practices in light of trustworthiness, followed by an analysis section on the state-of-the-art to summarize what is available today, and where there is more research needed to achieve the goal of trustworthy software. A last section summarize the research carried out and shortly indicates the future work planned in the OPTET project.

2 Fundamentals

In this section we introduce the two basic concepts “trust” and “trustworthiness” in order to be able to analyze how trustworthiness is addressed by different software development disciplines. Both concepts focus on the outcome of the STS but are different in the view of the trustor and trustee(s) perspective. In general, trust is the trustor's prior estimation that an STS will provide an appropriate outcome, while trustworthiness is the probability that the same STS will successfully meet all of the trustors’ requirements. The balance between trust and trustworthiness is a core issue for software development because any imbalance (over-cautiousness or misplaced trust) could lead to serious negative impact, e.g. concerning the acceptance of the software by its (potential) users.

2.1 The Notion “Trust”

We define trust in a system as a property of each individual trustor, expressed in terms of probabilities and reflecting the strength of their belief that engaging in the system for some purpose will produce an acceptable outcome. Thus, trust characterizes a state where the outcome is still unknown, based on each trustor's subjective perceptions and requirements. A stakeholder would decide to place trust on an STS if his trust criterion was successfully met; in other words, their perceptions exceed or meet its requirements. A trustor having engaged in a system for multiple transactions can (or will) update the current trust level of that STS by observing past outcomes.

A presence of subjective factors in trust decisions means that two different trustors may have different levels of trust for the same STS to provide the same outcome in the future, even if they both have observed exactly the same system outcomes in the past. More specifically, subjective perceptions can depend on trustor attributes, which capture social factors such as age, gender, cultural background, level of experience with Internet-based applications, and view on laws. Subjective requirements, on the other hand, are represented by so-called trust attributes that quantify the anticipated

utility gains or losses with respect to each anticipated outcome. Thus, relatively high levels of trust alone may not be adequate to determine a positive decision (e.g., if the minimum thresholds from requirements are even higher). Similarly, it is possible to engage in a system even if one's trust for an acceptable outcome is low (e.g., if the utility gains from this outcome are sufficiently high).

2.2 The Notion "Trustworthiness"

We regard trustworthiness as an objective property of the STS, based on the existence (or nonexistence) of appropriate properties and countermeasures that reduce the likelihood of unacceptable outcomes. A stakeholder (e.g., the system designer, a party performing certification) shall decide to what extent a system is trustworthy based on trustworthiness criteria. These criteria are logical expressions in terms of systems attributes, referred to as quality attributes. For example, trustworthiness may be evaluated with respect to the confidentiality of sensitive information, the integrity of valuable information, the availability of critical data, the response time or accuracy of outputs. Such quality attributes shall be quantified by measuring systems' (or individual components') properties and/or behavior. Objectivity in assessing trustworthiness for a particular attribute is based on meeting certain predefined metrics for this attribute or based on compliance of the design process for this attribute to our predefined system specifications.

Thus, the trustworthiness of an STS may be evaluated compared to a target performance level, or the target may be its ability to prevent a threat from becoming active. Such issues are defined by the trustworthiness attributes that have a dual interpretation. Until recently, trustworthiness was primarily investigated from a security or loyalty perspective while assuming that single properties (certification, certain technologies or methodologies) of services lead to trustworthiness and even to trust in it by users. Compared to this approach, we reasonably assume that such a one-dimensional approach is insufficient to capture all the factors that contribute to an STS's trustworthiness and instead we consider a multitude of attributes.

In this paper, our definition for trustworthiness attributes reflects the design-time aspects. A trustworthiness attribute in this sense is a property of the system that indicates its capability to prevent potential threats to cause an unexpected and undesired outcome, e.g., a resilience assurance that it will not produce an unacceptable outcome.

2.3 Trustworthiness of a Software Application

In order to prove to be trustworthy, software applications could promise to cover a set of various quality attributes [1],[4] depending on their domain and target users. Trustworthiness should promise a wide spectrum including reliability, security, performance, and user experience. But trustworthiness is domain- and application-dependent, and a relative attribute that means that if a system is trustworthy with respect to some Quality of Service (QoS) like performance, it would not necessarily be successful in being secure. Consequently, trustworthiness and trust should not be regarded as a single construct with a single effect, they are rather strongly context

dependent in such a way that the criteria and measures for objectively assessing the trustworthiness of a software application are based on specific context properties, like the application domain and the user groups of the software.

A broad range of literature has argued and emphasized the relation between QoS and trustworthiness (e.g. [5],[6],[9],[10],[11]). Therefore, trustworthiness is influenced by a number of quality attributes other than just security-related.

In the context of this work we strictly adhere to the perspective of a to-be-constructed system, and therefore will ignore potential trustworthiness attributes that are at the earliest available at runtime, like reputation or similar concepts representing other users‘ feedback. Additionally, some literature proposes quality attributes (e.g. authentication, authorization, data encryption or access control), that refer to means for achieving certain properties of a system. These means are not reflecting attributes but defining means for establishing the corresponding attributes within the system. Such “attributes” were not within the scope of our analysis.

In prior work, we have investigated the properties and attributes of a software system that determines the trustworthiness of the system. To this end, based on the S-Cube quality reference model [7], we built a taxonomy of attributes (shown in Fig. 1) that is a foundation to define objective criteria and measures to assess the trustworthiness of a system. Some quality attributes referenced in the literature (e.g. [8],[12],[13],[14]) refer to means for achieving a certain kind of property of a system. Therefore, we do not consider them as trustworthiness attributes, but as means to manifest the corresponding properties to the system. Only the attributes contributing to trustworthiness identified in literature review is included in the model. Some quality attributes, e.g. integrity, can be achieved, among other ways, through encryption. In this case, the high-level attribute (integrity) is included as a contributor to trustworthiness, but not encryption because it is encompassed by the higher-level attribute.

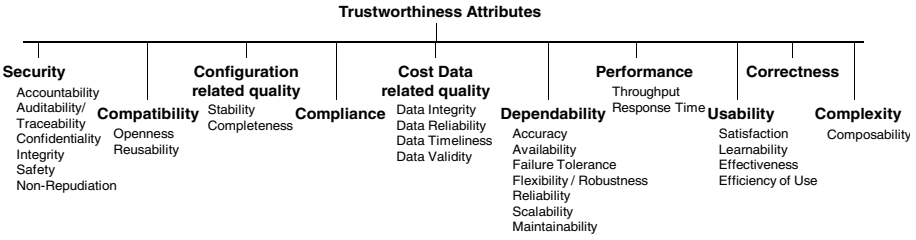

Fig. 1. Attributes that determine trustworthiness of a software application during development

We have included attributes that have been studied in the literature in terms of trustworthiness. Fig. 1 outlines the major result of that work. More details can be found in [2]. Actually, we have identified some additional attributes that are candidates for attributes that influence the trustworthiness of a system (e.g. provability, or predictability). These potential trustworthiness attributes need further investigation on their impact on trustworthiness.

Based on these trustworthiness attributes, we have studied several software design methodologies with respect to the extent in which these methodologies address the

systematic realization of trustworthiness to a system under development. In next section, the result and evaluation of these studies is presented.

3 Review of Development Models and Practices

Recently, a number of development practices have been proposed, both from a theoretical as well as from a practical point of view, to address security of the software to-be-developed. As described above, security is an important component of trustworthy software, but neither is it the only one, nor will it be sufficient to look solely at preserving / creating a good level of security to attain trustworthiness. For example, transparency plays an important role for the creation of trust, and therefore for the trustworthiness of software.

In this section, we will look into the major software engineering processes or process enhancements that target security to build a „secure“ software system and identify corresponding innovation potential, specifically towards extending security to trustworthiness. A more exhaustive overview of development methodologies can for instance be found in Jayaswal and Patton's "Design for Trustworthy Software" [20], though it does not specify how these methodologies contribute to the trustworthiness of the product. This reference documents their generic characteristics and an overview of the historical evolution of different development strategies and lifecycle models.

We will briefly describe which elements of the development approaches will actually increase or inhibit trust, and how the approaches could be used for modeling trustworthiness.

3.1 Plan-Driven

In a plan-driven process [17] one typically plans and schedules all of the process activities before the work can start. The *Waterfall model* is a well-known example of plan-driven development that typically includes the following phases:

- Requirements analysis
- System design
- Implementation
- Testing (unit, integration and system testing)
- Deployment
- Operation and maintenance

Many of the simplistic software manufacturing projects follow a plan-driven model. This approach has been followed by industrial software development for a long time. It is relatively easy to assure non-functional requirements throughout the rest of the process, but the key issue is that they need to be identified completely in the first phase. Plan-driven processes such as the Waterfall model originate from aerospace and other manufacturing industries, where robustness and correctness is usually an important concern, but are often considered being too rigorous, inflexible and a bit old-fashioned for many software development projects. There are examples of

Waterfall trustworthy software development processes in the literature, e.g. COCOMO. Therefore, there should be means to assure trustworthiness and enhance the process. There can be more formal variants of this process, for instance the B method [21], where a mathematical model of the specification is created and then automatically transferred into code. For the general plan-driven process we consider the following trustworthiness characteristics to be valid:

Trustworthiness gains:

- Formal system variants are well suited to the development of systems that have stringent safety, reliability or security – and thus potentially also trustworthiness – requirements.

Trustworthiness losses:

- Vulnerable with vague, missing or incorrect security and trustworthiness requirements in the first place.
- Does not offer significant cost-benefits over other approaches, which on a tight budget can lead to less focus on trustworthiness.
- Little flexibility if new attacks or types of vulnerabilities are discovered late in the development process.
- Usability for modeling trustworthiness

In a plan-driven process one can apply structured testing on units as well as on a system as a whole. In addition, it is relatively easy to keep track of the implementation of safety, reliability or security and potentially also trustworthiness requirements. As such, the plan-driven approach supports modeling in general, but not specifically for trustworthiness.

3.2 Incremental

Incremental development (cf. [19]) represents a broad range of related methodologies where initial implementations are presented to the user at regular intervals until the software satisfies the user expectations (or the money runs out). A fundamental principle is that not all requirements can be known completely prior to development. Thus, they are evolving as the software is being developed. Incremental development covers most of the agile approaches and prototype development, although it could be enhanced by other approaches to become more formal in terms of trustworthiness.

Trustworthiness gains:

- New and evolving requirements for trust may be incorporated as part of an iterative process.
- The customer will have a good sense of ownership and understanding of the product after participating in the development process.

Trustworthiness losses:

- Mismatch between organizational procedures/policies and a more informal or agile process.
- Little documentation, increasing complexity and long-lifetime systems may result in security flaws. Especially, documentation on non-functional aspects that are crosscutting among different software features implementation could not be well documented.
- Security and trustworthiness can be difficult to test and evaluate, specifically by the user, and may therefore lose focus on the development.

Incremental development allows new and evolving requirements for trustworthiness to be incorporated as part of an iterative process. Iterative processes allow for modeling of properties, but changes to the model that reflect changed or more detailed customer expectations, will in turn require changing the design and code, eventually in another iteration. Additionally, there are no specific trustworthiness modeling capabilities.

3.3 Reuse-Oriented

Very few systems today are created completely from scratch; in most cases there is some sort of reuse of design or code from other sources within or outside the organization (cf. [19]). Existing code can typically be used as-is, modified as needed or wrapped with an interface. Reuse is of particular relevance for service-oriented systems where services are mixed and matched in order to create larger systems. Reuse-oriented methodologies can be very ad-hoc, and often there are no other means to assure trustworthiness.

Trustworthiness gains:

- The system can be based on existing parts that are known to be trustworthy. This does not, however, mean that the composition is just as trustworthy as the sum of its parts.
- An existing, trustworthy part may increase trust (e.g. a known, trusted authentication).

Trustworthiness losses:

- Use of components that are "not-invented-here" leads to uncertainty.
- Increased complexity due to heterogeneous component assembly.
- The use of existing components in a different context than originally targeted may under certain circumstances (.e.g. unmonitored re-use of in-house developed components) jeopardize an existing security / trustworthiness property.

This approach has both pros and cons regarding trustworthiness modeling. On the positive side, already existing, trustworthy and trusted components may lead to easier,

trustworthiness modeling for the overall solution; adequate software assurance, e.g. a security certification, or source code availability may help in improving trustworthiness of re-used "foreign" components. The drawback is that there is a risk that the trustworthiness of the combined system may decrease due to the combination with less trustworthy components.

3.4 Model-Driven

Model-driven engineering (MDE) [22] (encompassing the OMG term Model-driven Architecture (MDA) and others) refers to the process of creating domain models to represent application structure, behavior and requirements within particular domains, and the use of transformations that can analyze certain aspects of these models and then create artifacts such as code and simulators. A lot of the development effort is put into the application design, and the reuse of patterns and best practices is central during the modeling.

Trustworthiness gains:

- Coding practices that are deemed insecure or unreliable can be eliminated through the use of formal reasoning.
- Coding policies related to trustworthiness, reliability and security could be systematically added to the generated code.
- Problems that lead to trustworthiness concerns can, at least theoretically, be detected early during model analysis and simulation.
- Separation of concerns allows trust issues to be independent of platform, and also less complicated models and a better combination of different expertise.

Trustworthiness losses:

- Systems developed with such methods tends to be expensive to maintain, and may therefore suffer from lack of updates.
- Requires significant training and tool support, which might become outdated.
- A structured, model-driven approach does not prevent the forgetting of security and trustworthiness requirements.
- Later changes during development need to review and potentially change the model.
- The (time and space) complexity of the formal verification of especially non-functional properties may lead to omitting certain necessary computations when the project is under time and resource pressure.

With a model-driven approach it is possible to eliminate deemed insecure or unreliable design and coding practices. An early model analysis and simulation with regards to trustworthiness concerns is possible and of high value. In addition, model-driven security tests could improve the trustworthiness. However, in general, there are no specific trustworthiness related modeling properties, it is just model-driven.

The major drawback (and risk) is that the computational complexity for verifying non-functional properties is very high.

3.5 Test-Driven

Test-driven development is considered to be part of agile development practices. In test-driven development, developers first implement test code that is able to test corresponding requirements, and only after that the actual code of a module, a function, a class etc. The main purpose for test-driven development is to increase the test coverage, thereby allowing for a higher quality assurance and thus requirements coverage, specifically related to non-functional aspects. The drawback of test-driven approaches consists in the fact that due to the necessary micro-iterations the design of the software is subject to on-going changes. This makes e.g. the combination of model-driven and test-driven approaches rather impossible.

Trustworthiness gains:

- The high degree of test coverage (that could be up to 100%) assures the implementation of trustworthiness related requirements.

Trustworthiness losses:

- The programming technique cannot be combined with (formal) assurance methodologies, e.g. using model-driven approaches, Common Criteria, or formal verification.

Test-driven development is well suited for assuring the presence of well-described trustworthiness requirements. Moreover, this approach can be successfully used to address changes of the threat landscape. A major drawback, though, is that it cannot easily be combined with modeling techniques that are used for formal assurance methodologies.

3.6 Common Criteria ISO 15408

The Common Criteria (CC) is a standardized approach [24] to evaluate security properties of (information) systems. A “Target of Evaluation” is tested against so-called ”Security Targets” that are composed of given Functional Security Requirements and Security Assurance Requirements (both addressing development and operations) and are selected based on a protection requirement evaluation. Furthermore, the evaluation can be performed at different strengths called “Evaluation Assurance Level”.

On the downside, there are some disadvantages: the development model is quite stiff, and does not easily allow for an adjustment to specific environments. Furthermore, Common Criteria is an „all-or-nothing“ approach, one can limit the Target of Evaluation or the Evaluation Assurance Level, but it is rather difficult to then express the overall security / trustworthiness of a system with metrics related to CC.

Trustworthiness gains:

- Evaluations related to security and assurance indicates to what level the target application can be trusted.
- CC evaluations are performed by (trusted) third parties.
- There are security profiles for various types of application domains.

Trustworthiness losses:.

- Protection profiles are not tailored for Cloud services.
- A CC certification can be misunderstood to prove the security / trustworthiness of a system, but it actually does only provide evidence for a very specific property of a small portion of the system.

The Common Criteria approach is unrelated to modeling in general, although the higher evaluation assurance levels would benefit from modeling. The functional security requirements may well serve as input for a (security-related) trustworthiness modeling, whereas the security assurance requirements, as the properties of the development process itself, shall be used for a modeling of the developing organization. Note that these constitute two different modeling approaches.

3.7 ISO 21827 Systems Security Engineering - Capability Maturity Model

Systems Security Engineering - Capability Maturity Model (SSE-CMM) is a specific application of the more generic Capability Maturity Model of the Software Engineering Institute at Carnegie Mellon University. Originally, in 1996 SSE-CCM was an initiative of the NSA, but was given over later to the International Systems Security Engineering Association, that published it as ISO 21827 in 2003. In contrast to the previous examples, SSE-CMM targets the developing organization and not the product / service to be developed. There are a number of so-called ”base practices“ (11 security base practices and 11 project and organizational base practices) that can be fulfilled to different levels of maturity. The maturity levels are identical to CMM.

Trustworthiness gains:

- The developing organization gains more and more experience in developing secure and more generically good quality software.
- The use of a quality-related maturity model infers that user-centric non-functional requirements, such as security and trustworthiness, will be taken into account.

Trustworthiness losses:

- This is an organizational approach rather than a system-centric approach; hence there is not really any guarantee about the trustworthiness of the developed application (which could e.g. be put to use in another way than it was intended for).

This approach focuses on the development of trustworthiness for the developing organization, instead on the to-be developed software, service or system. The security base practices may serve as input for modeling trustworthiness requirements when modeling the development process.

3.8 Building Security in Maturity Model / OpenSAMM

The Building Security In Maturity Model (BSIMM) [23] initiative has recognized the caveat of ISO 21827 being oriented towards the developing organization, and has proposed a maturity model that is centralized around the software to be developed. It defines activities in four groups (Governance, Intelligence, SSDL Touch points, Deployment) that are rated in their maturity according to three levels. OpenSAMM is a very similar approach that has the same origin, but developed slightly differently and is now an OWASP project.

This standard presents an ideal starting point for developing trustworthiness activities within an organization, since it allows tracking the maturity of the development process in terms of addressing security requirements – this could also be used for trustworthiness.

Trustworthiness gains:

- The maturity-oriented approach requires the identification of security (and potentially) trustworthiness properties and assures their existence according to different levels of assurance.
- The probability of producing a secure (and trustworthy) system is high.

Trustworthiness losses:

- There is no evidence that the system actually is trustworthy or secure.

This approach means to develop trustworthiness for the developing organization, instead of the to-be developed software, service, or system. The security base practices may serve as input for modeling trustworthiness requirements when modeling the development process.

3.9 Microsoft SDL

In 2001, Microsoft has started the security-oriented software engineering process that has probably had the largest impact across the whole software industry. Yet, the „process“ was more a collection of individual activities along the software development lifecycle than a real structured approach. The focus point of the Microsoft SDL - that has been adopted by a large number of organizations in different variants - is that every single measure was optimized over time to either have a positive ROI or it was dropped again. This results in a number of industry-proven best practices for enhancing the security of software. Since there is no standardized list of activities, there is no benchmark to map activities against.

Trustworthiness gains:

- The world‘s largest software manufacturer does use this approach.
- The identified measures have proven to be usable and effective over the course of more than a decade.

Trustworthiness losses:

- There is no evidence that the system actually is trustworthy or even secure.

Microsoft SDL is a development-related threat modeling and was Microsoft`s major investment to increase the trustworthiness of its products („Trustworthy Computing Initiative“). The comparability is only given if more detailed parameters are specified. For the modeling of trustworthiness, this method is only of limited help.

3.10 Methodologies Not Covered in This Paper

During the analysis process, a significant number of other methodologies and approaches have been investigated, among others, ISO 27002, OWASP Clasp or TOGAF. We dropped these here since they either replicate some of the capabilities already mentioned above or because their contribution to trustworthiness showed to be rather small.

4 Conclusions from the State of the Art Analysis

After having analyzed the different methodologies and best practices, we can make two major observations. The first observation is related to the nature of the methodologies and best practices. There are two major types of approaches:

- *Evidence-based* approaches that concentrate on evidences, i.e. some sort of qualitative “proof” that a certain level of security, safety etc. is actually met, and
- *Improvement-based* approaches that concentrate on improving the overall situation within the software developing organization with regards to more or less specific requirements.

Evidence-based approaches are typically relatively rigid and therefore often not used in practice, except there is an explicit need, e.g. for a certification in a specific market context. The origin of evidence-based approaches is either research or a strongly regulated market, such as e.g. the defense sector. In contrast to those, improvement-based approaches allow for customization and are therefore much better suited for the application in various industries, but lack in general the possibility to create any kind of evidence that the software developed actually fulfills some even fundamental trustworthiness expectations.

Assuming that evidence-based and improvement-based approaches are – graphically speaking – at the opposite ends of a continuous one-dimensional space, a way to improve trustworthiness of software applications might be to identify approaches that

are “sitting in between” these two types (for example, by picking and choosing elements of different approaches, augmented with some additional capabilities). One option might be to release the burden of qualitative evidence creation by switching to / encompassing evidences based on quantitative aspects. We propose to investigate how metrics for the trustworthiness attributes presented in Section 2 can be used to create evidences by applying selected elements of the improvement-based approaches.

A second major observation relates to the scope of the activities described in the methodologies and best practices. There are three types of “scope”:

- *Product-centric* approaches emphasize the creation and/or verification of attributes of the to-be-developed software,
- *Process-centric* approaches concentrate on process steps that need to be adhered to enable the fulfillment of the expected goal and
- *Organization-centric* approaches focus on the capabilities of the developing organization, looking at a longer-term enablement to sustainably develop trustworthy software.

Some approaches combine the scope, e.g. Common Criteria both mandates verifying product-related and process-related requirements, whereas others, such as SSE-CMM [25] concentrate on only one scope. Current scientific discussions targeting trustworthiness related attributes are mainly focusing on product-centric approaches which is very understandable given the fact that this is the only approach that focuses on evidences on the software itself, whereas practices used in industry often tend towards a more process- or even organization-centric approach (SSE-CMM, CMM, ISO 9001). We therefore propose to investigate how to evolve the above-mentioned evidence-based activities around metrics towards covering process- and organization-centric approaches.

5 Conclusion and Future Work

In this paper we presented an overview on how existing development methods and practices support the development of trustworthy software. To this aim, we first elaborated on the notion of trust and trustworthiness and presented a general taxonomy for trustworthiness attributes of software. Then we analyzed some well-known general software development methodologies and practices with respect on how they support the development of trustworthy software.

As we have shown in the paper, existing software design methodologies have some capacities in ensuring security. But, the treatment of other trustworthiness attributes and requirements in software development is not yet well studied. Trustworthiness attributes that have major impact on acceptance of STS, must be taken to account, analyzed, and documented as thoroughly as possible. In this way the transparency of the decisions under taken during the development will remove potentially the uncertainty of stakeholders of respective software.

The main ideas and findings of our work will be further investigated. It is important to understand how the trustworthiness attributes and the corresponding system properties can be addressed in the system to be in a systematic way. As a next step, we will investigate trustworthiness evaluation techniques for enabling and providing effective measurements and metrics to assess trustworthiness of systems under development. Furthermore, we will develop an Eclipse Process Framework (EPF) based plug-in that will support the process of establishing trustworthiness attributes into a system and guiding the developer through the development activities. Using this plug-in during the development process, the corresponding project team will be supported by guidelines, architectural patterns, and process chunks for developing trustworthy software and later on to analyze the results and evaluate the trustworthiness of the developed software.

Acknowledgements. This research was carried out with the help of the European Commission's 7th framework program, notably the project "OPTET". We specifically would like to thank all participants of Work Package 3 for contributing to the analysis of the methodologies and best practices.

References

1. Gol Mohammadi, N., Paulus, S., Bishr, M., Metzger, A., Koennecke, H., Hartenstein S., Pohl, K.: An Analysis of Software Quality Attributes and Their Contribution to Trustworthiness. In: 3rd International conference on Cloud Computing and Service Science (CLOSER), Special Session on Security Governance and SLAs in Cloud Computing – CloudSecGov, available in SCITEPRESS Digital Library, to appear in Springer-Verlag, SSRI, Aachen (2013)
2. Leveson, N., Stolzy, J.: Safety analysis using Petri nets. IEEE Transactions on Software Engineering 13(3), 386–397 (1987)
3. Schumacher, M., Fernandez-Buglioni, E., Hybertson, D., Buschmann, F., Sommerlad, P.: Security Patterns: Integrating Security and Systems Engineering. Wiley Series in Software Design. Wiley (2005)
4. Mei, H., Huang G., Xie, T.: Internetware: A software paradigm for internet computing, pp. 26–31. IEEE Computer Society (2012)
5. Araújo Neto, A., Vieira, M.: Untrustworthiness: A Trust-Based Security Metric. In: 4th International Conference on Risks and Security of Internet and Systems (CRiSIS), France, pp. 123–126 (2009)
6. San-Martín, S., Camarero, C.: A Cross-National Study on Online Consumer Perceptions, Trust, and Loyalty. Journal of Organizational Computing and Electronic Commerce 22, 64–86 (2012)
7. Chen, C., Wang, K., Liao, S., Zhang, Q., Dai, Y.: A Novel Server-based Application Execution Architecture. In: International Conference on Computational Science and Engineering, CSE 2009, vol. 2, pp. 678–683 (2009)
8. Harris, L.C., Goode, M.M.: The four levels of loyalty and the pivotal role of trust: a study of online service dynamics. Journal of Retailing 80(2), 139–158 (2004)
9. S-Cube: Quality Reference Model for SBA. S-Cube - European Network of Excellence (2008), http://www.s-cube-network.eu/results/deliverables/wp-jra-1.3/Reference_Model_for_SBA.pdf/view

10. ISO/IEC 9126-1: Software Engineering – Product quality – Part: Quality Model, International Organization of Standardization, Geneva, Switzerland (2001)
11. Gómez, M., Carbó, J., Benac-Earle, C.: An Anticipatory Trust Model for Open Distributed Systems. In: Butz, M.V., Sigaud, O., Pezzulo, G., Baldassarre, G. (eds.) ABiALS 2006. LNCS (LNAI), vol. 4520, pp. 307–324. Springer, Heidelberg (2007)
12. Yolum, P., Singh, M.P.: Engineering self-organizing referral networks for trustworthy service selection. IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans 35(3), 396–407 (2005)
13. Yan, Z., Goel, G.: An adaptive trust control model for a trustworthy component software platform. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 226–238. Springer, Heidelberg (2007)
14. Boehm, B.W., Brown, J.R., Lipow, M.: Quantitative Evaluation of Software Quality. In: Proceedings of the 2ndInternational Conference on Software Engineering (ICSE), pp. 592–605. IEEE Computer Society Press, Los Alamitos (1976)
15. Adrion, W., Branstad, M., Cherniavsky, J.: Validation, Verification, and Testing of Computer Software. ACM Computing Surveys 14, 159–192 (1982)
16. McCall, J.A., Richards, P.K., Walters, G.F.: Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality. US Department of Commerce, National Technical Information Service (NTIS), Final technical rept. (1977)
17. Royce, W.W.: Managing the Development of Large Software Systems: Concepts and Techniques. In: IEEE WESTCON, Los Angeles CA, pp. 1–9 (1970)
18. Boehm, B.: A Spiral Model of Software Development and Enhancement. IEEE Computer 21(5), 61–72 (1988)
19. Sommerville, I.: Software Engineering, 9th edn. Pearson, Boston (2011)
20. Jayaswal, B.K., Patton, P.C.: Design for Trustworthy Software: Tools, Techniques and Methodology for Developing Robust Software. Prentice Hall (2011)
21. Wordworth, J.: Software Engineering with B. Addison Wesley Longman (1996)
22. Schmidt, D.C.: Model-Driven Engineering. IEEE Computer 39(2), 25–31 (2006)
23. McGraw, G., Chess, B.: A Software Security Framework: Working Towards a Realistic Maturity Model. InformIT (October 2008)
24. ISO/IEC 15408:Information technology - Security techniques - Evaluation criteria for IT security - Part 1: Introduction and general model, Geneva, Switzerland (2009)
25. ISO/IEC 21827:2002: Information technology – Systems Security Engineering – Capability Maturity Model (SSE-CMM) Geneva, Switzerland (2002)