# Chapter 3

# System and Software Quality Engineering: Some Application Contexts

*This chapter discusses software quality engineering in the application context of two categories of software systems: the information systems subcategory of information management, and embedded systems. Such a choice is based on their presence and/or the importance they bear in everyday social life. Information management systems, no matter their size and complexity, are present in almost every domain of social life. They can be found in libraries, hospitals, big international corporations, small enterprises, governments, or military institutions. Embedded systems are present in the life of almost every individual on the planet. They reside in wristwatches, microwave ovens, TVs, chip credit cards, cars, and planes, but also in missile guidance systems, nuclear reactor controllers, and nuclear missiles.*

*And a practical remark: even if the choice is limited to only two categories, the approach used in both cases is valid for any other category of software or system.*

## 3.1 SOFTWARE QUALITY ENGINEERING IN THE INFORMATION MANAGEMENT SYSTEMS ENVIRONMENT

Information management systems are systems that help the users better exploit and understand the information remaining in their access. In this book, the word "access" is understood as *legal* access. As the concept, information management systems were first developed by IMB in 1968 and consisted of two layers, database system and transaction system (the interested reader can follow this story on the IBM Website). In fact, this basic structure still makes up the backbone of most contemporary information management systems. This type of system can perform multiple tasks such as document management, knowledge management, customer relationship

*Software Quality Engineering: A Practitioner's Approach*, First Edition. Witold Suryn.
© 2014 the Institute of Electrical and Electronics Engineers, Inc.
Published 2014 by John Wiley & Sons, Inc.

management, and enhanced collaboration throughout the organization. Usually these systems also provide some deep administration and integration capabilities, relevant search functionalities, as well as solid security features that protect the information they contain.

The information management systems represent what employees use to execute their daily tasks. Because of the amount of crucial business data they may contain, the *security* attribute seems very important. Businesses simply cannot allow these data to fall into the wrong hands. Further down that line, the data processed by these systems may vary from memos to employees to sensitive financial or production information, where, in the latter case, the *accuracy* is a feature that conditions the results of related operations. Finally, since not every employee is an IT-savvy user, it seems recommendable that these systems be user-friendly, which means *operable* and customizable.

*Security*, as defined in ISO 25010 [1], means the "degree to which a product or system protects information and data so that persons or other products or systems have the degree of data access appropriate to their types and levels of authorization." This quality characteristic has five subcharacteristics (the definitions come from ISO/IEC 25010):

- Confidentiality: degree to which a product or system ensures that data are accessible only to those authorized to have access
- Integrity: degree to which a system, product, or component prevents unauthorized access to, or modification of, computer programs or data
- Nonrepudiation: degree to which actions or events can be proven to have taken place, so that the events or actions cannot be repudiated later
- Accountability: degree to which the actions of an entity can be traced uniquely to the entity
- Authenticity: degree to which the identity of a subject or resource can be proved to be the one claimed.

What might happen if these quality attributes were not engineered into the system or software, or were not effective enough?

The absent *confidentiality* may lead to breaking the informational, social, and operational structure of an organization. Letting *anybody* access *anything* could disclose a company's confident data (including trade secrets), personal files of employees, plans, or strategies, causing a catastrophe that would paralyze the organization functioning for good and for a long time.

The *integrity*, as understood by ISO/IEC 25010, leads to connotations of industrial espionage and sabotage. In fact, missing *integrity* would allow someone properly prepared to access the organization's information management system's data and play with it according to his or her goals. The next day it might happen that the competition knows how to build the company's secret new product, but the company do not know how to do it, because all the plans and designs were deleted from their system. Actually, the latter is a bit overdramatized, for the companies that have important secrets keep them in several secured places.

*Nonrepudiation* and *accountability* refer to "I did not do it" types of situations. In information management it is important to know with a required level of certainty who sent the information and, if the consequences of this action are in any sense "special," to be able to trace it back (with a convincing proof) to the entity responsible for it and make him or her accountable. The value of these attributes is true both in internal and external exchange of information, but is this importance critical? The false information sent by a fake entity, if convincingly wrapped, may invoke harming actions, even chaos, but still it will not be similar to what may be created by missing confidentiality or integrity.

The *authenticity* of the data circulating in an organization's information management system is important principally from the decision making point of view. In simple terms, making decisions based on unverified information may, but does not have to, lead to incorrect decisions. Depending on the importance of these decisions the consequences may be marginal or catastrophic, as when stock exchange operations are based on rumors.

*Accuracy*, replaced in ISO/IEC 25010 by *functional correctness* (a subcharacteristic of the *functional suitability* characteristic of dynamic/external quality) is defined as the "degree to which a product or system provides the correct results with the needed degree of precision." Those readers who sometimes watch the news from the stock exchange have surely noticed that the exchange ratios between currencies sometimes go to the precision of one hundredth of a cent. The reason for such an exceptional precision is the quantity of money that flows through the stock exchange every day. If a transaction of \$100 was supposed to be done with 0.01% (one hundredth of a cent) precision, but due to low accuracy attribute was performed with 1% (one cent) precision, the resulting sum instead of being \$100 ± 1¢ would be \$100 ± \$1. Not a big tragedy on this level of sums, but for a transaction of \$100 million, such an error might result in \$1 million loss. So this explains the need for precision, but there is also "degree to which a product or system provides the correct results." The "degree" illustrates number of operations that were executed with poor precision against the total number of operations under test, so "the smaller, the better." If this degree exhibited by an information management system is close to zero, the system is considered of the appropriate *functional correctness*. The further it goes from zero value, the less this quality attribute is present in the system or software and, depending on the importance and type of its usage, the repercussion may vary from negligible to grave.

The *operability*, understood as the "degree to which a product or system has attributes that make it easy to operate and control" (ISO/IEC 25010), exhibits its importance especially on the level of employee adaptation to the system or software and, in consequence, on the resulting effectiveness of their work. Customizability makes a logical extension of operability, for which the two basic justifications could be:

- Organizations using information management systems are highly diversified, so their employee needs are not exactly the same and the system should allow not only for easy operation but for required adaptations as well

- In the era of globalization of businesses, the organizations tend to have offices in different regions of the planet, so both operability and customizability may play a key role in quick and effective setup of theses offices.

It should be understood that the aforementioned information management system quality attributes were chosen as commonly recognized for this category of IT systems, so the recommendations presented this chapter are naturally nonexhaustive. In real life cases these attributes could be analyzed as the first, but, for the sake of the final quality of the developed system or software, should rather not be the only ones taken into consideration.

## 3.2 SOFTWARE QUALITY ENGINEERING IN AN EMBEDDED SYSTEMS ENVIRONMENT

Embedded systems make quite often the invisible part of everyday life of most people around the globe. The majority of modern appliances contain a box where some smart microprocessor executes lines of code to control the required process. But also much less harmless or safe instruments contain embedded systems that control their behavior. And while some flaw in laundry washer software quality may create a red stain on your new shirt, a similar flaw in a missile-guiding system may cost human lives.

A common perception of an embedded system is a "closed system." Today it is and is not true at the same time. The system that contains an assembly code burnt into PROM definitely is not an open one and any active operation on it would require having it installed somewhere outside its natural environment. But if some change should be needed in software residing in a Java card (a microchip credit card with Java Virtual Machine and service applets) it can be done directly inside the chip through the appropriate application programming interface (API). The same can be said about cellular phones, so are they still embedded systems or just "very micro" but nonetheless open systems?

This question may be answered through the basic analysis of the features of the existing incarnations of embedded systems. A *closed embedded system* is characterized by:

- Development: outside the actual device
- Testing:
  - inside the device—"black box" only
  - if "white box" required—outside the device in emulated/simulated environment
- Maintenance: outside the actual device *only*.

So, naturally, an *open embedded system* would allow for:

- Development: *principally* outside the actual device
- Testing: "white box" and "black box" possible
- Maintenance: inside and outside the actual device.

Why is it so important to know how open a given embedded system is? Because:

- Evaluation of quality requires measurement
- Measurement requires an access to both the code and the behavior of the embedded system (also on the level of the available stimuli)
- Evaluation of quality invokes engineering decisions
- Engineering decisions for their realization often require physical manipulation of the embedded system
- Level of openness impacts the effectiveness of manipulations and, as such, the required quality engineering results.

This list may and should invoke a very important methodological question: if the fact of the system being *embedded* influences so severely the technical means of engineering the quality into it, are the available quality-related tools applied for nonembedded systems usable at all in the embedded systems case? Before this question is answered, the quality-related tools applied for nonembedded systems should be identified. The quality engineer's toolbox consists of at least four sets of instruments:

- quality model
- quality measures and related measurement methods and techniques
- evaluation methods and techniques
- engineering models, processes, and best practices.

Returning to the question "are the available quality-related tools applied for open systems usable at all in embedded systems case?" one might be tempted to answer "no," arguing that "an embedded system is too closed and machine-dependent to allow for elaborate manipulations required by quality engineering." A plausible answer, but not the only one possible. The second, equally plausible answer could be "perhaps," as ISO/IEC 9126 and ISO/IEC 25000 state in their text that "characteristics defined are applicable to every kind of software, including computer programs and data contained in firmware." Finally, there is also a firm "yes." In order to prove the practical value of this answer, the activities presented in Fig. 3.1 should be taken into consideration.

As the main real difference between embedded and nonembedded systems that matters is the level of their "closeness," the most important activity required in order to fully apply available quality engineering techniques is to "unclose" them (through the appropriate transformation environment [TE]). This "unclosure" will allow the internal/static quality engineering (the engineering on the level of developing the source code) to be executed. External/dynamic quality and quality in use are usually decided in the design phase and evaluated when the code is actually run, so there is no need for external or in use quality-dedicated TE. The eventual interventions that may result from this higher-level evaluation of quality will have to be engineered into the code, in which case the existing TE shall be applied. In conclusion, the available quality engineering tools and methods are generally applicable to most of

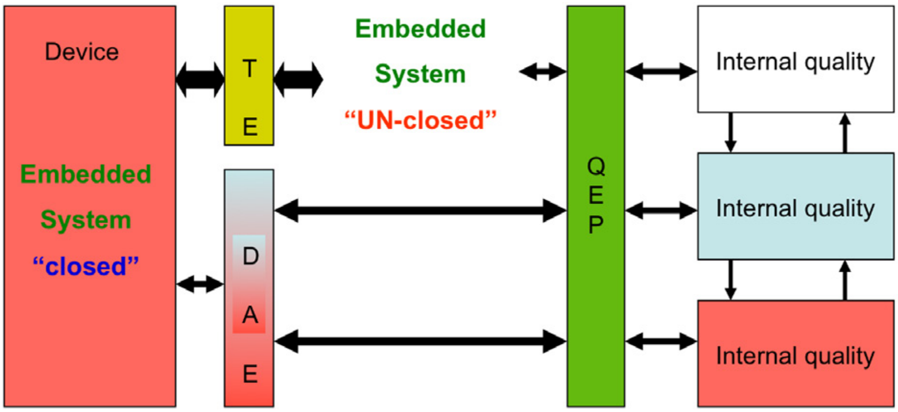

**Figure 3.1** Adaptation of quality engineering processes to embedded systems.

known types of software, even if they may require some adjustments in the development environment, but none of these adjustments create obstacles justifying negligence in engineering quality into developed embedded software.

As in any type of software, the final result is perceived through the lenses of functionality and quality. There is no real user who would accept a software system, embedded or not, having a panoply of functionalities but missing the required quality (e.g., safety or reliability). On the other hand, it would probably still be possible to convince a user to accept a software product with limited set of functionalities but exhibiting excellent quality. In other words, quality is as important for embedded systems as it is for general purpose ones. Or is it more important?

Before answering this question, it may be profitable to first analyze the areas of application of embedded software or system. Some commonly recognized applicability contexts may be defined as follows [2, 3]:

- Mobile computing
- Multimedia
- Telecom
- Automotive
- Avionics
- Industrial control and sensors
- Consumer electronics and appliance
- Military
- Health
- Space technology.

A quick analysis will immediately reveal that some of these applications are potentially more harmful than the others (such as military or health), which in turn invokes the notion of *criticality*. The criticality of a system can be defined in the following way: "A critical system is integral to the proper running of an operation. If a critical application fails for any length of time the result may be catastrophic, including loss of life, serious injury, platform or mission failure, operational chaos or even bankruptcy."

Commonly recognized categories of embedded systems such as real-time interrupt-driven (RTID), mission-, or business-critical [2, 3] each have their sets of characteristics that position them on the scale of criticality, however, such a classification is not complete enough to cover the most important criticality areas. In fact, a vaster categorization would be required to identify those that are most prone to missing quality. Such a categorization would consist of:

- Real-time interrupt-driven (RTID)
- Mission-critical
- Business-critical
- Safety-critical
- Security-critical.

When matching the aforementioned categories against the applicability fields of embedded systems a categorization matrix can be created (Table 3.1).

How exactly can one identify the impact of missing quality on the criticality of a given embedded system? Can it be predicted, measured, foreseen? If so, are the software engineers able to control and manipulate it?

The first thing that comes to mind when analyzing the impact of quality on the criticality of the embedded system is the analogy to the risk impact, understood in its classic definition. In simple terms, less quality means more risk, which, if applied to the criticality of an embedded system, translates into making its user more vulnerable than he or she would be if quality was there or, in an extreme case, pushing the embedded system to an "explosion area" (Fig. 3.2). To help us better understand this analogy, three fictive examples are discussed in the following:

- Very high criticality: an on-board control system of a missile
- High criticality: automotive—an on-board control system a vehicle
- Low criticality: appliance—an on-board control system of a washing machine.

**Table 3.1** Embedded Systems Categorization Matrix

| Application Category | Mobile Comp. | Multi media | Tele com | Auto motive | Avio nics | Health | Industrial Control | Consumer Appliances | Space | Military |
|---|---|---|---|---|---|---|---|---|---|---|
| RTID | | | | | | | X | X | | |
| Business | X | X | X | X | X | X | | | X | |
| Mission | X | | X | | X | X | | | X | X |
| Safety | | | | X | X | X | X | X | X | X |
| Security | | | | | | X | | | X | X |

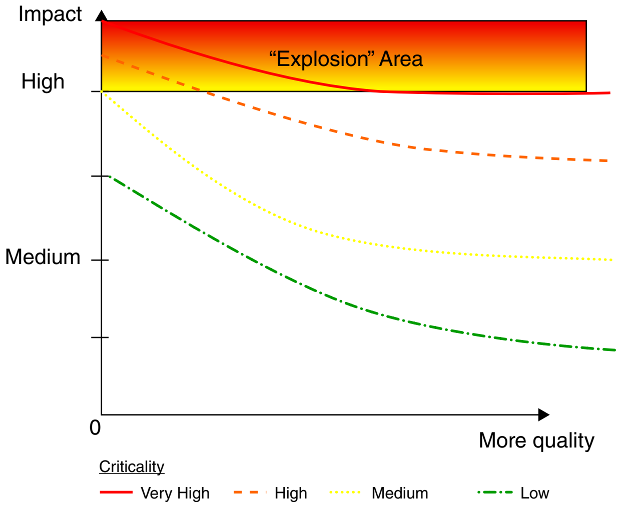

**Figure 3.2** Illustration of the impact of quality on criticality of an embedded system.

In all three cases, the quality attribute of *restorability* (the subcharacteristic of recoverability, the characteristic of reliability) has been measured. In ISO/IEC 9126-2 the attribute of *restorability* is defined as the "ability of the system to restore itself properly after an abnormal event." When analyzing theoretically the impact of low restorability in each of the three examples, one will inevitably come to the following simple conclusions:

- The missile case: if restorability is low, then the probability of an uncontrolled explosion becomes high
- The vehicle case: if restorability is low, the driver may have an accident
- The washing machine case: if restorability is low, then the probability of having dirty laundry becomes high.

In order to further analyze the most interesting example of the three, the missile, the "obtained" (measured) attribute of restorability has had the value set up at 0.5, which means that the embedded control system was able to "intelligently reboot" and restore itself in half of all observed cases. In other words, the probability of the manifestation (PM) of a wrong restore operation of the missile control system is 50%. For a missile, it looks really bad; however, there is a second condition that may moderate the severity of the situation: the occurrence of the stimulus (disturbance) that would make the control system restore (probability of disturbance, PD). In verbal form, the prediction of the impact of the restorability 0.5 could sound as follows: half of missiles having this embedded system on board will end up badly *if unexpected disturbance* requiring a restore operation *occurs*.

This expression can be translated into a classic, risk evaluation-based notation:

(1) IP = PM × PD

(2) Risk Exposure = IP × Consequences (like cost or loss of human life)

Where IP: Impact Prediction; PM: Probability of Manifestation; PD: Probability of Disturbance.

To finally build the link between quality and its impact on criticality of an embedded system, the following facts must be taken into consideration:

- Impact on criticality ≈ Risk Impact (classical definition)
- Quality is measurable (quantitative), as there exist:
  - applicable quality models (McCall, Boehm, ISO/IEC 9126, ISO 25010)
  - associated quality measures (ISO/IEC 9126, ISO/IEC 25022, ISO/IEC 25023, when published)
  - evaluation methods and techniques (ISO/IEC 14598, ISO/IEC 25040, ISO/IEC 25041, ISO/IEC 2504, when published)
- Quality is definable (qualitative)
- Applying quality engineering methods allows for identifying (critical and noncritical) technical elements of real quality of given software
- Level of quality and known technical data should allow for measuring the impact (the available technical data shall be representative to the case).

The resulting impact prediction process that encapsulates both quality evaluation and the probability of the occurrence of the source event (the "unexpected disturbance") is shown in Fig. 3.3.

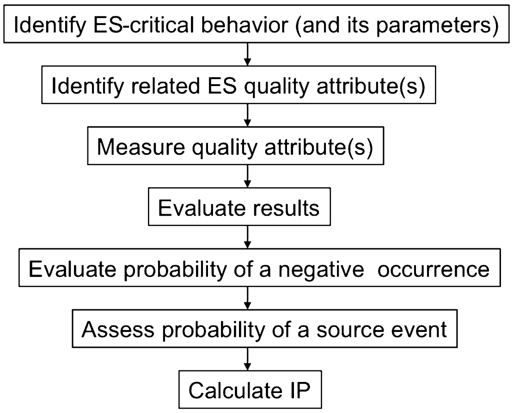
Identify ES-critical behavior (and its parameters)
Identify related ES quality attribute(s)
Measure quality attribute(s)
Evaluate results
Evaluate probability of a negative occurrence
Assess probability of a source event
Calculate IP

**Figure 3.3** The quality–criticality impact prediction process.

Using the aforementioned approach in the process of choosing quality attributes relevant to a given embedded system or software could help identify the ones that are critical to the context in which the system is being used, and by doing so, through the proper engineering activities limit the probability and potential effects of its harmful behavior.

From a practical perspective, the choice of real instruments from quality engineer's toolbox mentioned earlier in this chapter is rather limited. Among all known quality models, at the moment of writing this book, only ISO/IEC 9126 comes with associated measures and recommendations for measurement techniques (its modernized replacement ISO/IEC 25000 SQuaRE series is still not completed).

This model and its measures are accompanied by the complementary set of standards, the "sister" standard ISO/IEC 14598 (or, in case of ISO/IEC 2502x, the standards from ISO/IEC 2504x division), which offer support for quality management and evaluation, which altogether makes the only widely known complete toolset for quality evaluation.

As for quality engineering models, processes, and practices, the choice of tools is also limited, with one of them being publicly available as the SQIM model discussed in details in Section 2.3.1.

The criticality analysis presented earlier, as well as the commonly recognized applicability contexts discussed in this chapter, help identify and choose the quality characteristics (attributes) that would most frequently be recognized as "important" for embedded software or systems. They were identified as: *fault tolerance, recoverability, reliability, accuracy,* and *security*.

*Fault tolerance* is defined in ISO/IEC 25010 as the "degree to which a system, product or component operates as intended despite the presence of hardware or software faults." Its importance may be neglected in case of a washing machine, but for a missile guidance system this quality attribute is critical.

*Recoverability* is defined in ISO/IEC 25010 as the "degree to which, in the event of an interruption or a failure, a product or system can recover the data directly affected and re-establish the desired state of the system." The importance of this quality attribute was proven earlier in this chapter, when the example of embedded systems in a missile, a car, and a washing machine were discussed.

*Reliability* is defined in ISO/IEC 25010 as the "degree to which a system, product or component performs specified functions under specified conditions for a specified period of time." This quality attribute is particularly important for embedded software that either interacts directly with living beings (e.g., medical life support equipment) or its failure has a direct and immediate impact on them (avionic, automotive, military). *Accuracy* and *security* were discussed in detail in Section 3.1.

Similarly to what was remarked in Section 3.2, it should be understood that the embedded system quality attributes discussed earlier were chosen as the most commonly recognized for this category of IT systems, so the recommendations presented this chapter are naturally nonexhaustive. In real-life cases, these attributes could be analyzed as the first, but for the sake of the final quality of the developed system or software, should not be the only ones taken into consideration.

## REFERENCES

1. ISO/IEC 25010 Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models. Geneva, Switzerland: International Organization for Standardization, 2011
2. Noergaard T. *Embedded Systems Architecture*. Amsterdam: Newnes, 2005.
3. Heath S. *Embedded Systems Design*. Oxford: Newnes, 2003.