Chapter 2

Quality Culture

After completing this chapter, you will be able to:

- understand the cost of quality;
- recognize the signs of a quality culture;
- identify the compromises of the five dimensions of a software project;
- know and follow the software engineer's code of ethics.

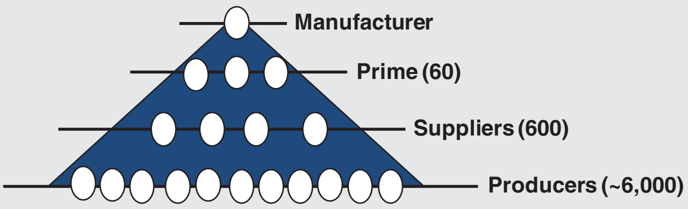
2.1 INTRODUCTION

In this chapter, we consider the concepts of the cost of quality, quality culture, and the code of ethics for software engineers. The issues related to quality will be applied to the context of software development. The principles set out in this book apply to those who develop, maintain, and work on information technology infrastructure, be it computer technicians, management computer specialists, or software and IT engineers. Typically, software engineers graduate from a school of engineering and are members of a professional order. We know that the term "software engineer" is currently used freely in both universities and the business world. Very few software engineers have obtained certified university degrees. For the engineer who is a member of a professional association, quality is part of the civil liabilities for the profession and must be managed prudently.

Non-quality in software has led to several catastrophes. In this chapter, we present examples where poor quality culture and the lack of a code of ethics have led to disastrous situations involving poor quality software and irreversible damage to people and the environment.

The following description is of the case of a Canadian medical device for treating cancer (Therac-25), which caused the death of several patients from massive irradiation. Similar incidents have occurred with devices developed by other companies throughout the world. We describe the Therac-25 case because it has been extensively studied and documented.

A large Japanese electronic product manufacturing company uses a considerable number of suppliers. The supplier pyramid is made up of a first level with some sixty suppliers, a second level with a few hundred suppliers, and a third level with a few thousand very small suppliers.



A software defect for a component produced by a third-level supplier caused a loss of over $200 million for this manufacturer.

Adapted from Shintani (2006) [SHI 06]

We believe that setting up a quality culture and software quality assurance (SQA) principles, as stipulated in the standards, could help solve these problems.

It is clear that quality, which is influenced by your organization's senior management and the organization's culture, has a cost, has a positive effect on profits, and must be governed by a code of ethics.

"In today's software market, the main focus is on costs, schedule and functions, and quality is lost in the noise. This is regrettable since mediocre quality performance is at the heart of most software costs and scheduling problems."

Watts S. Humphrey

In the software industry, there are still too many quality problems that are becoming increasingly severe and with more consequences (consult the Incidents and Horror Stories Involving Software in the appendix at the end of this book). In the following text box, we describe the case of a defective medical device, the Therac-25, which injured and killed patients. Its software had many defects.

The Therac-25 Medical Device

Computers are increasingly being introduced into safety-critical systems and, as a consequence, have been involved in accidents. Some of the most widely cited software-related accidents in safety-critical systems involved a computerized radiation therapy machine called the Therac-25. Between June 1985 and January 1987, six known accidents involved massive overdoses, by the Therac-25, resulting in deaths and serious injuries.

In 1982, the Therac-25 replaced older models of the Therac-6 and Therac-20 radiation therapy machines at Atomic Energy Canada Limited (AECL). The new model was computer/software controlled, whereas the older models were controlled by electromechanical components. In the new model, the software was supposed to immediately stop the machine/treatment in the event of a malfunction. This is very complex and expensive equipment, costing around \$1 million. Some software of Therac-25 had been recently created, whereas other parts had been reused from the previous models. However, reused software from older models was not essential to the safety of the treatment since it was the electrotechnical components that ensured the additional safeguard.

During the incidents, the patients complained of severe burns, and radiation overdose was not initially identified as a possible cause of these accidents. In one case, the patient complained of having a burning sensation. The technician told the patient that there was nothing indicating that a problem had occurred during the treatment. This same patient had to have a mastectomy and subsequently lost the use of one arm and her shoulder.

Another incident occurred in Ontario. During the treatment, the machine shut down after a few seconds and indicated that no dose was delivered to the patient. The operator then pressed the Proceed command key. The machine shut down again. The operator repeated the process a few times and the machine displayed “no dose” delivered. The patient complained of a burning sensation. The patient died a few months later of an extremely virulent cancer. An AECL technician later estimated the patient had received between 13,000 and 17,000 rads. Typical single therapeutic doses are in the 200-rad range. Doses of 1000 rads can be fatal if delivered to the whole body.

Later that year, another patient was burned while the unit indicated it was in pause mode. This patient died of his injury 5 months later. Failing to reproduce the problem, AECL concluded that the patient was probably burned by an electrical shock unrelated to the radiation treatment. An independent company was asked to check the device and found no electrical problems.

After this incident, the US Food and Drug Administration reported to AECL that this device did not operate in accordance with the law and required AECL to inform all of its users about these problems and possible consequences to the patients. Users of the Therac-25 formed a user group and questioned AECL about the lack of transparency concerning these accidents. Sometime later that year, the newspapers started publishing stories about two of the accidents that had occurred.

It seemed that AECL did not consider that the software could have been the cause of these incidents since AECL had reused software that worked well in previous models. It was thought that by reusing software that was functioning and verified, this would automatically ensure its reliability. They had forgotten that certain parameters of the new model were different than the older models. Electromechanical devices in the older models were overriding software defects and preventing equipment malfunctions. With the Therac-25 model, the company had, for cost reasons, removed the electromechanical devices and thus exposed the faulty software instructions and harmed patients by inadvertently removing the additional safety feature.

Basic software-engineering principles that apparently were violated with the Therac-25 include:

- documentation should not be an afterthought;
- software quality assurance practices and standards should be established;
- designs should be kept simple;
- ways to get information about errors, for example, software audit trails, should be designed into the software from the beginning;
- the software should be subjected to extensive testing and formal analysis at the module and software level; system testing alone is not adequate.

Furthermore, these problems are not limited to the medical industry. It is still a common belief that any good engineer can build software, regardless of whether he or she is trained in state-of-the-art software-engineering procedures. Many companies building safety-critical software are not using proper procedures from a software engineering and safety-engineering perspective.

Most accidents are system accidents; that is, they stem from complex interactions between various components and activities. To attribute a single cause to an accident is usually a serious mistake. In this article, we hope to demonstrate the complex nature of accidents and the need to investigate all aspects of system development and operation to understand what has happened and to prevent future accidents.

The Therac-25 accidents are the most serious computer-related accidents to date (at least non-military and admitted) and have even drawn the attention of the popular press.

Adapted from Leveson and Turner (1993) [LEV 93]

Readers are invited to read the 24-page article, available on the internet, written by Leveson and Turner (1993) [LEV 93], describing the six accidents involving massive overdoses to patients in more detail as well as a description of the investigations conducted to discover the multiple causes of the Therac-25 accidents.

2.2 COST OF QUALITY

One of the major factors that explains the resistance to implementing quality assurance is the perception of its high cost. In organizations that develop software, it is rare to find data on the cost of non-quality.

In this part of the book, we define the components of the cost of quality. Next, we discuss the cost of quality used in a major American company working in the military industry, and the data collected by Professor Claude Y. Laporte while working with different organizations. Lastly, we present a case study using data collected from a major transportation equipment manufacturer in Canada, Bombardier Transport.

"We never have time to do work correctly the first time, but we always manage to find the time to redo the work one or two times."

Anonymous

Thinking in terms of costs helps attract the attention of administrators. This approach means better positioning of the software quality file and confronting the sometimes negative perceptions of company administrators, project managers, and often engineers from other disciplines (e.g., mechanical engineer). This approach is preferred to a technological discussion since the terminology of costs is common to almost all administrators, and a constant concern for management. Thus, SQA must strive to present the software quality improvement file in a way that is homogenous and consistent with the company's other investments, that is, as a business case for the organization. It is also a professional way to present an investment file.

As software engineers, you are responsible for informing administrators of the risks that a company takes when not fully committed to the quality of its software. A practical manner of beginning the exercise is to identify the costs of non-quality. It is easier to identify potential savings by studying the problems caused by software.

To convince administrators of the necessity of quality software, they must be shown the impacts of not having quality software. This can be done by collecting data on the experiences of a company that invested in quality programs versus one that did not. They must be presented with the positive results and benefits obtained from all aspects of applying quality, as well as the negative effects and failures due to a lack of quality or a concern for quality.

The costs of a project can be grouped into four categories: (1) implementation costs, (2) prevention costs, (3) appraisal costs, and (4) the costs associated with failures or anomalies.

If we are certain that all development activities are error free, then 100% of costs could be implementation costs. Given that we make mistakes, we need to be able to identify them. The costs of detecting errors are appraisal costs (e.g., testing).

Costs due to errors are anomaly costs. When we want to reduce the cost of anomalies, we invest in training, tools, and methodology. These are prevention costs.

"Quality is not only an important concept, it is also free. And it is not only free, it is the most profitable product that we produce! The real question is not how much a quality management system costs, but what is the cost of not having one."

Harold Geneen
CEO ITT Corporation

The "cost of quality" is not calculated in the same way in all organizations. There is a certain amount of ambiguity between the notion of the cost of quality, the cost of non-quality, and the cost of obtaining quality. In fact, this is a non-issue, regardless of what it is called, it is important to clearly identify the different costs taken into account in our calculations. To do so, we can see how other companies have dealt with this.

In the most commonly used model at this time, costs of quality take into account the following five perspectives: (1) prevention costs, (2) appraisal costs, (3–4) failure costs (internal during development, external on the client's premises), and (5) costs associated with warranty claims and loss of reputation caused by non-quality.

The only difficulty with this model is clearly identifying the activities associated with each perspective and then tracking the actual costs for the company. We must not underestimate the complexity of this second step! The calculation of the cost of quality in this model is as follows:

$$\begin{aligned}\text{Quality costs} = {} & \text{Prevention costs}\\ & + \text{Appraisal or evaluation costs}\\ & + \text{Internal and external failure costs}\\ & + \text{Warranty claims and loss of reputation costs}\end{aligned}$$

– Prevention costs: This is defined as the cost incurred by an organization to prevent the occurrence of errors in the various steps of the development or maintenance process. For example, the cost of training employees, the cost of maintenance to ensure a stable manufacturing process, and the cost of making improvements.

– Appraisal costs: The cost of verifying or evaluating a product or service during the different steps in the development process. Monitoring system costs (their maintenance and management costs).
– Internal failure costs: The cost resulting from anomalies before the product or service has been delivered to the client. Loss of earnings due to non-compliance (cost of making changes, waste, additional human activities, and the use of extra products).
– External failure costs: The cost incurred by the company when the client discovers defects. Cost of late deliveries, cost of managing disputes with the client, logistical costs for storing the replacement product or for delivery of the product to the client.
– Warranty claims and loss of reputation costs: Cost of warranty execution and damage caused to a corporate image as well as the cost of losing clients.

In the late 1990s, a hospital in the Montreal region developed a clinical research management platform. Before a manager who was specialized in medicine and was an advocate of software quality joined the hospital, between 40% and 50% of defects were identified by the doctor-users of the software for each version. After the manager set up software quality processes, defects dropped by 20% within 1 year. The development team was delighted with the results, and especially with the idea of not having to rework half the functions with each version. However, management found these processes to be too long and decided to lay off the manager. After his departure, the process was abolished, and several of the best employees left the hospital because they were demotivated when old habits set in again.

In addition to the costs listed above, an organization may suffer other repercussions following the development of a defective software such as lawsuits, penalties imposed by the court, a deterioration of the market value of their shares, the withdrawal of financial partners, and the departure of key employees.

Krasner [KRA 98] published a table to better understand the activities and costs for three quality perspectives (see Table 2.1).

Cost of quality models started appearing in the 1950s based on the work of Feigenbaum and colleagues. They proposed a method for classifying the costs associated with the quality assurance carried out on a product. They proposed an economic point of view and empirically studied the relationship of each perspective on total cost.

Table 2.1 Categories of Software Quality Costs

Major categories	Subcategories	Definition	Typical costs
Prevention cost	Quality basis definition	Effort to define quality, and to set quality goals, standards, and thresholds. Quality trade-off analysis	Definition of release criteria for acceptance testing and related quality standards
	Project and process-oriented interventions	Effort to prevent poor product quality or improve process quality	Process improvement, updating of procedures and work instructions; metric collection and analysis; internal and external quality audits; training and certification of employees
Appraisal or evaluation cost	Discovery of the condition of the product	Discovery of the level of non-conformance	Test, walk-through, inspection, desk-check, quality assurance
	Ensuring the achievement of quality	Quality control gating	Contract/proposal review, product quality audits, "go" or "no go" decisions to release or proceed, quality assurance of subcontractor
Cost of anomalies or non-conformance	Internal anomalies or non-conformance	Problem detected before delivery to the customer	Rework (e.g., recode, retest, re-review, re-document, etc.)
	External anomalies or non-conformance	Problem detected after delivery to the customer	Warranty support, resolution of complaints, reimbursement damage paid to customer, domino effect (e.g., other projects are delayed), reduction of sales, damage to reputation of enterprise, increased marketing

Source: Adapted from Krasner 1998 [KRA 98].

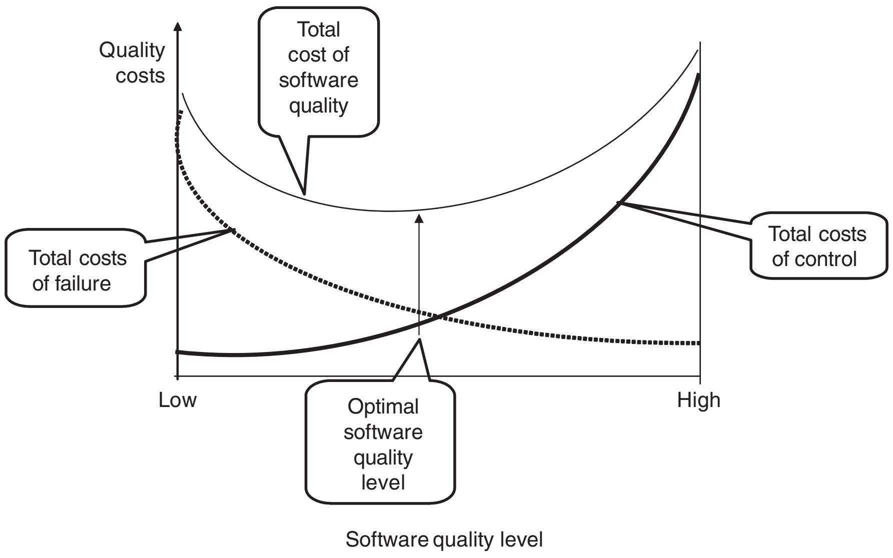


Figure 2.1 Balance between the software quality level and the cost of quality [GAL 17].

As illustrated in Figure 2.1, increased detection and prevention costs lead to reduced costs related to failures (internal or external) and vice versa: a drop in the costs of detection and prevention leads to an increase in the costs related to failures. As well, there is a relationship between the level of software quality and the total cost of detection and prevention. The discussion in the previous section on the different business fields enables us to see that software with high criticality will need a higher quality level than software with lower criticality, which will mean increased detection and prevention costs.

A

Acme Communications Inc. received a new project. The project involved automating a survey system for a company that provides consulting services to contractors. The company already had experience in developing survey software.

After meeting with the client, the project manager analyzed the information and prepared a two-page document that specified his understanding of the requirements. The estimate was 40 hours of work for the first delivery. The project was considered to be so simple that it was assigned to an intern. The project manager, without having consulted with the previous developers, asked an intern to use the source code of an existing application that he believed to be similar. Thirty hours after the development began, the intern had not yet understood the source code. The manager had already promised the first delivery within the time estimated, so there remained only 10 hours and the entire weekend for the version to be ready for the following Monday. The manager felt it necessary to ask a former developer to do the work.

The developer's analysis showed that this application was new and that there was no similar one existing in the company. In other words, the project had to be started from scratch. The manager, anxious after hearing this information, asked the developer whether he could work the entire weekend to deliver something to the client. The developer saw this as a tremendous opportunity and asked for a 25% salary raise.

The first version was delivered, but with only 70% of the features required. The final version was completed after requiring an extra 20-hour workload. The project initially estimated at 40 hours instead took a total of 84 hours.

The first objective of the SQA is to convince management that there are proven benefits to SQA activities. To do this, he must be convinced of this himself. We all have been taught the following: "identifying an error early in the process can save a lot of time, money and effort."

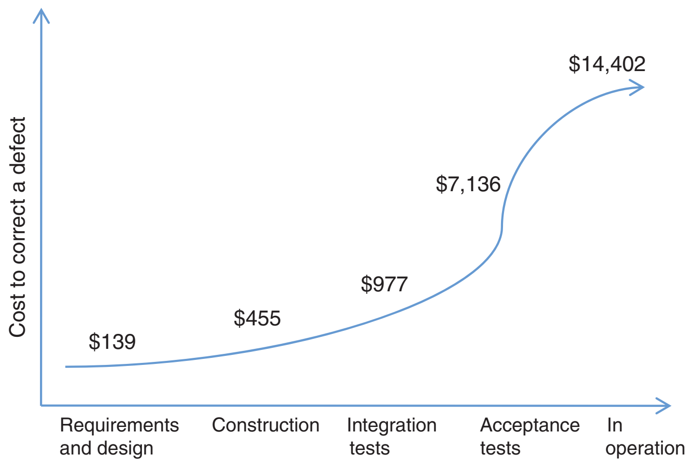


Figure 2.2 Costs of propagating an error[1] [JON 00].

After years spent studying and measuring software practices, Capers Jones [JON 00] estimated the average costs of fixing defects in the software industry (see Figure 2.2). The simulations show that it is profitable to identify and fix a defect as soon as possible. A defect that arises during the assembly phase will cost three times more to fix than one that is corrected during the previous phase (during which we should had been able to find it). It will cost seven times more to fix the defect in the

[1]These costs are based on the value of the American dollar in 1981.

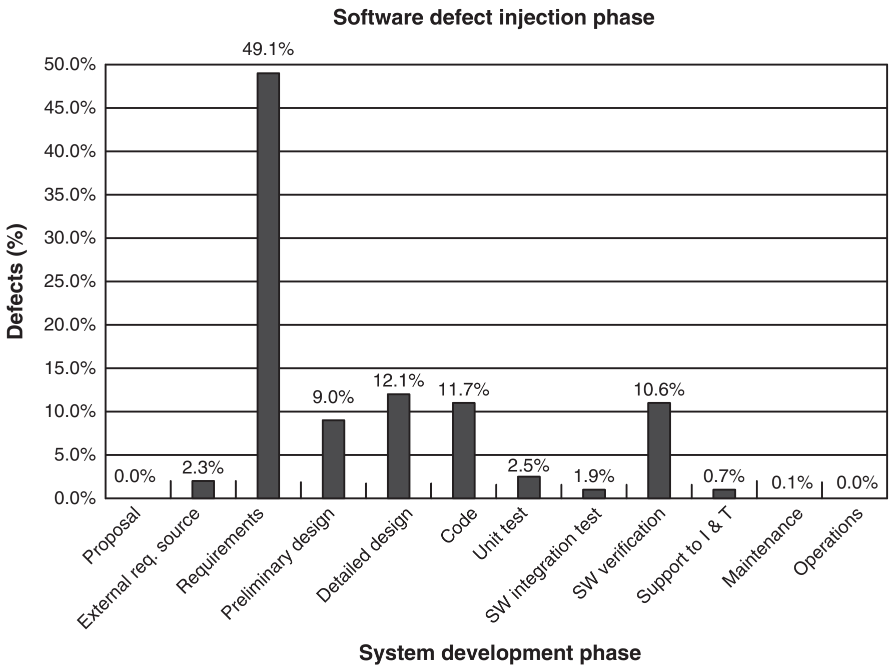


Figure 2.3 Defect injection distribution during the life cycle [SEL 07].

next phase (test and integration), 50 times more in the trial phase, 130 times more in the integration phase, and 100 times more when it is a failure for the client, and has to be repaired during the operational phase of the product.

In fact, Boehm [BOE 01] also published that it would only cost $25, in 2001, to correct a problem identified in a requirements and specifications document. This cost increases to $139 if the problem is discovered during programming. The origin of software defects ended up being the subject of many studies. For example, Figure 2.3 presents the origin of the defects throughout the development life cycle of software in a case study published by Selby and Selby (2007) [SEL 07].

In this case study, we see that approximately 50% of defects occurred during the requirements phase. If these defects are not corrected early on, they will be very expensive to fix during the test and operational phases. As well, it is quite possible that correcting these defects will involve extending the delivery schedule, since all too often the effort involved in correcting the defects was not factored into the initial plan. If, however, the software must be delivered by a set date, it is quite possible that a large number of defects will not be corrected in order to meet the delivery date. Therefore, the client will have to use software containing a large number of defects while awaiting a new version.

In the defense industry, the American company Raytheon carried out a study on the cost of quality [DIO 92, HAL 96]. This study measured the cost of quality over a

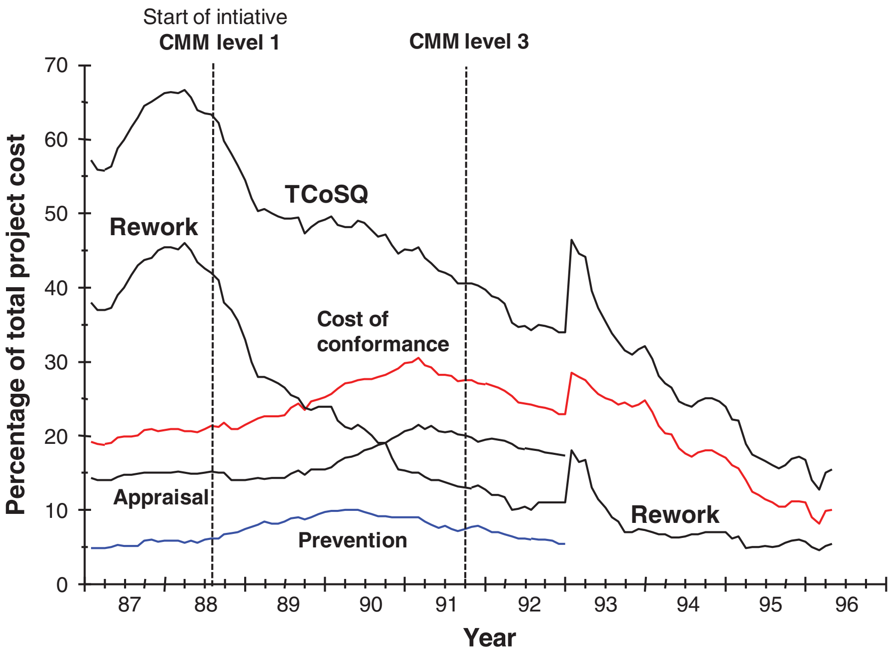


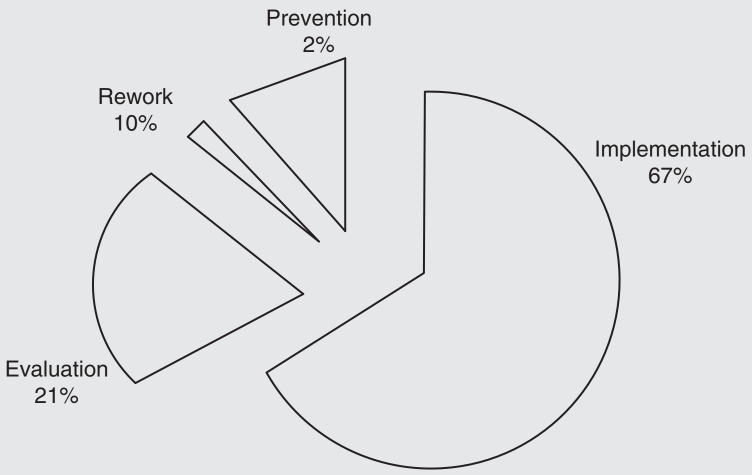
Figure 2.4 Data on software quality costs [HAL 96].

9-year period. Figure 2.4 shows that, at the beginning of the study, the rework cost for this company was evaluated at approximately 41% of the total cost of the projects, at 18% when process maturity had been improved and 11% when the process maturity was at level 3, and then 6% when it was at level 4 maturity. The figure also describes that with prevention costs of less than 10% of the total cost of the projects, the rework costs were reduced from 45% to less than 10% of the total cost of the projects.

The study by Dion concluded that the cost of quality seems to be correlated with the implementation of increasingly mature processes. Another study on quality costs published by Krasner proposes that the rework costs vary between 15% and 25% of the development costs for a level 3 maturity organization (see Table 2.2).

The Case of Bombardier Transport

A project to measure software quality costs was carried out within the software development group at Bombardier Transport located in Quebec. A team, composed of 15 specialized software engineers in this group, was commissioned to develop control software for the subway system in a large American city. They decided to set up data collection to measure the quality costs for this project. The measuring activity took place in four steps: drawing up a list of typical activities related to software quality costs; categorizing these activities (prevention, evaluation, and correcting anomalies); developing and applying weighting rules; and finally, measuring software quality costs. In all, 27 weighting rules were developed, and one weighting rule was assigned to each project task. More than 1121 software activities were analyzed for a project amounting to 88,000 hours of work. The results obtained showed that, for this project, the software quality cost represented 33% of the total project cost. The rework, or anomaly costs were 10%, prevention costs were 2% and evaluation costs were 21% of the total development costs. The following figure illustrates these results.



Budget breakdown for a software project at Bombardier Transport [LAP 12].

Table 2.2 Relationship Between the Process Maturity Characteristic and Rework [KRA 98]

Process maturity	Rework (percent of total development effort)
Immature	≥50%
Project controlled	25% – 50%
Defined organizational process	15% – 25%
Management by fact	5% – 15%
Continuous learning and improvement	≤5%

The Cost of Quality

Over many years, Professor Laporte collected data on the costs of quality. Engineers and managers of multinational organizations and Master of software engineering students working for organizations in the Montreal area provided these data. The following table shows that the average rework cost is approximately 30%.

	Site A American engineers (19)	**Site A American managers (5)**	**Site B European engineers (13)**	**Site C European engineers (14)**	**Site D European engineers (9)**	**Course A 2008 (8)**	**Course B 2008 (14)**	**Course C 2009 (11)**	**Course C 2010 (8)**	**Course E 2011 (15)**	**Course F 2012 (10)**	**Course G 2013 (14)**	**Course H 2014 (11)**	**Course I 2015 (13)**	**Course J 2016 (14)**
Performance costs	41%	44%	34%	31%	34%	29%	43%	45%	45%	34%	40%	44%	36%	37%	40%
Rework costs	30%	26%	23%	41%	34%	28%	29%	30%	25%	32%	31%	25%	29%	27%	27%
Appaisal costs	18%	14%	32%	21%	26%	24%	18%	14%	20%	27%	20%	19%	20%	22%	20%
Prevention costs	11%	16%	11%	8%	7%	14%	10%	11%	10%	8%	9%	12%	15%	14%	13%
Quality *	71	8	23	35	17	43	19	48	35	60	55	72	44	23	

* Number of defects per 1000 lines of source code.

Adapted from Laporte et al. (2014) [LAP 14]

We hope that this section, which discussed the concepts of the cost of quality, has convinced you of the importance of implementing improvements in software processes and quality assurance so as to better understand the cost categories to include in your estimates. It is indeed possible to produce quality software while minimizing rework costs (since rework costs are, if the organization has made the right choices, either losses or profits). The use of the cost of quality concept could affect the level of a company's competitiveness. We have shown that a company that invests in defect prevention can offer a product at a lower cost, with less risk of failures and can gradually make strides ahead of their competitors.

Defect prevention is an activity that has not always been popular with software developers. This activity is wrongly perceived as a waste of time and energy. The following section presents a model that evaluates the efficacy of the defect elimination activity and its associated costs. The objective is to convince you that adding a quality activity is worthwhile, and once you are convinced, you will then be able to convince your bosses.

2.3 QUALITY CULTURE

Tylor [TYL 10] defined human culture as "that complex whole which includes knowledge, belief, art, morals, law, custom, and any other capabilities and habits acquired by man as a member of society." It is culture that guides the behaviors, activities, priorities, and decisions of an individual as well as of an organization.

Wiegers (1996) [WIE 96], in his book "*Creating a Software Engineering Culture*," illustrates the interaction between the software engineering culture of an organization, its software engineers, and its projects (see Figure 2.5).

According to Wiegers, a healthy culture is made up of the following elements:

– The personal commitment of each developer to create quality products by systematically applying effective software engineering practices.

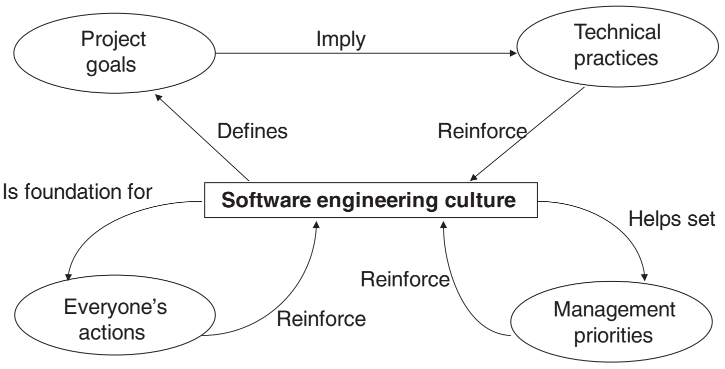


Figure 2.5 Software engineering culture.
Source: Adapted from Wiegers 1996 [WIE 96].

– The commitment to the organization by managers at all levels to provide an environment in which software quality is a fundamental factor of success and allows each developer to carry out this goal.
– The commitment of all team members to constantly improve the processes they use and to always work on improving the products they create.

"The challenge of designing the Boeing 777 was 20% technical and 80% cultural."
John Warner
President (retired), Boeing Computer Systems

As software engineers, why should we be concerned with this aspect? First of all, quality culture cannot be bought. The organization's founders must develop it from the creation of the company. Then, when employees are selected and hired, the founders' corporate culture will slowly begin to change, as illustrated in Figure 2.5. Quality culture cannot be an afterthought; it must be firmly integrated from the outset, and constantly consolidated. The objective of management is to instill a culture that will promote the development of high quality software products and offer them at competitive prices in order to produce income and dividends within an organization where employees are engaged and happy.

The second reason why a software engineer should be interested in the cultural aspects of quality is that effecting change within an organization does not boil down to placing an order with employees. The organization must work to change its culture with the help of change agents. We now know that one of the main stumbling blocks to change within an organization is the organization's culture.

Employees must feel involved and be able to see the benefits of any change. For example, if the change provides no benefit to a worker and has only been made to satisfy a manager's whim, the worker will not be interested in this change. The actions of the employee in our example will not foster the reinforcement of corporate culture and its degree of maturity will not be affected. It is imperative that cultural change is managed in order to obtain the desired results [LAP 98].

You may be asked to "cut corners" regarding the quality of your work or the way in which things are done (comic strip in Figure 2.6). Managers and clients may increase pressure and ask you to skip steps. In the most desperate situations, you may be asked to start programming even before having identified the needs. This old way of thinking about IT still prevails in some companies.

It is not always easy to resist pressure from people who are paying the bills or your salary. In certain cases, you will feel that you have no choice: do what you are being asked to do or leave (see the comic strips in Figures 2.7 and 2.8).

It may be difficult to imagine spending your career in this type of environment. When a quality culture is well established, employees will get involved even in times

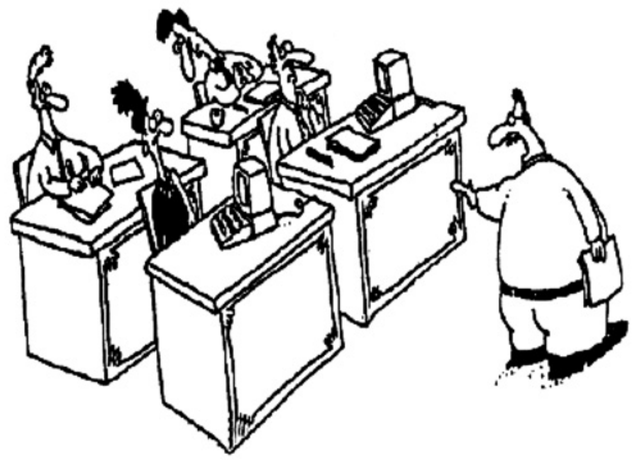


Figure 2.6 Start coding... I'll go and see what the client wants!
Source: Reproduced with permission of CartoonStock ltd.

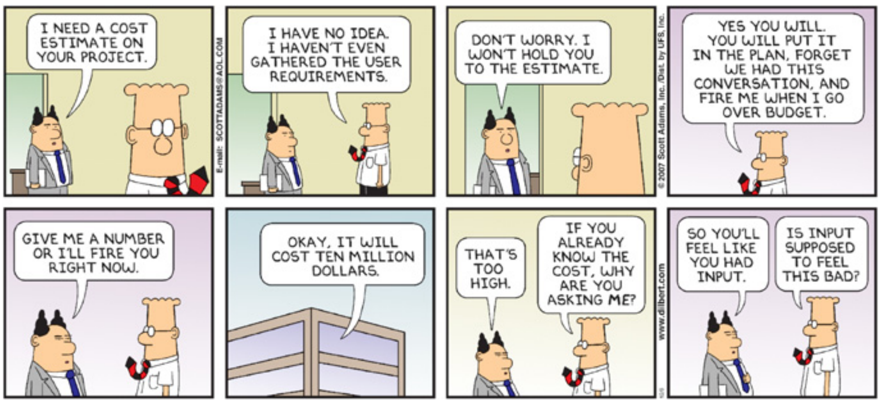

I NEED A COST ESTIMATE ON YOUR PROJECT.
I HAVE NO IDEA. I HAVEN'T EVEN GATHERED THE USER REQUIREMENTS.
DON'T WORRY. I WON'T HOLD YOU TO THE ESTIMATE.
YES YOU WILL. YOU WILL PUT IT IN THE PLAN, FORGET WE HAD THIS CONVERSATION, AND FIRE ME WHEN I GO OVER BUDGET.
GIVE ME A NUMBER OR I'LL FIRE YOU RIGHT NOW.
OKAY, IT WILL COST TEN MILLION DOLLARS.
THAT'S TOO HIGH.
IF YOU ALREADY KNOW THE COST, WHY ARE YOU ASKING ME?
SO YOU'LL FEEL LIKE YOU HAD INPUT.
IS INPUT SUPPOSED TO FEEL THIS BAD?
E-mail: SCOTTADAMS@AOL.COM
© 2007 Scott Adams, Inc./Dist. by UFS, Inc.
www.dilbert.com

Figure 2.7 Dilbert is threatened and must provide an estimate on the fly. DILBERT © 2007 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

of crisis. It is for this reason that software developers and managers must adopt principles that motivate them to stick to their processes even during these difficult periods. Of course, we are always more flexible during a crisis, but not to the point of dropping all quality assurance activities and our good judgment.

In difficult situations, you must never let your boss, a colleague, or your client convince you do to bad work; proceed with integrity and intelligence with your boss and clients.

The client is not always right. Yes, you heard right! However, he probably has a valid point of view and you have a duty to listen to the client's concerns. Having said that, you are the person solely responsible for interpreting his needs and converting them into specifications and reasonable estimates of effort and duration. The

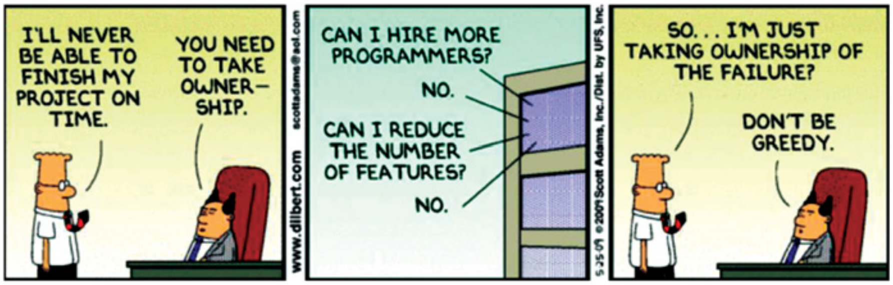

I'LL NEVER BE ABLE TO FINISH MY PROJECT ON TIME.
YOU NEED TO TAKE OWNER-SHIP.
CAN I HIRE MORE PROGRAMMERS?
NO.
CAN I REDUCE THE NUMBER OF FEATURES?
NO.
SO... I'M JUST TAKING OWNERSHIP OF THE FAILURE?
DON'T BE GREEDY.
www.dilbert.com scottadams@aol.com
5-25-09 © 2009 Scott Adams, Inc./Dist. by UFS, Inc.

Figure 2.8 Dilbert tries to negotiate a change in his project. DILBERT © 2009 Scott Adams. Used By permission of UNIVERSAL UCLICK. All rights reserved.

highest added value of a business analyst in a project is to mitigate expectations and find a solution that will meet requirements and is realistic, feasible, and practical. Be aware of overly aggressive salespersons. One of my friends used to say "Nothing is impossible for those who do not have to do it."

Be honest with your clients and make sure that you clearly communicate the limitations and scope of the work that will be done. Wiegers lists fourteen principles to follow to develop a culture that fosters quality (see Table 2.3).

The culture of an organization is a defining factor of successful efforts to improve processes. "Culture" includes a set of shared values and principles that guide

Table 2.3 Cultural Principles in Software Engineering [WIE 96, p. 17]

1.	Never let your boss or client cause you to do poor work.
2.	People must feel that their work is appreciated.
3.	Continuing education is the responsibility of each team member.
4.	Participation of the client is the most critical factor of software quality.
5.	Your greatest challenge is to share the vision of the final product with the client.
6.	Continuous improvement in your software development process is possible and essential.
7.	Software development procedures can help establish a common culture of best practices.
8.	Quality is the number one priority; long-term productivity is a natural consequence of quality.
9.	Ensure that it is a peer, not a client, who finds the defect.
10.	A key to software quality is to repeatedly go through all development steps except coding; coding should only be done once.
11.	Controlling error reports and change requests is essential to quality and maintenance.
12.	If you measure what you do, you can learn to do it better.
13.	Do what seems reasonable; do not base yourself on dogma.
14.	You cannot change everything at the same time. Identify changes that will reap the most benefits, and start to apply them as of next Monday.

behaviors, activities, priorities, and decisions of a group of people working in the same field. When colleagues share beliefs, it is easier to bring about changes to increase group efficiency as well as the chances of survival.

A

A development team was set up. A few of the team members were experts in the field, while others were new. The development tools and processes that guided the project were new and therefore not yet mature. The project manager emphasized team unity to create a learning environment. This allowed the team to practice these new processes. Mistakes were allowed, and thus improvement was continuous. The manager's style combined humor and enthusiasm, which also fostered team spirit. Team activities were carried out to promote individual success as well as the teams' success. The manager also set up a culture of expertise, which meant that each team member had his or her area of expertise and focused all efforts on improving this expertise.

Quality involves an important social aspect in which the level of involvement and collaboration of each member in the company becomes essential. It is imperative to promote, and continuously support, beliefs and practices in terms of quality in order to preserve and enrich this critical aspect of corporate culture.

It is important to understand the context in which the organization is developing software in order to be able to understand why it is using a specific practice and not another [IBE 03]. Indeed, a company that develops and distributes software in a highly regulated sector is quite different from a company that develops its own applications for in-house operations. Moreover, other factors within these organizations may also be influencing practices, such as risk and project scope, mastering business rules, and laws for the sector. By analyzing the situation, the software engineer can better evaluate the changes that should be made to promote the development of a quality culture.

Quality culture must, above all, be expressed by the willingness to cultivate it emanating from the company's upper management and not only from the engineering team who, in a way, plays the role of quality police. In the most extreme cases where accidents occurred, corporate management very often seemed involved in choosing non-quality in order to maximize shareholder profits without having a long-term vision. The next section will provide tools for managing these situations.

2.4 THE FIVE DIMENSIONS OF A SOFTWARE PROJECT

Wiegers developed an approach at Eastman Kodak in New York, in order to better frame project start-up discussions. He believed there are five dimensions that must be managed as part of a software project (see Figure 2.10). These dimensions are not independent. For example, if you add staff members to the project team, this may (but not always) shorten the *deadline*; however, this will increase *costs*. A common compromise is to shorten the deadlines by reducing *features* or reducing *quality*. Compromises that need to be made among these five dimensions are not easily made, but can be illustrated so as to have more realistic discussions and better document these decisions. For each project, you must have an idea of which dimensions have more constraints, and which are able to offset these constraints.

Three roles can be associated with each dimension: (1) a driver, (2) a constraint, and (3) a degree of freedom.

A driver is the specific and major reason for which the project must be carried out. For a product that has competition in the market and must offer new features that are expected by clients, the deadline is the main driver. For a project to update software, the driver could be a specific feature. Defining the project driver and associating it with one of the five dimensions allows us to focus on the status of each dimension.

A constraint is an external factor that is not under the control of the project manager. If you have a set number of resources, the "staff" dimension will be a constraint. Often cost will be a constraint for a project under contract. Quality is also a software constraint for medical devices or helicopters, for example.

Certain dimensions, such as features, may have roles as both drivers and constraints when a feature is non-negotiable. Any dimension that is not a driver or a constraint provides a certain degree of freedom. These dimensions can be adjusted to contribute to achieving an objective under a given constraint. The five dimensions cannot simultaneously be either all drivers or all constraints. Therefore, the priority of each dimension must be negotiated with the client right from the start of the project.

Here are two examples of how to use this negotiation model proposed by Wiegers. A Kiviat diagram with five dimensions allows us to illustrate the model using a graduated 10-point scale where 0 means total constraint and 10 means total freedom.

Figure 2.9 describes a project developed in-house with a set team size and a schedule with little leeway. The team is free to determine which features to implement and the quality level for the first version. Since the cost is linked for the most part to staff expenses, it will differ depending on the amount of resources used. Given the schedule allows for some leeway, deadlines may vary slightly.

Figure 2.10 describes the flexibility diagram for a critical software project in which quality is a constraint and the schedule has a high degree of freedom. The patterns shown in these diagrams provide an idea of the type of project you are dealing with. A project cannot focus all its dimensions at 0. There must be some latitude with certain levels to ensure some project success.

All too often in short discussions we tend to speak only of budget and schedules. It is this attitude that generally leads to overruns and chronic dissatisfaction in our industry. We therefore have to get our administrators and clients accustomed to the reality of not always having the features requested, without defects, delivered quickly by a small team at a low cost.

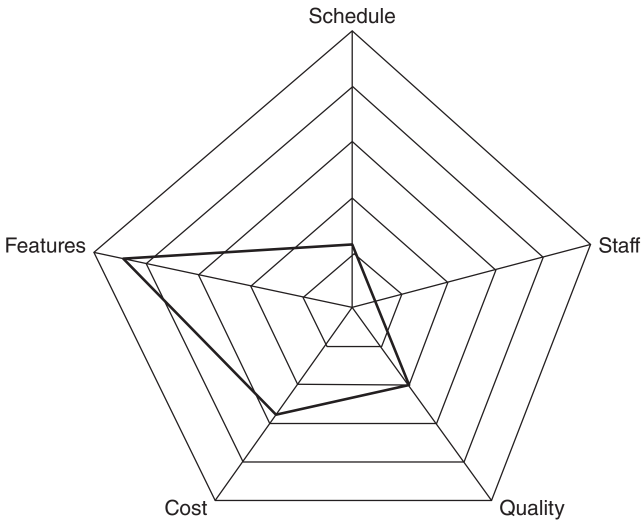


Figure 2.9 Diagram of the flexibility of an in-house project.
Source: Wiegers 1996. [WIE 96, p. 30]. Reproduced with permission of Karl E. Wiegers.

Table 2.4 can be drawn up to summarize the results of negotiation at the start of a project. For each dimension, the table describes the driver, the constraints, and the degree of freedom. The software engineer will attempt to describe the values given to each dimension. The example in Table 2.4 is related to the case in Figure 2.9.

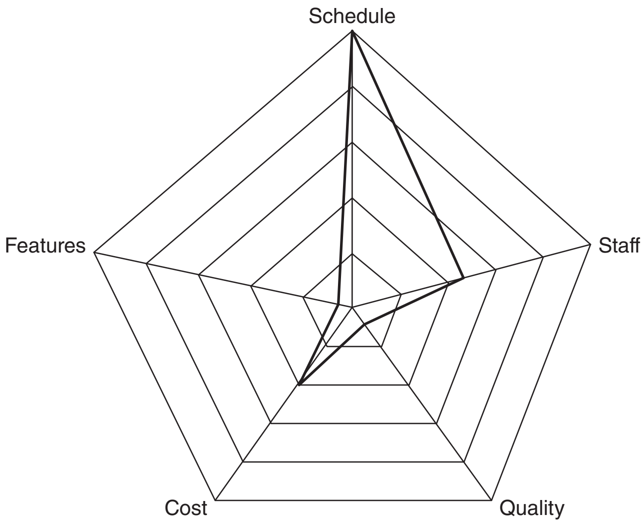


Figure 2.10 Diagram of the flexibility of a critical software project.
Source: Wiegers 1996. [WIE 96]. Reproduced with permission of Karl E. Wiegers.

Table 2.4 Summary Table of Dimensions Negotiated for an In-House Project [WIE 96, p. 32]

Dimension	Driver	Constraint	Degree of freedom
Cost			20% overrun acceptable
Features			60–90% of priority 1 features must be in release 1.0
Quality			Release 1.0 can contain up to five known major defects
Schedule	Release 1.0 must be delivered within 4 months		
Staff		Four people	

A major characteristic of a quality culture and software engineering principles is that expectations and promises are established professionally and in a negotiated manner. Of course, this type of step may at first meet with resistance, but this approach will help you to avoid accepting projects that are not realistic or impossible to carry out in the conditions laid down. Therefore, you should get into the habit of taking a pass on these unrealistic projects and not committing to unavoidable disaster! You can also use other tools that are available to you to help make the right decisions. The next section presents the code of ethics that can be used to convince your clients and superiors of the importance of having a quality culture. Referring to a code of ethics can help the engineer better understand and communicate a quality culture.

2.5 THE SOFTWARE ENGINEERING CODE OF ETHICS

The first draft of the software engineering code of ethics was developed in cooperation with the Institute of Electrical and Electronics Engineers (IEEE) Computer Society and the Association for Computing Machinery (ACM) in 1996. Following this, the draft was widely circulated to elicit comments and suggestions for improvement. The IEEE and the ACM approved the current version in 1998 [GOT 99a; GOT 99b; IEE 99]).

Different perspectives clashed as the code of ethics was being developed. One such confrontation took place in regards to how to approach ethics. The first approach was based on the inherent virtuous nature of humans. This implies that we simply have to indicate the proper direction and people will follow it. Proponents of this approach wanted a code that inspires good behavior with few details. The other approach, based on rights and obligations, requires an outline of all rights and all obligations. The supporters of the latter approach wanted a very detailed code.

Another source of tension stemmed from conflicts in terms of what priority to give to the code's principles. For example, should we favor the employer or the public? This was resolved by indicating that the public good comes before loyalty to the employer or to the profession.

Some people would have preferred to list the standards to be used. It was decided not to list any and instead specify within the code that the currently accepted standards should be used.

On the other hand, everyone agreed that software engineers must be proactive when they become aware of potential problems within the system. Therefore, clauses were added to require engineers to communicate potentially dangerous situations, and to allow software engineers to denounce these types of situations.

To satisfy those who wanted a high-level code and those who wanted a more detailed code, two versions of the code were developed: an abridged version and a complete version.

The code has already been adopted by many organizations. For example, the code is a document that is part of an employee contract. The employee must sign it upon hiring. Over the years, the code became a de facto standard. In some cases, companies may decide not to do business with a company that does not adhere to this code.

"When the devil comes to visit us, he will not have big horns. He will not do any harm, he will not hurt a living being. He will just encourage us to lower our standards and ethics, just a little bit, and the rest will follow … "

Albert Brooks in the film "Broadcast News"

The code describes eight commitments with which peers, the public, and legal organizations can measure the moral behavior of a software engineer (see Table 2.5).

Each commitment, called a principle here, is described in one sentence and supported by a certain number of clauses that include examples and details that help in its interpretation. Software engineers and software developers who adopt the code agree to respect the eight principles of quality and morality.

Following are some examples of the code of ethics: Principle 3 (product) declares that software engineers will ensure that their products and any related changes meet the highest possible professional standards. This principle is supported by 15 clauses. Clause 3.10 states that software engineers will carry out the tests, debugging and reviews of software and the related documents on which they are working.

The code has been translated into nine languages: German, Chinese, Croatian, English, French, Hebrew, Italian, Japanese, and Spanish. Many organizations have publically adopted the code of ethics and a few universities have included it as part of their software engineering study program. The complete version is found in Appendix 1.

Table 2.5 The Eight Principles of the IEEE's Software Engineering Code of Ethics [IEE 99]

Principle	Description
1. The public	Software engineers shall act consistently with the public interest.
2. Client and Employer	Software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest.
3. Product	Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.
4. Judgment	Software engineers shall maintain integrity and independence in their professional judgment.
5. Management	Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. Profession	Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.
7. Colleagues	Software engineers shall be fair to and supportive of their colleagues.
8. Self	Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.

WWW

The software engineer's code of ethics, translated in 9 languages, is available on this site:
http://seeri.etsu.edu/Codes/default.shtm

2.5.1 Abridged Version: Preamble

The abridged version of the code states the main aspirations, whereas the articles in the complete version of the code provide examples and more information as to the manner in which these aspirations must be reflected in the behavior of the software engineer. Together, the abridged and complete versions of the code form a consistent whole. Indeed, without the statement of aspirations, the code risks seeming boring and full of legalese, and without any detailed development included, aspirations could seem abstract and devoid of meaning.

Software engineers who are professionally involved in carrying out analysis, specifications, design, development, tests, and maintenance must respect the principles outlined in the code of ethics.

In accordance with their commitment to the health, safety, and the public good, software engineers must adhere to eight principles presented in Table 2.5.

To use the code professionally, a simple usage procedure is proposed. It is important to understand that the practices are presented in a specific order: from most important to least important. Therefore, you must examine any conflict and go through the code one article at a time to see which article relates to the situation.

As soon as you identify an article in the code of ethics that represents the situation described, you should note that the article has been infringed. Then you should briefly explain why this situation violates the code. Continue in this way, going through all the articles one by one, since the situation may violate more than one article in the code.

Case Study—Confirm [OZ 94]

In 1988, a consortium made up of the Hilton and Marriott hotel chains and of the Budget Rent-A-Car car rental company decided to develop a centralized reservation system to make reservations for airplane tickets, hotel rooms, and car rentals. This project was assigned to AMR Information Services (AMRIS), a company belonging to the American Airlines Corporation. This company had already successfully developed an airplane ticket reservation system used by large companies, that is, the SABRE system.

The partners agreed on a budget not exceeding $55.7 million and a schedule of no more than 45 months. This system had to perform transactions at a cost not exceeding $1.05 per reservation.

At the end of the design phase in September 1989, AMRIS put forth a development plan, which could cost $72.6 million. The cost per reservation was $1.39 instead of $1.05.

In the summer of 1990, two partners were concerned about the system delivery date. Employees working on the CONFIRM project estimated that the project was not respecting deadlines. They had been asked by their supervisors to modify their dates for updates to reflect the original dates of the project.

In February 1991, AMRIS presented a new plan for $92 million. The president stepped down, and, in 1992, more than 20 employees also left. The employees were not happy with project management. They also thought that the managers imposed unrealistic delivery dates and lied about the project status.

In the summer of 1991, a consultant, hired by the developer to evaluate the project, submitted a report. One vice-president was not happy with the consultant's observations. He "buried" the report and fired the consultant. During this period, the Marriott hotel chain was being billed $1 million a month.

In April 1992, AMRIS admitted that it was 2–6 months behind schedule. Hilton was having serious problems with the beta version. Again in April, AMRIS wrote to the partner companies about the managers deliberately concealing a number of major technical and performance problems. It also announced that system development was behind by 15–18 months. As well, eight executives were fired, and 15 employees transferred.

In May 1992, it was announced to the partners that the CONFIRM system did not meet the performance and reliability requirements.

In July 1992, after more than 3.5 years of development and having spent $125 million, the project was abandoned. It was also determined that should the system crash, the database would not be recoverable.

AMRIS settled out of court with the partner companies. This company was said to have been sued for more than $500 million, and that it settled the entire dispute for around $160 million.

2.5.2 The Example of the Code of Ethics of the Ordre des ingénieurs du Québec

Since the code of ethics of the Ordre des ingénieurs du Québec, an association of professional engineers, is quite similar to the code of ethics of the software engineer presented in this chapter, we will illustrate only one of the consequences that an engineer who does not respect this code of ethics may be subject to.

The following text box contains an example of a sanction incurred by an engineer.

Example of a Notice of Permanent Removal from a Society of Professional Engineers

Pursuant to section 180 of the Professional Code, notice is hereby given that on June 4, 2015, the Disciplinary Council of the Society of Software Engineers has declared that Mr. Paul Roberts, at business address 12345 Near Here Street, is guilty of various offenses, including:

In Denver, on or about October 14, 2014, as part of an inspection mandate located at 12345, Client Avenue in Denver, the engineer Paul Roberts issued opinions that were not based on sufficient factual knowledge about software design in his report thus violating Article 4.02 of the Code of ethics of software engineers.

Under this decision, the Commission has ordered Mr. Paul Roberts temporarily removed from the roll of the Society for 6 months. This decision is enforceable from its delivery to the respondent, that is, as of June 11, 2015.

Denver, June 4, 2015
Secretary of the Disciplinary Board
Society of Software Engineers

Another possible consequence is the limitation of the right to practice for a given period, the obligation to undergo training, fines, and the obligation to redo the professional engineering certification.

The following form can be used for the software staff in an organization to demonstrate their commitment to the code of ethics.

Employee Commitment to the Code of Ethics

(Date) … … .

I have read the Software Engineer's Code of Ethics developed by the IEEE and ACM.

Last Name	First Name	Signature

You can also change the text of the form above to modify the level of commitment for the personnel using these statements:

– I agree to adhere to the Software Engineer's Code of Ethics developed by the IEEE and ACM.

– I agree to respect the Software Engineer's Code of Ethics developed by the IEEE and ACM.

A ceremony could be held where all software professionals would swear, in front of their colleagues, to respect the code. The ceremony would end with the signing of the form above. This ceremony could be organized annually to remind everyone, especially new employees, of the importance of respecting this code.

The signed form could also be made available on the organization's intranet.

Remember that engineers from certain countries must first respect the code of ethics imposed by their professional order.

2.5.3 Whistle Blowers

Sometimes, in an organization, a person must make certain situations public. The whistle blower seriously thinks that the interest of the client or public is at stake, and denounces the situation either internally or externally. Internally, he may communicate with the ombudsman or security group, which represents upper management, whereas externally he can contact his professional order or a journalist. When the whistle blower denounces a situation internally, he may be subject to pressure or attacks from his superiors and colleagues. He might also be fired. This fear of reprisals is quite real. However, the engineer is protected; otherwise the practice of whistle blowing would not be encouraged. The professional tribunal ensures that the identity of the whistle blower is protected by law, if he so wishes. As well, there is an article in the code that stipulates the engineer being accused does not have to communicate with the complainant should he be told his name.

2.6 SUCCESS FACTORS

In the following text box, several factors in relation to the culture of an organization which are likely to foster the development of quality software are listed.

Factors that Foster Software Quality

1) Good team spirit.
2) The skills of the members in the organization (it is imperative that managers select competent people in their organization to carry out the different tasks; if not, even with good team spirit, good managers, good communication, and good processes that are correctly applied, if people are not competent, nothing will really be effective).
3) Managers who set a good example.
4) Effective communication between colleagues, managers, and the client.
5) Recognizing and valuing initiatives to improve quality.
6) Highlighting the notion of organizational culture by qualifying it as the key factor in guaranteeing quality.
7) Defining the culture of an organization as being a set of shared values and principles guiding the behaviours, activities, priorities, and decisions of a group of people who work in the same sector.
8) Including the notion of culture in an organization's strategy and ensuring that it is respected by all personnel.
9) In small and medium-sized businesses, there is often a discrepancy between the perception and ideas of quality that the managers have as opposed to the technical teams; therefore, it is up to the software engineer to educate managers and other members of the organization about the implications of quality, while proposing adapted solutions to enable the organization's objectives to be met.
10) According to Wiegers, the level of involvement of the client throughout the project is the factor that will have the most impact on software quality (indeed, for the client to consider having been provided with a quality product, the product must first and foremost accurately meet the client's requirements. If certain requirements were not well understood or were poorly interpreted at the beginning of the project, the costs related to correcting the situation will only increase as the project advances): having

a client representative who is involved throughout the project can only help to clear up any ambiguities as they arise.

11) Clearly defined roles and responsibilities.

12) Allocating the necessary budgets.

In the following text box, some factors related to organizational culture which could adversely affect the development of quality software are listed.

Factors that may Adversely Affect Software Quality

1) Give employees the responsibility but not the authority to take the actions necessary to ensure the project's success.

2) When we "shoot the messenger."

3) When managers hide their heads in the sand rather than solve problems.

4) Lack of knowledge in quality assurance.

5) Unrealistic time frames.

6) A lack of common working methodology between team members.

7) A manager who says yes to everyone.

2.7 FURTHER READING

GOTTERBARN D. How the new software engineering code of ethics affects you, *IEEE Software*, vol. 16, issue 6, November/December 1999, pp. 58–64.

LEVERSON N. G. and TURNER C. S. An investigation of the Therac-25 accidents, *IEEE Computer*, vol. 26, issue 7, July 1993, pp. 18–41.

WIEGERS K. E. Standing on principles, *The Journal of the Quality Assurance Institute*, vol. 11, issue 3, July 1997, pp. 1–8.

2.8 EXERCISES

2.1 Give arguments to convince management that it is necessary and profitable to invest in SQA.

2.2 In the most commonly used model today, the costs of quality consider five perspectives. Describe the quality cost formula and give some examples to illustrate each perspective.

2.3 What is the relationship between software quality and the total cost of detection and prevention according to observations made by researchers?

2.4 What are the benefits to identifying and eliminating defects early on in the software life cycle?

2.5 How can we use the results of the Raytheon study to convince management of the benefits of setting up SQA?

2.6 According to Weigers, what elements make up a healthy quality culture in an organization?

2.7 Name 5 of the 10 cultural principles of software engineering.

2.8 What are the five dimensions of a software project according to Wiegers?

2.9 Draw up the summary table of dimensions negotiated for the project described in Figure 2.10.

2.10 Apply the code of ethics and indicate the two main clauses that have been infringed in the following situations:

a) Peter, a software engineer, developed software for his company. His company develops and sells inventory software. After months of work, he finds he is stuck regarding several parts of the software. His manager, not understanding the complexity of the problem, wants the work finished this week. Peter remembers that a colleague, Francine, had showed him modules from a commercial software package that she developed at another company. After studying this package, Peter directly incorporates some of these modules into his software. However, he did not tell Francine or his boss, and did not mention this in the software documentation. What clauses in the code did Peter violate?

b) A company developed software to manage a nuclear power plant. The software was designed to manage the plant's reactor. While inspecting the code, Marie found major errors in the software. Frank, Marie's boss, said that the software must be delivered to the client this week. Marie knows that the errors will not be corrected in time. What clauses in the code require Marie to take action?

c) In a company with approximately 10 people, you were named by the president to apply the code of ethics. You are head of the development and maintenance team. Describe the steps in your action plan to apply the code of ethics.

2.11 Name factors within an organization that make it easier to apply the code of ethics.

2.12 What clauses in the code of ethics require a software engineer to denounce a potentially dangerous situation?

2.13 What clause in the code of ethics clearly states that a software engineer must not have pirated software in his possession?

2.14 Read the CONFIRM case [OZ 94], which describes the development of a reservation system that became a money pit, and:

a) Identify the clauses in the code that were infringed.

b) What should the AMRIS directors have done differently?

c) What could the AMRIS developers have done differently?

d) How can a consumer know whether software will cause him irreparable damage before it is actually installed on his workstation?

e) If the software causes major errors a few months after being installed, how could the consumer have protected himself?

2.15 Read the case of Therac-25 [LEV 93], a medical device that has caused the death of several patients, and:

a) Identify the clauses in the code that were infringed.

b) What could you have said to a representative of AECL who has made the usual excuses in the software industry in order to avoid responsibility (complexity, testability, and development process)? To back up your answer, explain in concrete terms what the company could have done to reduce the risks inherent in these three characteristics of the software product?

c) You were recently named director of SQA for the new Therac-30 project. This project will reuse Therac-25 technology to produce a more efficient version. What SQA precautions should be taken?

d) You talked with the Software Quality Director for the Therac-25. He shared the lessons learned from the incidents caused by Therac-25. List four of these lessons.

2.16 You have just purchased a new computer. The technician tells you that he installed demo software on your computer. When you start up your computer, you notice that commercial software was installed. What should you do?