Chapter 4

Trustworthiness of IT Systems and Services

4.1 WHAT IS TRUSTWORTHINESS IN THE IT DOMAIN?

The evolution of information technology and its products is one of the most rapid evolutions in the history of humanity. There are signals, from an IT evolution point of view, that we are fast approaching a trustworthiness critical entry point. Some indicators supporting this hypothesis are the IT market size and recent estimations on data quantities being processed globally. As can easily be found if one surfs patiently over the Internet, different sources give rather similar information:

- In 2006, the amount of digital information created, captured, and replicated was 1,288 × 1,018 bits. In computer parlance, that is 161 exabytes or 161 billion gigabytes. This is about 3 million times the information in all the books ever written.
- Between 2006 and 2010, the information added annually to the digital universe increased more than six-fold, from 161 exabytes to 988 exabytes.

All these enormous numbers make necessary two simple questions to be asked:

- What happens if only 0.001% of important information becomes untrustworthy?
- What happens if some of the systems involved in processing these data go "untrustworthy"?

The answer to these questions makes the most real raison d'être for intensified IT trustworthiness development efforts.

What *trustworthiness* means in the IT domain is the result of many decades of evolution of different, not always closely related sciences such as psychology, sociology, engineering, and even history. The widely known definition of trustworthiness applied in software engineering domain has been proposed and published in

Software Quality Engineering: A Practitioner's Approach, First Edition. Witold Suryn.

Published 2014 by John Wiley & Sons, Inc.

Reference 1 and is related to conformance to requirements: "An entity is trustworthy if there is sufficient credible evidence leading one to believe that the system will meet a set of given requirements. Trust is a measure of trustworthiness, relying on the evidence provided."

The notion of trustworthiness applied in human relationships is being continuously discussed on different open fora, with one of the most interesting perspectives published on Wikipedia [2]: "Trustworthiness is a moral value considered to be a virtue. A trustworthy person is someone in whom we can place our trust and rest assured that the trust will not be betrayed."

The records of the Minnesota State Archives [3] give a very succinct yet precise illustration of what trustworthiness may mean for the history: "[Trustworthiness] refers to the reliability and authenticity of records."

Finally, the linguistic definition related to *engineering* that can be found in most online and printed dictionaries is: "Trustworthiness is an attribute of an entity deserving of trust or confidence, being dependable and reliable."

The key notions that can be withdrawn as a common denominator from all the above definitions are: reliability, credibility, and dependability. And, in the essence, they represent the lion's share of what a contemporary IT user (consumer) would expect from software or system that processes his or her *sensitive* information. However, to be true to the reality, to present a twenty-first century user's perception of trustworthiness, a larger list of attributes must be drawn:

- Quality
- Reliability
- Credibility
- Dependability
- Completeness of required functions
- Proper quality–cost ratio (so the software was not overpaid)
- Post-sale maintenance and service
- Pre- and post-sale training
- Documentation
- Responsibility for the product.

Some of these elements may seem secondary in comparison with, for example, quality or dependability, but the real market shows cases where suppliers have lost the trust of their customers after offering "lousy" post-sale support.

The introduction of the Internet and wireless broadband networks together with service-oriented architectures has facilitated the composition of interactive services using web technologies and web communities.

The interrelationships and interdependencies between formerly stand-alone systems and networks are leading to complexities in the infrastructures of our society that have never been seen before. These complex systems and networks have access to massive amounts of personal and business data, information, and content in ways

that are difficult for users to understand and control. In recent years we have witnessed a growing series of accidents and attacks on the Internet and on applications and databases. Through service attacks, viruses, phishing, spyware, and other malware, criminals disrupt service provisioning and steal personal or confidential business data for financial gain or other purposes. These disruptive attacks impact the e-market on an international scale.

Apart from external attacks, there are also development-related causes for software to fail. Some of these are:

- Change of the program source code
- Modification of its context by updating libraries or changing its configuration
- Modification of its test suite.

Any of these changes can cause differences in program behavior and, as a result, influence the user's level of trust associated with it. A popular belief here is that the remedy for losing the user's trust is *tests*.

Software testing is a way to find faults in software. It is a process of supplying a system under test with some values and making conclusions on the basis of its behavior. The most significant weakness of testing is that the functioning of the tested system can, in principle, only be verified for those input situations that were selected as test data. According to Dijkstra [4], testing can only show the existence but not the nonexistence of errors. Proof of correctness can only be produced by a complete test, that is, a test with all possible input values, input value sequences, and input value combinations under all practically possible constraints, which is theoretically possible but unfeasible in practice. In conclusion, tests are one of crucial components in building software or system trustworthiness, but alone are not enough.

The literature related to trust and software trustworthiness attributes and factors mentions several different approaches, with few of them presented briefly in the following. The National Science and Technology Council (NSTC) in the United States concentrates on high confidence systems and cyber-physical systems that are categorized into physical, biological, and engineered systems whose operations are integrated, monitored, and/or controlled by a computational core. Components are networked at every scale. Computing is "deeply embedded" into every physical component, possibly even into materials. The computational core is an embedded system, usually demanding real-time response, and is most often distributed. The behavior of a cyber-physical system is a fully integrated hybridization of computational (logical) and physical action [5]. In this category, the emphasis is on safety, stability, and performance attributes.

Avizienis et al. have defined dependability as the reliance that can justifiably be placed on the service that the system delivers [6]. Dependability has become an important aspect of computer systems since everyday life increasingly depends on software. Although there is a large body of research in dependability, architectural level reasoning about dependability is only just emerging as an important theme in software

engineering. This is due to the fact that dependability concerns are usually left until too late in the process of development. Additionally, the complexity of emerging applications and the trend of building trustworthy systems from existing untrustworthy components are urging dependability concerns to be considered at the architectural level. Lemos et al. have recognized fault prevention, fault removal, fault tolerance, and fault forecasting as important quality attributes for dependable systems [7].

A group of multinational technology and consulting firms such as HP, IBM, Intel, and Microsoft have formed the Trusted Computing Platform Alliance (TCPA) [8]. Their belief is that the totality (hardware, firmware, software) of the components is responsible for enforcing a security policy so that the system operates as expected. The group eventually grew to over 190 members and focused on "improving trust and security on computing platforms." Later, TCPA was replaced by the Trusted Computing Group (TCG) that concentrates on hardware and software security and use of cryptography in maintaining security.

Microsoft has developed an integrated process called trustworthy computing for improving the security of commercial software as it is being developed. Microsoft believes that there are three facets for building more secure software: repeatable process, engineer education, and metrics and accountability [9]. In Microsoft's experience, the benefits of providing more secure software (e.g., fewer patches, more satisfied customers) outweigh the costs.

The Trusted Computer National Evaluation Criteria (TCNEC) restricted trustworthiness based on security as the only attribute to consider [10]. Parnas et al. define software trustworthiness as level of appropriateness of using software engineering techniques to reduce failure rates, including techniques to enhance testing, reviews, and inspections [11]. In the study of Trustworthy Software Methodology (TSM) originally performed by the US National Security Agency, software trustworthiness was defined as "the degree of confidence that exists to meet a set of requirements."

The Data and Analysis Center for Software (DACS) has produced two reports, Software Assurance for Project Management [12] and Software Security Assurance [13]. In the former, the researchers emphasized that software trustworthiness factors and metrics are only restricted to the project management-related factors and metrics, while in the latter the focus is on security assurance for measuring software trustworthiness and they used a number of security-enhanced software methodologies such as Microsoft's Trustworthy Computing SDL.

There are also works discussed in the next sections that develop trustworthiness structures based entirely on the notions of quality and applying as its framework the ISO/IEC 9126 set of standards. As can be seen, the perception of trustworthiness and related concepts vary from one organization or researcher to another and all of them are in some sense partial.

Before going further into the details of the contemporary positioning of trustworthiness, it may be profitable to shortly repaint the actual IT market diversity. *OTS (off-the-shelf)* applications enjoy the biggest population of users. In this category we find, among others, software that supports individual tasks or set of tasks such as text editing, media processing and playing, individual accountancy or finance, some

data processing, and simple database applications. Using such software usually does not require specific IT knowledge but rather some self-learning about how to use it and eventually adapt it to our needs.

Commercial Modular Systems (CMS) are built from predeveloped modules or subsystems for a known user, having large, configurable, yet finished set of functionalities. In most cases, functionalities are linked together into "services." The user population is mostly corporate and as such requires intensive training before becoming fully operational and productive. The training may be divided into end-user type, which is service-oriented, and technical user training, which goes into the category of operation and maintenance. In both cases, the training is done by the supplier or his or her delegated representatives (certified companies). One of the best-known examples would be the ERP type of IT systems.

Individual On-Demand Systems (IOS) are developed to meet individual and unique requirements of the known user. The system may offer functionalities, services, or both. As in case of a CMS, the user population is mostly corporate and in consequence also requires intensive training. The training types are similar too, that is, they may be divided into end-user and technical-user training, but are done by the supplier only, as the supplier is usually is the only source of knowledge available to the user.

What do all these applications/systems have in common?

- The user owns them (or at least the license to use them)
- The user knows how build them
- The user knows whom to pursue (or sue) if something goes wrong
- The trustworthiness in this context is manageable, as the responsibility can be pinpointed to a known entity.

The last statement is a bit of wishful thinking, for actual legal and market ramifications make the fight of an individual consumer against an IT giant very difficult at best. However, the framework is there, which means that there is a known entity (a supplier) that can be reached and made accountable for its actions. Except we are not there yet. This aspect becomes even more disputable if the recent developments in IT domain are taken into consideration. For some years now there are two buzzwords that excite both suppliers and IT consumers, not necessarily for the same reasons: service-oriented architecture and cloud computing.

Service-Oriented Architecture (SOA) is essentially a collection of services. These services communicate with each other in order to establish the cooperation that can involve two or more services coordinating some activity in order to "do a job." The technology of Web services is the most likely connection technology of service-oriented architecture. The idea by itself is not necessarily very new, as its younger, less "fancy" predecessors such as DCOM or CORBA have been well known for a few decades now. The user's position in regard of trustworthiness is in such a case "undefined" at best, as there is no "named entity" that could eventually be linked to the service as a whole. In other words, there is no one to blame within a given SOA structure for eventual losses that the consumer has suffered.

Cloud computing is a general term for anything that involves delivering hosted services over the Internet. These services are broadly divided into three categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). *IaaS* is simply a virtual server instance with a unique IP address and blocks of storage on demand that can be used by the customer as it suits him or her. *PaaS* represents a set of software and product development tools hosted on the provider's infrastructure that allow the customer create his or her own application or even system. *SaaS* is a vendor-supplied hardware and software product infrastructure that interacts with the user through a front-end portal. The user's position in regard to trustworthiness in these three categories of services seems to be a bit more controllable, as there is a "vendor" or "supplier" who can theoretically be linked to the given service, but in fact the user may easily fall into a modern version of a classic "dot com" pattern. All these "vendors" and "suppliers" are no more than virtual entities, hidden somewhere in the "cloud" of the Internet, never seen, never talked to, practically out of any direct control.

So what could or would be a twenty-first century IT user's perception of trustworthiness in an era of *cloud computing* or *SOA*? It can be analyzed in a rather speculative way from the perspective of trustworthiness attributes discussed earlier, although the obtained results most probably would be too general for a particular IT consumer. Instead, it would be recommended that the user, before deciding whether a given application/service/system potentially exhibits required trustworthiness, analyze as precisely as possible the known trustworthiness attributes by answering the few related questions listed in the following:

- Quality. Of what? Service, system, SOA sub-services, PaaS?
- Reliability. (Same as the above.)
- Credibility. Whose credibility? Supplier's, vendor's, system's?
- Dependability. (Same as the above.)
- Completeness of required functions. In an era of SOA and cloud computing, are we still talking about "functionalities"?
- Proper quality–cost ratio (so the software was nor overpaid). "How easy will it be to be verified in my particular case?"
- Post-sale maintenance and service. "What and how should and can be done in my particular case?"
- Pre- and post-sale training. (Same as the above.)
- Documentation. Who, what, and how should deliver?
- Responsibility for the product. Who represents the "product"?

4.2 ROLE OF IT TRUSTWORTHINESS IN CONTEMPORARY SOCIETY

Contemporary society has gradually become profoundly dependent on information technology. We are so used to its presence that it has become almost transparent,

invisible, until something bad happens. Once it happens and once it is bad, the trust in our technological "servant" immediately diminishes. The existence of this lack of trust in information technology is known in the industry or we would not have backups, RAID solutions (redundant array of independent disks), redundant systems, and so on, however, the social objective of verifying the level of system or service trustworthiness goes far beyond mere technology. This objective is to assure a user in a proof-based way that he or she can trust the service (product, software) he or she wants or has to use. Obviously, if the developer puts a "you can trust it" stamp on his or her products no one will take it seriously, at least no one who lost data or hours of work trusting the "professionalism" of previously used software. Attribution of such a stamp should come from an independent, trusted, and professional entity having a good history of correct evaluations and should be a normal market practice.

The user could then, before buying a software product or service, verify the credibility of the supplier/producer, the quality of his or her services (meant here as, e.g., post-sale or customer support services) and seeing on the product a "you can trust it" stamp from the certification agency be relatively well assured that the risks of using it are reduced to an acceptable minimum.

This idea can be more precisely expressed through three practical domains of activities of a trustworthiness certification entity:

- Validation of IT solutions. This activity falls into the category "process for the evaluator" discussed in Section 4.3.2.2, and has as an objective to validate an IT solution on behalf of the future user. The "you can trust it" stamp, together with the exhaustive trustworthiness evaluation report, could serve, among others, as the crucial attachment to the main contract.
- Verification of IT suppliers. In this case, the certification could be applied to the supplier rather than to the product. In fact, not every "product" has to be software-based even if it exists in IT domain. The whole market of IT consultancy is based on services as products and the certified service supplier would be probably more sought after than an uncertified one.
- Certification of IT products and services. The whole market of OTS products and Internet-based (SOA, cloud computing) services would be the addressee of this part of a trustworthiness evaluation effort. The user then could choose between a box with a stamp that says "Trustworthiness Level y/10. Certified by National Trustworthiness Certification Agency. Valid until 20xx" and a box with no such stamp. The same could be applied to services, so the user would not be afraid that his or her sensitive data will begin its own quest in the labyrinth of the World Wide Web.

4.3 MONITORING, MEASUREMENT, AND CERTIFICATION OF IT TRUSTWORTHINESS

If the role of trustworthiness is to be effectively realized, at the base of this process there has to be an intellectually defendable and practically applicable understanding

of what exactly constitutes trustworthiness in contemporary IT domain. Being a rather new technological concept, trustworthiness is under scientific scrutiny in several different ways, beginning with a quality-only static approach, through semi-dynamic, architecture-based solutions to dynamic, behavior-based measurement and evaluation process. These different perspectives are discussed further in the following sections.

4.3.1 Classic Approaches

The numbers quoted in previous section are widely known and acknowledged within the IT community. Several scientific efforts and research projects have been undertaken to tackle the problem of building the proper mechanisms (models, theories, methodologies) to allow for efficient estimation of trustworthiness of, generally speaking, information technology products. Same of them, such as the method proposed in Reference 14 (Fig. 4.1), the solution developed in Beijing University project "TRUSTIE" (Fig. 4.2) [15] or the methodology HATS (Highly Adaptable and Trustworthy Software Using Formal Models) [16] developed by Swedish Chalmers University of Technology (Fig. 4.3), have taken as their basis software quality measurement and evaluation approach represented in ISO/IEC 9126. As can be noticed, both models are parametrical, attribute-oriented, and heavily based on the ISO/IEC 9126 quality model.

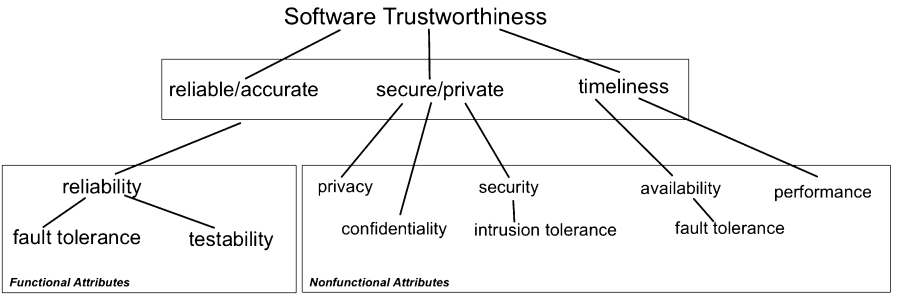

Figure 4.1 Software trustworthiness model (adapted from [14]).

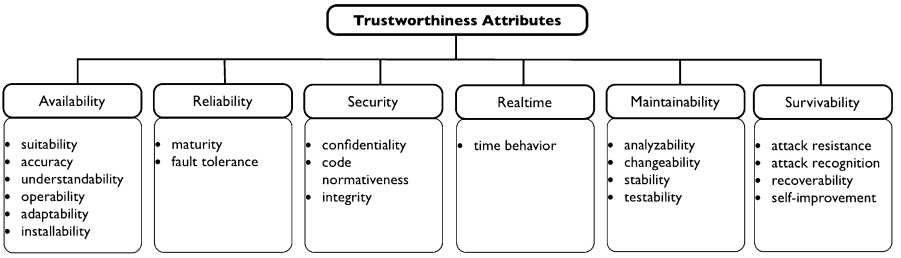

Figure 4.2 Methodology "TRUSTIE" (adapted from [15]).

Figure 4.3 HATS (adapted from [16]).

A relatively recent and rather innovative development is the dynamic, architecture-based solution, having two cooperating "engines": the trust engine and the auction engine [17]. The trust engine is "responsible for trust negotiation among services in order to establish the services that will be exchanged between the participating modules and set up a negotiated trust level for service access." The second "engine," the auction engine, is responsible for a computing trust level that the service provider has to prove. Once the trust level is bilaterally established and agreed upon, the whole service architecture may begin the realization of required tasks.

4.3.2 Dynamic Approaches: Behavioristic Method of Software, System, and Service Trustworthiness Certification

The notion of software trustworthiness evaluation in the literature is inherently subjective. It depends on how the software is used and in what context it is used. Moreover, different users evaluate a software system according to different criteria, point of view, and background. Therefore, when assessing software trustworthiness, it may be counterproductive to look for a general set of characteristics and parameters; instead, there is need to define a model that is tailored to the functional and quality requirements that the software has to fulfill. Such an approach is introduced in a behavioristic model for verifying software trustworthiness (BeMSET) based on scenarios of interactions between the software and its users and environment. These interactions consist of simple scenarios of examples or counterexamples of desired behavior. The approach supports incremental changes in requirements/scenarios.

4.3.2.1 General Concepts of the Method

The general concept behind the behavioristic approach to IT system or service trustworthiness is constructed on four basic points [18]:

- The final user does not want or have to be IT literate. He or she simply wants to be a *user.*

- The trust for a system or service does not have to be total. It is enough if it is *contextual*, that is, within the space of user's interests or needs.
- The behavior of a service/system in the given usage context can be represented in a formal way, applying one of the existing formal notation methods. In case of BeMSET, this notation is a combination of unified modeling language (UML) scenarios and finite state machine (FSM) computation model.
- The observed trends in information technology indicate that the distance between a consumer (user) of IT services and executive mechanisms necessary for these services grows.

The behavioral trustworthiness in order to fulfill its objectives as stated in Section 4.2 has to be supported by a process allowing for a transition from technical or descriptive expression of evaluation requirements (depending on the type and IT proficiency of the requester) to the documented results of evaluation and eventual issuance of a certificate.

Figure 4.4 presents such a framework process, a general model of BeMSET where the whole procedure is divided into six consecutive steps:

- *Identification of required scenarios/objects.* In this step the trustworthiness context and related usage scenarios are identified (example: context: online banking; scenario: transfer of funds from account A to account B).
- *Transformation of scenarios to their FSM representations.* In this step, the scenarios from Step 1 are transformed into FSM table(s) of states, transitions, and stimuli. This formal form of the service under evaluation is stored in a database for further processing and as part of the knowledge base for a given category of services.
- *Execution of scenarios by a real service or system.* In this step the scenarios from Step 1 are executed by a real system or service under evaluation. The states and transitions exhibited by it are recorded in FSM table(s) and stored in the database.
- *Comparison of transformed and recorded FSM tables.* This step applies mathematical methods of comparison of the two obtained FSM representations of the service or system under evaluation. As the result, the differences (deltas) in paths, states, and transitions between the real and required behavior of a system or service are found and documented.
- *Results evaluation.* This step has as an objective to qualitatively and quantitatively analyze the comparison results from the previous step. The applied analysis method (of the choice of the evaluator) should allow for classifying the discrepancies into at least three basic categories: fail, conditional pass, pass.
- *Issuance of the certificate.* The certificate must be based on a sound evaluation from the previous step and can be full, conditional, refused, one-time, renewable, periodic, or with no time limit. In case of a failed reevaluation,

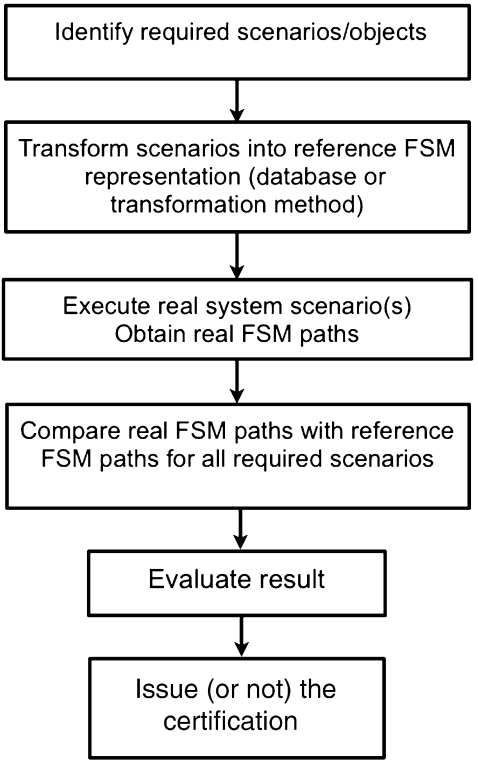

Figure 4.4 General model of BeMSET.

the certificate may also be revoked. All these cases should follow the rules established in and for a given certification authority.

In reality the BeMSET process is considerably more complex. In its form proposed in Reference 19, the process is built in three phases, each having several stages or steps (Fig. 4.5).

Phase 1: Identifying attributes of software or service trustworthiness. This phase is dedicated to identifying and characterizing the attributes and factors from user scenarios and/or UML scenarios. It is important to note that not only are the quality attributes that are important for the overall trustworthiness taken into account, but also the domain of software systems or services as well (see Section 1.2.3). The context of use is taken into consideration as the third perspective. It can be found out by answering the following questions:

- Who will use the software (users and roles)?
- What will be done with the software (functional requirements)?
- Where the software will be used (environment)?
- How will the software be used (quality requirements)?

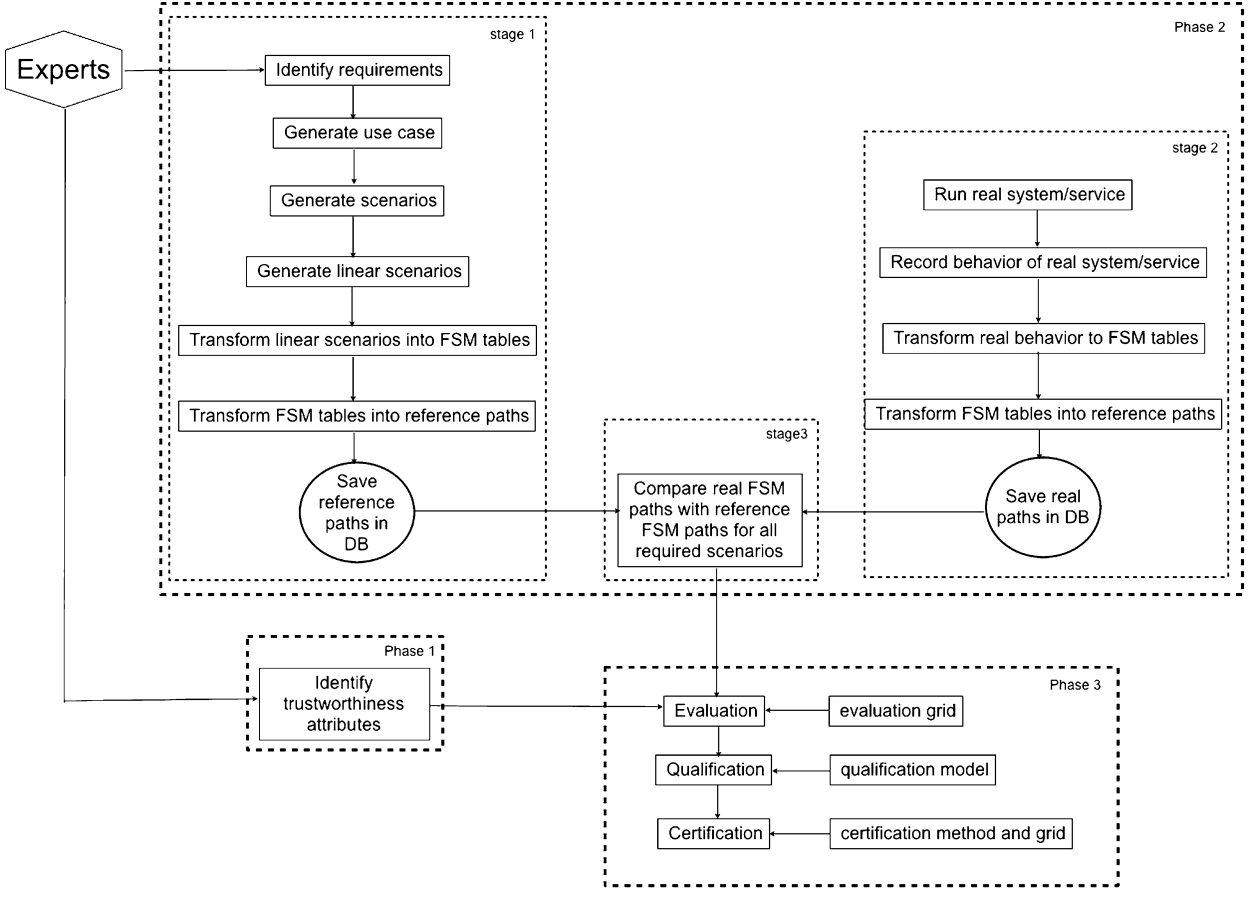

Figure 4.5 BeMSET trustworthiness evaluation process.

As the result, a generic list of all possible factors and attributes will be created and presented to the user. Then the user will choose those attributes and factors that relate and matter to the software system or service under investigation and will identify and define the properties of his or her interest. Therefore, the result of this phase is the definition of a generic model for factors and attributes of functional and/or quality requirements of a software system or services.

Phase 2: Developing a formal behavioristic model. In this phase, the selection (or development, if need be) of a formal specification language and the choice of a suitable formal notation (in case of BeMSET, FSM) for developing a formal behavioristic model are made. There are three stages and each stage has its own steps, as follows:

- Stage 1: Transforming reference requirements to FSM representation
  - identification of reference requirements
  - extraction of reference functionalities and quality attributes
  - generation of reference use cases
  - extraction of reference scenarios
  - transformation of any hierarchical (tree format) scenarios to nonhierarchical (linear format) scenarios
  - generation of reference FSM tables from each nonhierarchical (linear format) scenario
  - execution of reference FSM tables to obtain the reference path representation
  - saving reference path representation into reference database.
- Stage 2: Recording real software system or service behavior
  - execution of real software system or services
  - recording of the real behavior into real FSM tables
  - execution of real FSM tables and development of the real path representation
  - saving real path representation into the recording database.
- Stage 3: Comparing real behavior with reference behavior
  - reading real path representation from the recording database
  - reading reference path representation from the reference database
  - comparing reference path representation with real path representation
  - analyzing and interpreting the observed behavior and its discrepancy from reference behavior for the software system or services under investigation.

This mechanism of the model enables recording of paths of all the states, transitions, events, and actions in the sequence in which they are performed (real) or should have been performed (reference). These paths are saved in a database for future reference for both real and expected (reference) behavior of software system and services.

Phase 3: Evaluation method. In this phase, the analysis of the results of comparison between real FSM paths and reference FSM paths for all requested scenario(s)

is executed. The evaluation of the results (whether the software system or services passed or not) is used as the basis for awarding (or not) the certification. The development or choice of the appropriate evaluation method of trustworthiness for software system or service under investigation is the most important part of this phase. For evaluation, all the information and results from Phase 1 and Phase 2 are taken into account. Also, the evaluation grid and qualification model are used to evaluate the behavior of software system or services. One other dimension of this phase is related to the interpretation of discrepancies between real behavior and reference behavior of software systems or services under investigation. The specific grids relate, among the others, to the category of the system or service under evaluation that should be used in certification process.

An additional remark about this process: let's keep in mind that the attributes and factors for trustworthiness of a system or software should be identified from user scenarios and/or UML scenarios. These attributes and factors contribute to analyzing and evaluating trustworthiness of the software system or service under investigation in a way shown in Fig. 4.6.

A scenario can be represented as a path consisting of state nodes and transition edges through the statechart diagram. Statechart diagrams have different types of actions that are performed within the state, such as [20]:

- Entry actions
- Input actions triggered by input conditions that do not cause state changes
- Exit actions

or cause state transitions, such as:

- Transition actions, which are a type of input action but are bound to a transition.

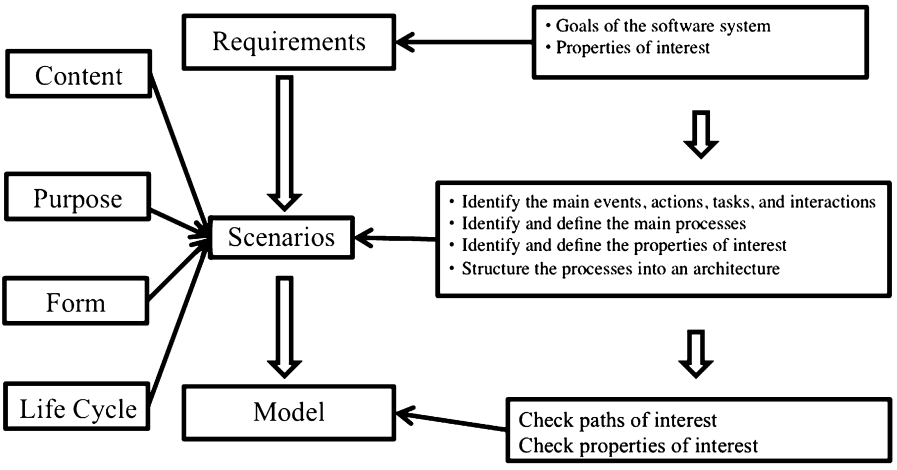

Figure 4.6 Views required to capture relevant aspects of scenarios.

A new scenario is written by adding states and state transitions to the state machine incrementally, in such a way that the requested scenario becomes executable as a path through the same statechart diagram. It should be noted that all those states and state transitions belong to the same use case. The following additional information is needed to generate a scenario and a statechart [21]:

- Any actions or events causing a trigger or state transitions such as clock events, pressing a button, or selecting a menu item
- Any action or events related to entering or choosing a data input or a change to data, such as completing a form or selecting a data item from a dropdown menu or trigger of timer or guard condition
- The object status of the system at the beginning state of scenario execution.

4.3.2.2 Service Trustworthiness Analysis and Certification

Let's recall that a basic definition of the adjective "trustworthy" uses the notion of "worthy of confidence" and "dependable" [22]. The term "trustworthiness" in the context of IT comes as the extrapolation of this definition and can be explained as "the assurance that a system deserves to be trusted, that it will perform as expected despite environmental disruptions, human and operator error, hostile attacks, and design and implementation errors" [23]. In real life, such an assurance is most often partial (or contextual) and needs to be evaluated using a stable and correct process and tangible data. The series of standards ISO 14598 provide the guidance and requirements for the evaluation process of software product quality from three different perspectives: development, acquisition, and an independent evaluation. It is also broadly known that quality makes an inherent component of trust, but equating it alone to trustworthiness would be considered a risky limitation of a trustworthiness concept. However, a practical question would be: Could the ISO/IEC 14598 evaluation processes be applied to evaluate trustworthiness, or it is necessary to develop something new? In light of the development of the ISO/IEC 25040, ISO/IEC 25041, and ISO/IEC 25042 standards dedicated to quality evaluation, the same question would apply to them. This possibility was analyzed using an example where an IT system was represented applying the behavioristic model of trustworthiness, BeMSET.

4.3.2.2.1 IT System Example

The IT system used in the analysis is a system for remote monitoring of diabetic patients in serious condition. In simple terms, a diabetic patient must connect each day before 10 a.m. to the system via its web portal, enter his or her blood sugar level, blood pressure, and weight, and submit these data for evaluation by the system. The basic functional requirements are then:

- The system shall collect patient's submitted data via its portal
- The system shall save the data in the medical history of the patient
- The system shall validate the collected data

- If the data represent a normal situation, the system shall wait for the next scheduled submission
- Otherwise, the system shall send an alert message to the patient's doctor
- If the patient does not submit the data before 10 a.m., an alert message shall be sent to the patient's doctor
- The alert message shall be sent in two minutes after the discovery of the patient's abnormal condition
- The alert message shall be sent by SMS (Short Message System), via telephone and email.

These requirements can be represented then as a state table or the series of paths built of required states and related transitions, such as when the system goes from the "waiting" state to the "send alert" state if there is no valid connection from a patient at or before 10 a.m.

From the BeMSET perspective, the important elements are:

- Contextual elements: the existence and execution of all required transition paths and their related states built from system behavior scenarios
- Quality elements: the existence of all required quality attributes of both states and transitions.

In the example, the contextual behavior is verified by examining all required transitions among states, while quality behavior is verified by measuring and evaluating related quality attributes, for which ISO/IEC 14598 (and its ISO/IEC 2504x) offers ready-to-use support. It seems justified, then, to verify whether this support could be extended to the evaluation of trustworthiness as the whole process in the context of BeMSET.

4.3.2.2.2 ISO/IEC 14598-3 Process for Developers

ISO/IEC 14589-3 [24] "provides requirements and recommendations for the practical implementation of software product evaluation when the evaluation is conducted in parallel with the development and carried out by the developer." From the perspective of BeMSET, the process (Fig. 4.7) corresponds rather to auto-evaluation by the service supplier prior to submitting the service to a certification organization.

The phases of the evaluation process for developers are defined as follows (from ISO 14598-3):

- "analysis of evaluation requirements which consists of identifying the quality requirements according to an agreed quality model"
- "specification of the evaluation which consists of determining the external measures and target measurement values (criteria for evaluation)"
- "design of the evaluation which consists of determining the internal measures and target measurement values (criteria for evaluation)"
- "execution of the evaluation which consists of collecting internal measurement values during development and comparing with target values (evaluation

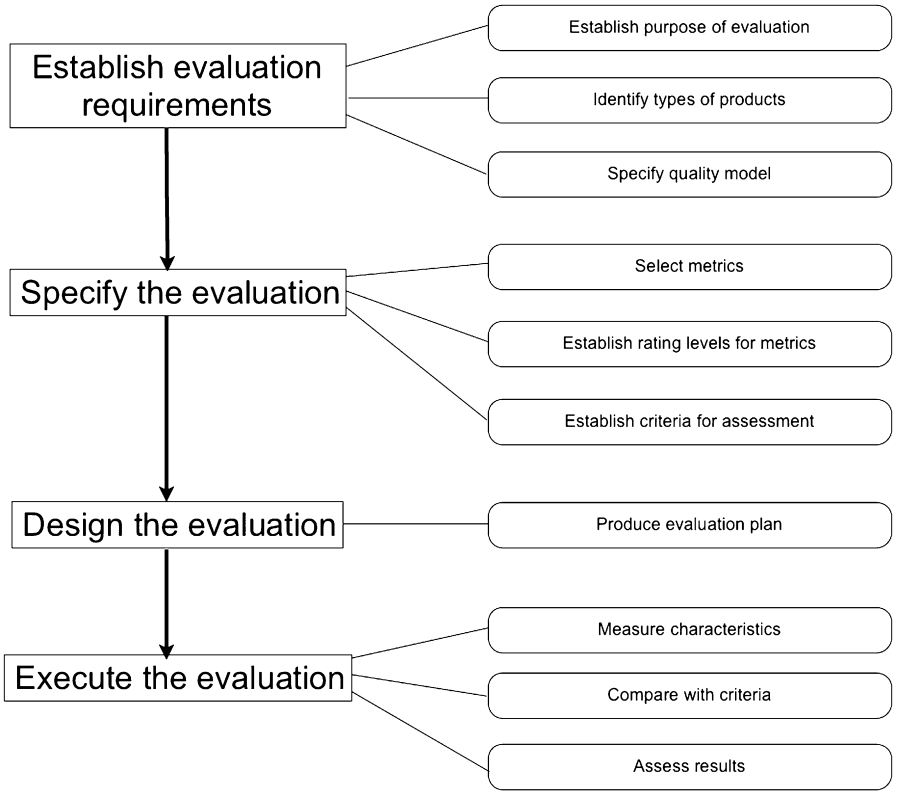

Figure 4.7 ISO/IEC 14598-1 evaluation process (adapted from [25]).

during development). Internal attribute values (quality indicators) are used to estimate end product quality"

- "conclusion of the evaluation which consists of collecting external measurement values when they become available and comparing with target values (evaluation of product quality)."

The first essential difference between evaluation of quality alone and evaluation of trustworthiness is the service (or software) life cycle phase in which the evaluation takes place.

Applying BeMSET requires the existence of a fully operational service, which means being far past internal quality definition and measurement stage. However, external quality may and should be perceived as "behavioral" as it applies to a running software or system that exhibits actions and reactions (services). Using this approach as a filtering mechanism to the process described in ISO/IEC 14598-3 automatically removes from it all activities related to internal quality, thus leaving the evaluation design and evaluation execution phases temporarily void.

The second essential difference between evaluation of quality and trustworthiness is the role of a verifier. In the ISO/IEC 14598 the developer *is* the verifier, while in auto-evaluation of trustworthiness using BeMSET, the developer becomes principally the user who evaluates both contextual and quality elements of the service, system, or software. Applying this observation to ISO/IEC 14598-3 indicates that the notion of "external quality" is too narrow and should be adjusted in all phases and activities that relate to it (analysis, specification, and conclusion of evaluation).

The third essential difference between evaluation of quality and trustworthiness is the notion of the "model." From the perspective of BeMSET, the quality model becomes a component of a larger model that also includes all the scenarios in which the system has to be trustworthy. The quality model should be used as an agreed-upon reference for structuring the quality requirements, and as such it can be a proprietary solution as well as a standardized one, like the model from ISO/IEC 9126 or ISO/IEC 25010. Additionally, keeping in mind that the ultimate objective of the analyzed process (of auto-evaluation) is the certification by an independent accrediting organization, the scenarios' parts of the overall model should come from the certification organization and be used as reference for granting the certification.

Despite being substantial, the aforementioned differences do not rule out the possibility of reusing the ISO/IEC 14598 (and ISO/IEC 2504x) process for developers in auto-evaluation of trustworthiness. After addressing them, the adapted process could consist the following five phases:

- Analysis of evaluation requirements, which consists of identifying all required transition paths, their related states, and corresponding quality requirements according to an agreed-upon BeMSET model
- Specification of the evaluation, which consists of determining the reference transition paths, required states, related quality measures, and target measurement values
- Design of the evaluation, which consists of determining execution sequences of evaluated paths, their order, recording and measurement techniques, and target quality measurement values
- Execution of the evaluation, which consists of executing and recording all specified paths and states and measuring quality values
- Conclusion of the evaluation, which consists of comparing recorded transition paths and states with their references, comparing quality measures with target values, and preparing the evaluation report.

ISO/IEC 14598-4 [26] contains "requirements, recommendations and guidelines for the measurement, assessment and evaluation of software product quality during acquisition of off-the-shelf software products, custom software products, or modifications to existing software products, in conjunction with the requirements of ISO/IEC 12207." From the perspective of BeMSET, the process (Fig. 4.8) corresponds to evaluation by the future user prior to the purchase of the service.

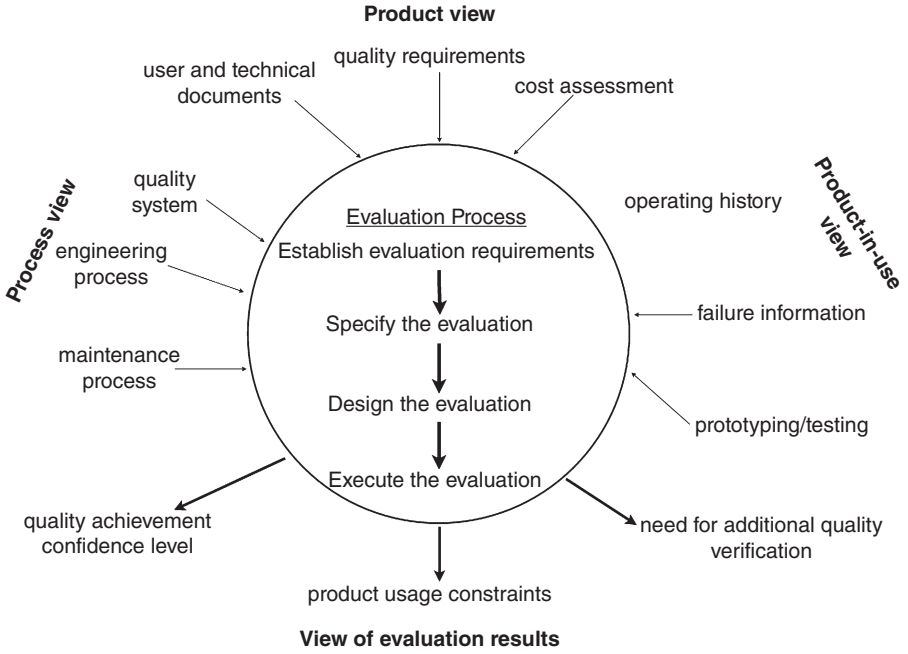

Figure 4.8 ISO/IEC 14598-4 evaluation process for acquirers (adapted from [26]).

The phases of the evaluation process for acquirers are defined as follows (direct quotation from ISO 14598-4):

- Establish evaluation requirements, define objectives and scope clearly. Identify:
  - what questions must be answered?
  - how rigorously should the evaluation be performed?
  - how complete are the inputs to the evaluation?
  - is additional evaluation planned for other phases of the project?
  - has any evaluation been done in previous phases of the project or by others?
  - what acquisition process is to be followed and how are evaluation input requirements communicated to the vendor?
- Specification of the evaluation. Once the requirements of the evaluation are understood:
  - select metrics that correlate to the characteristics of the software product and establish rating levels
  - select the most effective set of evaluation methods
  - establish procedures for summarizing the results of the evaluation of different quality characteristics and other aspects that contribute to the assessment of quality of a software product in a particular environment.

- Design of the evaluation. Prepare an evaluation plan describing the evaluation methods and the schedule of the evaluator actions. Identify the tie-points between evaluation activities and acquisition activities
- Execution of the evaluation. Conduct the selected evaluation activities, and analyze and record the results to determine the suitability of the software product(s). Analyze the impact of identified deficiencies and options to regulate the use of the existing software product. Draw conclusions with respect to the acceptability of the product and the ultimate decision to buy or not to buy.

The application of the analysis method used previously renders the following observations:

- In the phase of establishing evaluation requirements, the references to "phases of the project" are inapplicable and should be removed or modified to better adhere to the nature of trustworthiness evaluation
- In the phase of specification of the evaluation, the references to "metrics that correlate to the characteristics of the software product" are inapplicable and should be modified to better adhere to the nature of trustworthiness evaluation.

As the remaining phases of the ISO 14598 process for acquirers are applicable as they are, the adapted and simplified process could consist the following four phases:

- Establish evaluation requirements, define objectives and scope. Identify:
  - what questions must be answered from the point of view of evaluated service behavior and its quality?
  - how rigorously should the evaluation be performed?
  - how complete are the inputs to the evaluation?
  - has any evaluation been done prior to this acquisition or by others?
  - what acquisition process is to be followed and how are evaluation input requirements communicated to the vendor?
- Specification of the evaluation. Once the requirements of the evaluation are understood:
  - select reference transition paths, states and related quality measures and establish rating levels
  - select the most effective set of evaluation methods
  - establish procedures for summarizing the results of the evaluation
- Design of the evaluation. Prepare an evaluation plan describing the evaluation methods and the schedule of the evaluator actions. Identify the tie-points between evaluation activities and acquisition activities
- Execution of the evaluation. Conduct the selected evaluation activities, and analyze and record the results to determine the suitability of the software product(s). Analyze the impact of identified deficiencies and options to regulate the use of the existing service. Draw conclusions with respect to the acceptability of the service and the ultimate decision to buy or not to buy.

ISO/IEC 14598-5 [27] provides "requirements and recommendations for the practical implementation of software product evaluation when several parties need to understand, accept and trust evaluation results." From the perspective of BeMSET the process (Fig. 4.9) corresponds to evaluation of the service by the certification organization.

The phases of the process for evaluators are defined as follows (due to its original volume, the presented quotation from ISO 14598-5 has been shortened):

- Analysis of evaluation requirements:
  - expressing the extent of the coverage of the evaluation by the requester
  - explaining the extent of confidence and stringency of evaluation by the evaluator
  - agreeing on the evaluation requirements
- Specification of the evaluation based on the evaluation requirements and on the description of the product provided by the requester
  - specifying the measurements to be performed on the product and its components
  - verifying the specification produced with regards to the evaluation requirements
- Design of the evaluation which produces an evaluation plan on the basis of the evaluation
  - documenting evaluation methods and producing a draft plan
  - optimizing the evaluation plan
  - scheduling evaluation actions with regard to available resources
- Execution of the evaluation plan which consists of inspecting, modeling, measuring and testing the products and its components according to the evaluation plan
  - manage the product components provided by the requester
  - manage the data produced by the evaluation actions (including report and records
  - manage the tools to be used to perform the evaluation actions
  - manage evaluation actions performed outside the evaluator's premises
  - manage the requirements implied by the use of specific evaluation techniques.
- Conclusion of the evaluation, which consists of the delivery of the evaluation report and the disposal by the evaluator of the product evaluated as well as its components when they have been transmitted independently.

The application of the analysis method used previously indicates that this process can be applied to evaluation of trustworthiness after three minor adjustments:

- Replacement of the term " product" by the term "service"
- Replacement of the term " product and its components" by the term "service"
- Removing the term "modeling" from execution of the evaluation plan phase.

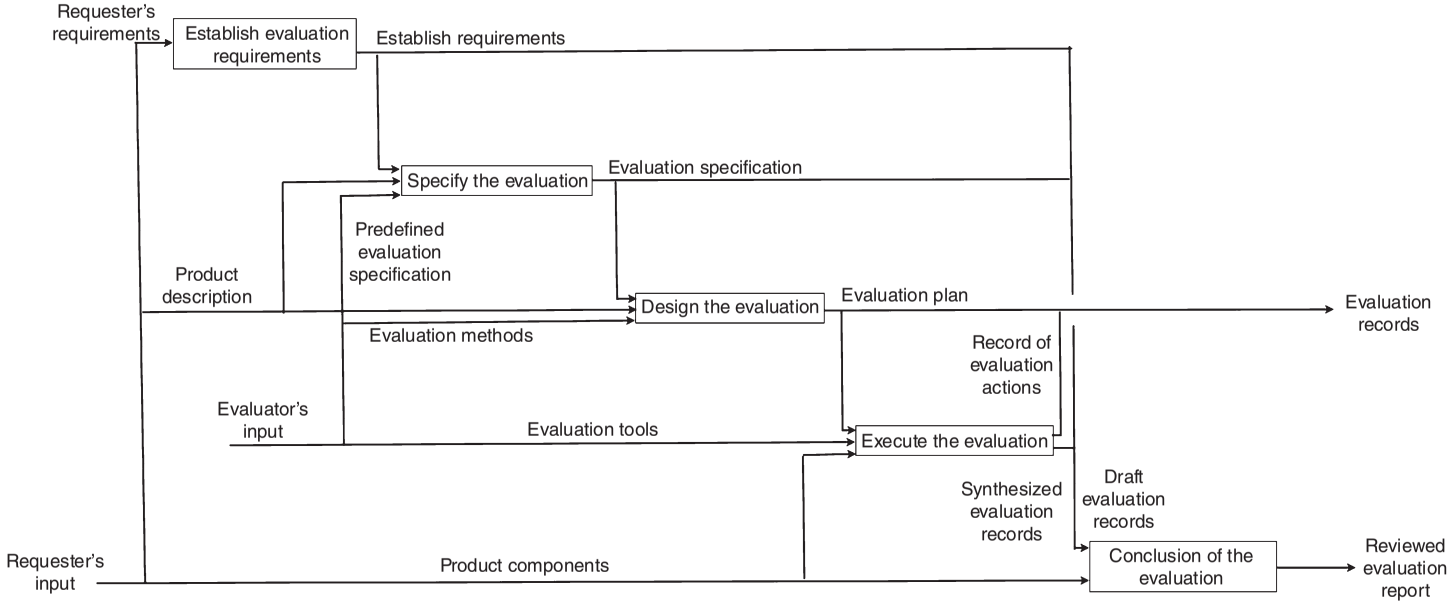

Figure 4.9 ISO/IEC 14598-5 evaluation process for evaluators (adapted from [27]).

In conclusion, the ISO/IEC 14598 (and ISO/IEC 2504x) evaluation offers a considerable support to the BeMSET-based evaluation of trustworthiness of software systems and services; however, not without sometimes substantial modifications.

The most essential identified modifications apply to the process for the developer, where the activities relating to internal quality and its measures should be removed; the evaluation requirements, specification, design, and execution should be focused on both contextual elements and quality elements rather than on quality alone; the model should be contextual quality rather than quality alone; and the role of the developer as verifier should be enhanced to cover the end-user perspective.

The modifications required for the process for the acquirers are relatively small and focus mostly on references to life cycle phases where the evaluation might take place and on direct references to evaluation of quality only.

The modifications required for the process for evaluators are reduced to replacing the term "product" by a more specific term, "service."

REFERENCES

1. Bishop M. *Computer Security: Art and Science*. Pearson Education, 2003.
2. "Trustworthiness," http://en.wikipedia.org/wiki/Trustworthiness.
3. Minnesota State Archives, http://www.mnhs.org/preserve/records/index.htm.
4. Dijkstra EW, Dahl OJ, Hoare CAR. *Structured Programming*. London: Academic Press, 1972.
5. Gill H. "High Confidence Software and Systems: Cyber-Physical Systems Progress Report: Semantics Perspective." In *Second Workshop on Event-based Semantics/IEEE RTAS; April 22–24, 2008*. Missouri: IEEE, 2008.
6. Avizienis A, Laprie JC, Randell, B. "Fundamental Concepts of Dependability." Technical Report 739; 2001. Department of Computing Science, University of Newcastle upon Tyne. Available at http://www.cs.cmu.edu/∼garlan/17811/Readings/avizienis01_fund_concp_depend.pdf. Accessed May 16, 2013.
7. De Lemos R, Gacek C, Romanovsky A. "ICSE 2002 Workshop on Software Architectures for Dependable Systems (Workshop Summary)." *ACM Software Engineering Notes* 2003; 28(5).
8. Safford D. "The Need for TCPA." IBM Research. http://www.ibm.com. Accessed April 30, 2011.
9. Mundie C. "Trustworthy Computing: Microsoft White Paper." Available at http://download.microsoft.com/download/a/f/2/af22fd56-7f19-47aa-8167-4b1d73cd3c57/twc_mundie.doc. Accessed May 15, 2013.
10. DOD 5200.28 STD Trusted Computer System Evaluation Criteria. Department of Defense, National Computer Security Center, 1985.
11. Parnas, D. "Evaluation of Safety-Critical Software." *Communications of the ACM* 1990; 33(6):635–648.
12. Fedchak E, McGibbon T, Vienneau R. "Software Project Management for Software Assurance: A DACS State-of-the-Art-Report." Final report September 30, 2007. Air Force Research Lab, Griffiss. Report number 347617. Contract Number SP0700-98-D-4000. Available from https://buildsecurityin.us-cert.gov/bsi/dhs/906-BSI.html. Accessed May 15, 2013.

13. Goertzel KM, Winograd T, McKinley HL, Oh L, Colon M, McGibbon T, Fedchak E, Vienneau R. "Software Security Assurance: A State-of-the-Art-Report." DACS and IATAC joint report, July 31, 2007. Available at http://iac.dtic.mil/csiac/download/security.pdf. Accessed May 15, 2013.
14. Voas J, Agresti WW. "Software Quality from a Behavioral Perspective." *IT Professional* 2004; 6(4):46–50.
15. Yin G. "Practices and Thinking of the Models in Trustie Environment." International ABC Conference March 15, 2010; Tokyo, Japan. Available from www.trustie.net. Accessed May 15, 2013.
16. Hahnle R. "HATS: Highly Adaptable and Trustworthy Software Using Formal Models." International Symposium on Formal Methods for Components and Objects; October 23, 2008, Sofia-Antipolis. Available at http://www-sop.inria.fr/oasis/FMCO/2008/slides/FMCO2008-HATS-ReinerHahnle.pdf.
17. Phoomvuthisarn S. "An Architecture Approach to Dependable Trust-Based Service Systems." In *6th International Conference on Service Oriented Computing; December 1–5, 2008*. Sydney, Australia: ICSOC, 2008.
18. Suryn W. Yuan Y. "Trustworthiness of Software, the Challenge of the 21st Century." In *The International Standard Conference on Software Trustworthy Testing; December 28, 2009–January 2, 2010*. Beijing, China: BUPT, 2009.
19. Nami M. Suryn W. "From Requirements to Software Trustworthiness Using Scenarios and Finite State Machine." In *38th IECON conference; October 25–28, 2012*. Montreal, Canada: IEEE, 2012.
20. Wagne F, Schmuki R, Wagner T, Wolstenholme P. "*Modeling Software with Finite State Machines: A Practical Approach*." Boca Raton, FL: CRC Press, Taylor & Francis Group, 2006.
21. Behrens H. "Requirements Analysis and Prototyping Using Scenarios and Statecharts." ICSE 2002 Workshop; May 19–25, 2002, Orlando, Florida.
22. Merriam-Webster Dictionary, 2010.
23. "Mobius Glossary." Available at http://www-sop.inria.fr/marelle/Mobius/mobius.inria.fr/twiki/bin/view/Mobius/Glossary.html#T.
24. ISO/IEC 14598-3 Software Engineering—Product Evaluation—Part 3: Process for Developers. Geneva, Switzerland: International Organization for Standardization, 2000.
25. ISO/IEC 14598-1 Information Technology—Software Product Evaluation—Part 1: General Overview. Geneva, Switzerland: International Organization for Standardization, 1999.
26. ISO/IEC 14598-4 Software Engineering—Product Evaluation—Part 4: Process for Acquirers. Geneva, Switzerland: International Organization for Standardization, 1999.
27. ISO/IEC 14598-5 Information Technology—Software Product Evaluation—Part 5: Process for Evaluators. Geneva, Switzerland: International Organization for Standardization, 1998.