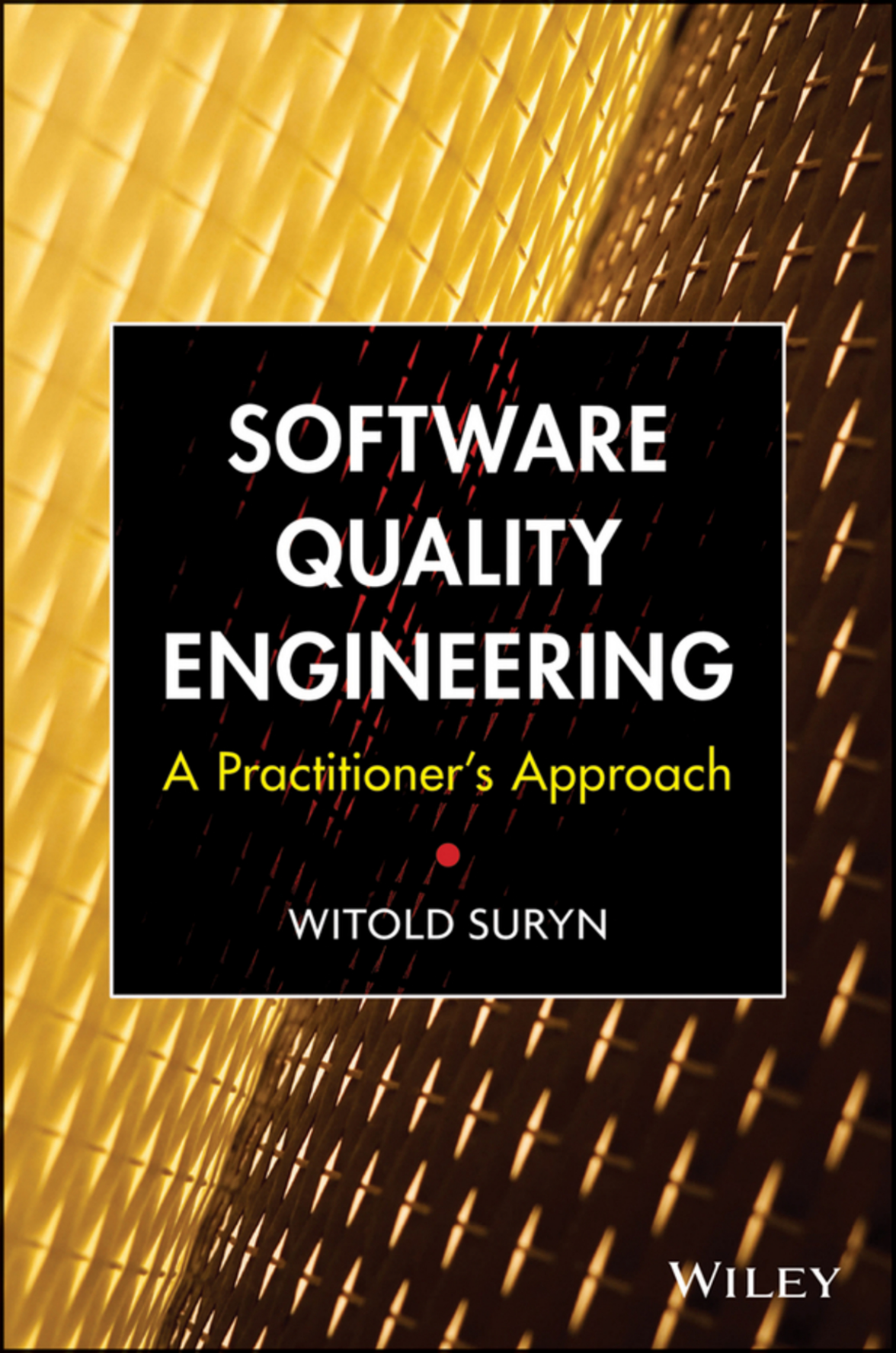
SOFTWARE QUALITY ENGINEERING
A Practitioner's Approach
WITOLD SURYN
WILEY

Software Quality Engineering

Software Quality Engineering
A Practitioner's Approach

Witold Suryn
École de Technologie Supérieure
Montréal, Canada

IEEE PRESS

WILEY

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Suryn, Witold.
Software quality engineering : a practitioner's approach / Witold Suryn.
pages cm
Includes bibliographical references and index.
ISBN 978-1-118-59249-6 (cloth)
1. Computer software–Quality control. I. Title.
QA76.76.Q35S87 2014
005.1'4–dc23
2013031271

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

For my daughter, Gabrielle

Contents

Preface

When in the early 1980s I began my adventure with information technology, I was enthusiastic, full of ideas, and profoundly naïve. I remember one of my older colleagues at the university saying, "With the microprocessor technology you just dream out what you want to do, and it will be done so." Since then I have gone through years of using the evolving information technology and every now and then I have had to stop everything I was doing to rescue it. Quite a few times I have lost hours of work, accumulated research data, and patience. . . .

Then I began asking myself the question, "Why is all that happening?" The technology is better and better, machines are more and more powerful, yet still I don't feel comfortable with keeping my important work in one place and format only. What is missing here?

About twelve years ago I found the answer: the vision for quality of information technology and systems may be there, but it is often not engineered into the products we use.

That was the moment when the idea of this book was born. The purpose of this book is to give a concise, engineering-oriented, and practical support to IT professionals and to those who are responsible for quality of the software or system they develop; those who negotiate new systems to be developed, delivered, and installed; those who will operate and use them; and those who will maintain them. The book is also intended to serve in academia as a manual to lectures that address the subject of software or system quality.

Software and system quality engineering is discussed in this book from four different perspectives: why it is important (Chapter 1), how to make it happen (Chapter 2), application contexts (Chapter 3), and what could be done to increase trust in contemporary software and systems (Chapter 4). Every chapter offers both a layer of theoretical introduction required to correctly grasp its content and a practical part that offers hands-on recommendations.

The effective use of this book depends on the reader's level of familiarity with the subject of software and systems quality. For the readers who possess practical knowledge of software and systems quality-related standards (ISO, IEEE), a considerable part of theoretical introduction may be deemed unnecessary. For the beginners or those who want to reorient their practices toward disciplined, standards-based approaches to engineering quality into software or a system, following the path of theory to practice is recommended. Finally, the practitioners who feel very comfortable with quality engineering matter may even go directly to Chapter 2, as it offers a lot of support in terms of practical identification, definition, and execution of engineering "to-dos" required in the process of developing a system or software that possesses both functionalities *and* quality.

Witold Suryn

Chapter 1

Why Software Quality Engineering?

Quality has become a critical attribute of software products as its absence produces financial, health, and sometimes life losses. At the same time the definition, or scope, of the domain of software quality has evolved continuously from a somewhat technical perspective to a perspective that embraces human aspects such as usability and satisfaction.

An increasing business-related recognition of the importance of software quality has also made software engineering's "center of gravity" shift from *creating an engineering solution* toward *satisfying the stakeholders*. Such a shift very clearly reflects the trend within the community of stakeholders who more and more often say: "I do not want to know about *bits and bytes*. I want a solution that satisfies my needs." The critical word here is "satisfaction," for it covers both functional and quality perception of the software solution being used.

Development organizations confronted with such an approach are, in general, not entirely prepared to deal with it even if their engineers are adequately educated. Moreover, if the education is there, it is quite often acquired through experience rather than a regular educational process, as the software engineering curricula being offered, with few exceptions [1], do not emphasize the importance of teaching software quality engineering.

One of practical responses to such a situation was the development of Software Engineering Body of Knowledge (SWEBOK) [2]. SWEBOK seeks to provide the knowledge that allows universities to build such educational programs that will allow producing professionals able to stay abreast of the fast-moving industry, but it also adds a scientific and innovative component to *best practices*. The continuation of this approach is this book.

So let's ask the question, "why software quality engineering?" as three partial questions:

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

Published 2014 by John Wiley & Sons, Inc.

- *Why software?* Because in contemporary social life software, systems and services rendered by software are omnipresent, beginning with the watches we wear, ending with nuclear electricity plants or spaceships.
- *Why quality?* Because if these instances of software work without the required quality we may be late, dead, or lost in space.
- *Why engineering?* As in every technical domain, it is engineering that transforms ideas into products, it is the verified and validated set of "to-dos" that help develop the product that not only has required functionalities but also executes them correctly.

To make this picture complete, another question should be asked: *why at all?* There is in fact only one reason: the user. Despite decades of evolution of information technology and its tools, the user still faces risky, unreliable, and quite often unintelligent products that far too often waste his or her time or money, and wear off his or her patience. So quality engineering applied to software, systems, and related services is intended to assist developers in building good, intelligent, and reliable products; to help users request and verify their quality needs; and for those who want to use software as easily as they use a dishwasher, to shield against faulty products and unprofessional suppliers.

1.1 SOFTWARE QUALITY IN THE REAL WORLD

For the users, a software product more and more often corresponds to a black box that must effectively support their business processes. As a consequence of this natural approach, business needs become a driving force of quality software product development and a stakeholder moves to the position of a car buyer and user rather than an involuntary expert in software engineering. And what he or she perceives at the end corresponds to expressed satisfaction at using a software product that possesses *both* required functionalities and required quality. When one of them is missing, a painful process of improvements and negotiations takes place to often end by changing the supplier and replacing the product with one that is *mature* enough do its job well on both accounts.

What exactly constitutes the quality of a product is often the subject of hot debate. The reason the concept of quality is so controversial is that there is no common agreement on what it means. For some it is "degree to which a set of inherent characteristics fulfills requirements" [3], whereas for others it can be synonymous with "customer value," or even "defect levels" [4]. A possible explanation as to why any of these definitions could not win a consensus is that they generally do not recognize different perspectives of quality, such as for instance the five proposed by Kitchenham and Pfleeger [5]:

- The transcendental perspective deals with the metaphysical aspect of quality. In this view of quality, it is "something toward which we strive as an ideal, but may never implement completely."

- The user perspective is concerned with the appropriateness of the product for a given context of use.
- The manufacturing perspective represents quality as conformance to requirements. This aspect of quality is stressed by standards such as ISO 9001 [6] or models such as the Capability Maturity Model [7].
- The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product.
- The final perspective of quality is value-based. This perspective recognizes that the different perspectives of quality may have a different importance, or value, to various stakeholders.

One could argue that in a world where conformance to ISO and IEEE standards is increasingly present in contractual agreements and used as a marketing tool, all the perspectives of quality are subordinate to the manufacturing view. This predominance of the manufacturing view in software engineering can be traced back to the 1960s, when the U.S. Department of Defense and IBM gave birth to Software Quality Assurance [8]. This has led to the belief that adherence to a development process, as in manufacturing, will lead to a quality product. The corollary to this belief is that process improvement will lead to improved product quality.

This opinion is not shared unanimously, as some parts of both industry and academia find it inaccurate or at least flawed. For example, G. Dromey states:

> The flaw in this approach [that you need a quality process to produce a quality product] is that the emphasis on process usually comes at the expense of constructing, refining, and using adequate product quality models [9].

Kitchenham and Pfleeger reinforce this opinion by stating:

> There is little evidence that conformance to process standards guarantees good products. In fact, the critics of this view suggest that process standards guarantee only uniformity of output [5].

Furthermore, data available from Agile [4] projects show that high quality is attainable without following a manufacturing-like approach.

However, some studies conducted at Raytheon [10] and Motorola [11] showed that there is indeed a correlation between the maturity level of an organization as measured by the Capability Maturity Model (CMM) and the quality of the resulting product. These studies provide data on how a higher maturity level (as measured by the CMM) can lead to:

- Improved error/defect density (i.e., the error/defect density lowers as maturity improves)
- Lower error rate
- Lower cycle time (time to complete parts of the lifecycle)
- Better estimation capability.

From these results, one could conclude the quality can be improved by following a mature process. Studies of the development of lifecycle models presented by Georgiadou [12] indicate that the maturity of the development process is reflected by the emphasis and allocation of testing and other quality assurance activities. The study demonstrated that the more mature the process and its underlying life cycle model, the earlier the identification of errors in the deliverables. However, these measured improvements are directly related to the manufacturing perspective of quality. Therefore, such quality improvement efforts fail to address the other perspectives of quality. This might be one of the reasons for the perception of the "quality problem" as one of the main failings of the software engineering industry. Furthermore, studies show that improvement efforts rooted in the manufacturing perspective of quality are difficult to scale down to smaller projects and/or smaller teams [13, 14]. Indeed, rather than being scaled down in smaller projects, these practices tend to be not performed at all.

Over recent years, researchers have proposed new approaches and models that try to encompass more perspectives of quality than just the manufacturing view. Geoff Dromey [9, 15] proposed such a model in which the quality of the end product is directly related to the quality of the artifacts that are a by-product of the process being followed. The reasoning is that if quality artifacts are correctly designed and produced throughout the life cycle, then the end product shall manifest attributes of good quality. This approach can clearly be linked to the product perspective of quality with elements from the manufacturing view. This is certainly a step from the manufacturing-only approach, but it fails to view the engineering of quality as a process that covers all the perspectives of quality. In Pfleeger and Atlee [16], the reader can find valid arguments against approaches that focus only on the product perspective of quality:

> This view [the product view] is the one often advocated by software metrics experts; they assume that good internal quality indicators will lead to good external ones, such as reliability and maintainability. However, more research is needed to verify these assumptions and to determine which aspects of quality affect the actual product's use.

All of this may be true to a certain extent, but what ultimately counts is a customer's *yes* said after the delivery is finalized.

Another absolutely natural trend observable within the "population of IT customers" is the desire to be properly served without having to become proficient in information technology. A customer just wants to buy, learn how to use, and then simply use a software product, just as he or she does with a car or a TV. This boils down to an extended (or shall we just say "professional and mature") responsibility of a software supplier, who now has to know not only what the customer is able to express, but also what the customer does not know that he or she knows. And then, when all questions are asked and answered, the supplier must continue on his or her way until the product is built and delivered to the customer's satisfaction.

Similarly to mathematics, the most important part of software and software quality engineering is to understand the problem. Whatever comes after is the result

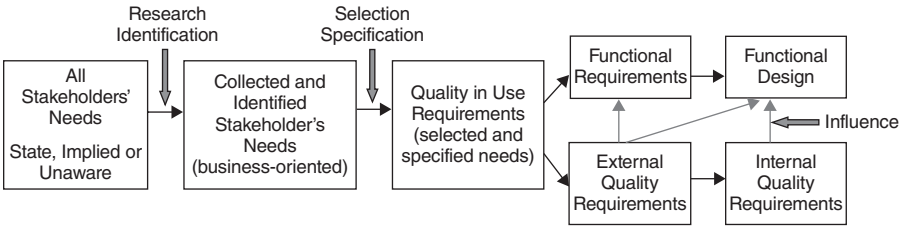

Figure 1.1 From "stated, implied, and unaware" needs to fully defined software product (based on personal communication of M. Azuma).

of knowledge applied to this understanding and, if we make an assumption that such knowledge exists, the final outcome makes the "executable form" of what was understood. The graveness of this statement is expressed by different kinds of statistics showing billion-dollar losses resulting from bad or incomplete understanding of the problem called a software product (a simple search on Google brings up thousands of hits on this subject). In case of software quality, the situation is even more dire, as the primary source of information, a customer, is usually able to at least signal his or her "functional" needs, but in the majority of situations is not knowledgeable enough to identify or discuss in precise terms the quality requested from the product under discussion. When it comes then to analyzing why something bad happened, customers blame suppliers (which is understandable) but the suppliers do not stay behind. From a purely professional point of view, one might ask: "Is that fair?" Who, between the user and the supplier, is supposed to be an expert, especially in a subject so difficult to define as quality? Should not it be the supplier who follows the process from Fig. 1.1 (with the customer having his or her "stated, implied or unaware" needs), in order to solicit, identify, and define required quality attributes and then later develop a software product that exhibits them? This question is a keynote and the main subject of this book.

1.1.1 Consumer Perspective

When a car manufacturer asks a customer about his or her opinion on the vehicle the latter uses or was using, the manufacturer, in fact, asks about an overall perception of the car in question, including both functionalities and the quality associated with them. In the case of software and even more in the case of software systems, the overall perception (or satisfaction) is heavily influenced by the verifiable existence of quality. During his many years of working in the IT industry and then teaching at a university, the author had the unique chance to ask the following question to IT professionals, students, and customers: "What would you be more inclined to accept, a system with a rich set of functionalities but with lower quality or the one with limited functionalities but with high quality?" The choice was in 99% of cases the same: the second. Interesting that the choice was identical even

if the interviewed persons were from different sides of the IT market "barricade," suppliers and users. Obviously, the choice becomes less firm when suppliers return to their workstations (or we would have only high quality and bug-free software) but still, such unanimity may be interpreted as a good sign. The choice may sound "generic" as the reaction, but its real context varies for a supplier and a consumer and even inside these categories, as in the case of an individual and a corporate consumer.

1.1.1.1 Individual Consumer

In the majority of cases the individual consumer is a person with no face and no name. Unfortunately, the consumer quite often has no rights, too. The simple fact that almost every software on the planet before installation requires the acceptance of license terms that virtually free the manufacturer from any responsibility makes the existence of quality an extra effort that has in mind the good reputation of the supplier rather than the well being of the user. Currently, no known legal case initiated by an individual user against an IT giant has been won. The most common individual user reaction to a software malfunction is "reboot, and pray it works a bit longer." So, in a way, it is the user who is responsible for his or her own misery, for instead of (massively) protesting, even suing, the consumer tends to sit tight and stay quiet. There is also another perspective from which the subject may be looked at: how big is the population of faceless and disenfranchised users who experience serious troubles with individual user-targeted software? How many of us stretch the application to its limits and how many just float in the main and central current of available functionalities? From what we may observe, the latter category is dominant, or the risk of huge financial losses would motivate the suppliers better. What could (or should) be done then to assure the minimal, acceptable quality of *any* software for a Mr. John Doe? One of the emerging options is the *certification* of IT products for the individual user market. The real value of the certification is however linked to the existence of real consequences, be they financial, legal, or even only hitting someone's reputation. If the customer was inclined *not* to buy a noncertified IT product, the supplier would be motivated enough to see quality as an obligation, not as an option. The certification itself could be applied *de jure* or *de facto*, depending on the level of pressure a given society would decide to apply. One of the most important aspects of "quality for John Doe" is the identification and definition of what exactly constitutes the *minimal, acceptable quality level*. This definition would then become a pass/fail criterion used in the certification process. The very interesting beginnings of activities aiming to increase individual consumer IT products' quality can be observed in several countries such as France (Infocert [17]) and Poland (SASO [18]), or on an international level (Quality Assurance Institute [19]). Even if none of them is officially sanctioned as government requested, the market itself reacted in surprisingly positive way. In case of SASO Poland, several local IT corporations requested the possibility to begin certification process, finding it the obvious option for proving the reliability of their products and, in consequence, enhancing their market reach.

1.1.1.2 Corporate Consumer

Corporate consumers may not always have one, identifiable face but in most cases they have recognizable power to demand and obtain. The fact of being "corporate" does not limit this category of IT customers to using only big IT structures, be it a system developed on demand or an individualized suite (like the ones from SAP or Oracle). On the contrary, simpler office applications play a substantial role in corporate world even if they are not used to serve business-critical processes.

From the perspective of IT proficiency, corporate consumers may be put in two distinctive categories: pure users and user-operators. Pure users are those who make the customers of system integration organizations (SIOs) such as HP Enterprise Services (formerly Electronic Data Systems (EDS)) or Oracle. Their business philosophy is "focus on what we know how to do and pay for required specialized services." In many cases the corporate customer of an SIO not only pays for the system, its installation, and required user training, but also pays for further operation and maintenance. Such a business arrangement, popularly known as *outsourcing*, seems to be a win-win solution for all involved. In theory everybody does what he or she knows best; the user focuses on his or her core business without the burden of having his or her own IT team, and the SIO runs the system with all required professionalism and responsibility. What is the place of (in this case) system quality engineering in it?

The simple fact of separating business processes and activities from the running IT machine that supports them puts the whole quality engineering responsibility on the side of a supplier (e.g., an SIO). The customer pays, among other things, to be able to express his or her needs and to be correctly understood using principally the taxonomy natural to his or her business. In consequence, a somehow trivial statement of "I will open a new facility in Japan that has to operate 24/7" will have to be translated into a set of precise functional and quality requirements for the supporting IT system by a supplier, who further should initiate a series of technical meetings where the customer's functional and quality needs are explained, negotiated, understood, and finally agreed upon. The difficulty of this challenge gets bigger when a discussion of quality takes place. Although questions concerning functional aspects of the system are usually easily understood and answered by an IT-unfamiliar user, a question such as "what are your usability requirements?" may raise a few brows. So the supplier not only has to identify his or her customer's quality requirements, but also has to explain them, verify them, and get the fully informed customer's approval, and then engineer them into the system.

The corporate customer's supplier's responsibility does not end with installing the system, training the staff, and turning the key in the ignition. As the parties are known by name and bound by elaborate contracts, the repercussions of missing quality may be traced back and legal and financial consequences can eventually be imposed on the guilty party. If an *outsourcing* contract has been signed, the responsibility for the system and its quality stays on the side of the supplier for the length of the contract.

In case of user-operators the quality engineering problem may be slightly less difficult, as this category of corporate customers is usually "IT-savvy." The biggest

challenge in the whole process of engineering quality, the identification and definition of quality requirements, may eventually be achieved through discussions in domain-specific language and applying domain-specific models and knowledge (e.g., using the ISO/IEC 25000 series of standards [20]), so the road to a correct understanding of quality needs for the given system is shorter and faster.

Then, after the installation and all required training, the system usually goes under the operation and daily maintenance of the user's IT team, with the supplier granting the support and warranty for a given period. Analyzing the responsibility for quality engineering in this type of situation brings a three-phase view: in the phase of the development and transition, it is supplier's sole responsibility; in the phase of the user's operation covered by supplier's warranty, the responsibility is "distributed" and creates the majority of conflicting situations because there is more than one entity manipulating the system; and in the phase of the whole remaining system life time the responsibility for engineering quality is entirely the user's.

One may ask, "what engineering of the quality may take place when system is in its operation phase?" More of this subject will be discussed in Chapter 2.

1.1.2 Supplier Perspective

The supplier's ultimate justification for developing any product is the profit, usually calculated in terms of the return on investment (ROI). It is widely known and accepted that developing functionalities of the system or software requires appropriate budget, but it is much less publicly obvious that engineering the quality into these functionalities costs money as well, and that it is not cheap. There is another aspect of quality that makes it "a child of a lesser god" in eyes of a developer: too often its presence or absence manifests itself after a considerably long time of operation. With the quality of a system or software it is like with a pair of shoes: their "functionalities," such as shape, color, and size, can be seen immediately, but verifying their "qualities," such as real quality of materials used or comfort in use, requires time and operation (walking a few kilometers) to be applied.

These two elements make up the basic reasoning for quality-related decisions. In other words, if the quality is so expensive that it will make the price prohibitive or eat up the profit, it will be reduced to a passable minimum. If, further in this direction, its lack will not be immediately noticed or will not reach the "pain threshold" of the user, it will also be reduced or even neglected. The third element in quality-related decision making is the famous *time-to-market*, the offspring of competition. On one hand, the competition makes a supplier try to build a better product than the other suppliers, but on the other hand, it creates a strong time pressure to reach the market before the competitors, and that always requires compromises. Depending on the corporate philosophy and culture of the supplier, the compromises may be applied to both functionalities and quality or to quality only.

Financial influences are not the only ones that decide the final quality of a software product or system. Quality requires engineering knowledge comparable to that used in development, but this knowledge is far younger and still in dynamic

evolution. In Chapter 2, quality engineering processes and activities are discussed in detail, but to create a simple, common reference for the two following chapters, these processes are named here:

- Identification and definition of quality requirements
- Transformation of requirements into quality attributes of the future software or system
- Transformation of quality attributes into engineering "to-dos" that can be communicated to developers and further realized
- Identification and estimation of interdependencies between development and quality engineering activities
- Design of quality measurement (design of quality tests)
- Quality measurement
- Quality evaluation.

In conclusion, the supplier's perspective on quality engineering is the result of a combination of financial constraints, software quality engineering knowledge existing in the organization, and the user's tolerance to poor quality.

1.1.2.1 Off-the-Shelf Software Products

Off-the-shelf (OTS) software products are "software product(s) available for any user, at cost or not, and used without the need to conduct development activities" [21]. This definition of OTS indicates the main targeted user as the one discussed in Section 1.1.1.1, a nameless, faceless customer. The suppliers of OTS software face all quality-related dilemmas discussed previously, that is, cost, manifestation, knowledge, and time, and from what can be seen in the market, they do not deal with them too well. In their seventh decade, information technology companies still happen to deliver unreliable, poorly engineered, and sometimes surprisingly user-unfriendly products. Why?

Besides a short budget, undemanding customers, and lack of required knowledge, an OTS supplier is exposed to another challenge: a difficulty in communication with the users. A massive user is an unknown user, not reachable directly, and unfortunately also rather IT-ignorant, so not helpful in identifying missing or required quality. Then how exactly is the OTS developer supposed to build a product of required quality if he or she cannot talk to his or her customers?

There are several possible approaches to helping the developer create an OTS product of correct and appreciated quality. Some of them are:

- Collecting users' feedback through surveys
- Collecting detailed crash reports
- Sociological analysis of the targeted user groups
- Extrapolation.

In order to be effective, *collecting users' feedback* in the domain of quality requires a considerable effort of design. The questionnaire cannot be too long because the