# Software Quality is Free

It is not the quality that is expensive, but rather the lack of it.

By Erik Philippus

*The traditional view on software development projects is that there is always a tension between time, money and quality. "You can't have all three", is the widespread software engineering truism, also known as the famous 'devil's triangle'. The common conviction is that a need to cut costs will inevitably result in a lower quality, or that strict ensurance of quality will probably jeopardize the project planning. It rarely happens that quality is deployed explicitly as a means to bring a project back on track. Existing methods of project scheduling focus on time/cost tradeoffs, but do not model quality explicitly.*
*In this article, Telelogic's affiliate Erik Philippus from **IMPROVEMENT** makes a stand against this subsidiary role of software product quality. Based on his own experience as an architect and consultant in a large variety of software development projects, he will argue that it is the lack of the quality that is expensive, not the quality itself.*

*Illustrated by his own experience with Telelogic's software quality assurance tool Logiscope™, he will underline that software product quality is an excellent vehicle for the alignment of the software development process with the business goals. However, selection of the most relevant quality attributes, the utilization of the appropriate suite of quality metrics and the roll-out of a quality improvement program is not something that can be accomplished overnight. To give you a jump start in advanced software quality management, a special bundle is now offered comprising the **IMPROVEMENT** Software Quality Workshop™, a demonstration of selected software metrics and rules, a hands-on inspection of your own source code with Telelogic's Logiscope™ and a one-year licence of Logiscope™ Audit + Rule Checker for a reduced price.*

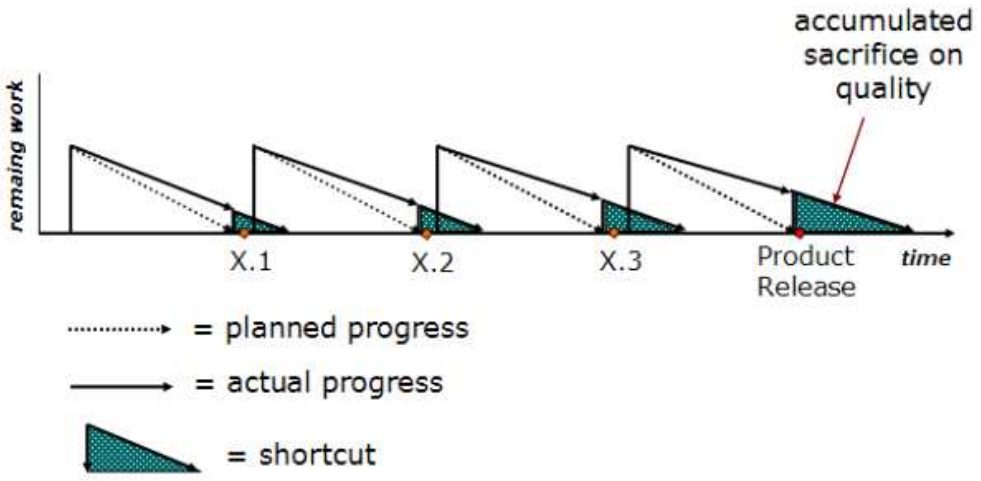
## The impact of software quality on business competitiveness

In recent years, software products have increased in size and complexity, becoming a critical and strategic asset in the organizations' business. In this scenario, to obtain software products with the appropriate level of quality, under the time and resources constraints established in projects, is a challenge. In spite of the fact that standards and methodologies to promote software quality assurance have been continually proposed, lacking software product quality is still responsible for a considerable increase in the total cost of many software-intensive systems. Although many successful software products exist today, the overall lack of attention to software quality has also led to many problematic systems as well as to many software projects that are late, over budget or cancelled. Latent defects and difficulty in maintaining or extending complex code have a dramatic impact on business competitiveness.

Although the majority of the software development cost is spent on quality-related issues such as quality control and maintenance, quality is rarely regarded as a predominant driver for a software development project. The traditional focus during software development is on functionality, only too often at the cost of quality. Of course it is vital that a system has the functionality as agreed-upon by stakeholders and expected by end-users; however in a recent report from Standish Group it is reported that an average of 64% of functions of software systems is never or rarely used. Hence the dominant focus on functionality cannot be justified from a pure business point of view. Other factors must be taken into account to explain the underrated position of quality in most software development projects.

## The unfortunate sacrifice on software quality

First of all, functionality is often more tangible then quality: most features can be simply demonstrated by execution of the software system in front of stakeholders, while it is more difficult to substantiate the surplus value of software product quality. This is further emphasized by most development methods, which hold software developers responsible for delivery of the specified functionality in the first place. Unfortunately, even the modern Agile approach has the tendency to underestimate the value of quality-driven development by its focus on 'working software'.




The pressure on development teams to make it to the next release will also put the focus on functionality. When the deadline for a software release looms ahead, quality is among the first 'victims' in the endeavour to meet the milestone on time. Especially in a multi-disciplinary environment, the need to synchronize the involved disciplines can overrule the software quality requirements in order to maintain the pace. But it is not common practice, however, to reserve time and resources in the next iteration to repair the lost quality. Due to repeatedly rushed work, quality-related issues will pile up each increment. Not surprisingly, this way of working will in most cases yield a system with serious quality flaws at the first release, resulting in expensive maintenance activities or even costly recalls.

## The return on investment in software quality

Hence, the desired software quality must have a crystal-clear relationship with business goals and end-user expectations. Not only the overall level of quality must be appropriate to the product, also the prioritization of quality requirements is a prerequisite for a successful quality-driven development project.

Experience shows that it is of overriding importance to make an upfront selection of the most relevant quality requirements. Only a shortlist of at most five explicit quality attributes, agreed upon by all stakeholders, is perceived to be practical as driver for software development projects. This list must include items that will give the final product distinguishing characteristics, as well as quality aspects that will be very costly (or even impossible) to build-in or enhance afterwards. In most cases, the preservation of critical (existing) quality attributes is equally important to the enhancement of quality attributes that are crucial for competitive position of the product.

It will be clear by now, that the software quality issue has its roots in the requirements phase. Software product quality is often expressed by stakeholders in rather vague terms. It is generally taken to mean that a software product provides value to its users, generates few serious complaints, and of course makes a profit. However, presenting hard facts that underline the *return on investment* in quality is a different story. To prove its worth, software quality must be defined in an explicit and tangible way. When a quality aspect of a software product is not concrete and measurable, it is meaningless as a driver for the software architecture, useless as criterion for the completion of tasks, and powerless as part of formal acceptance tests.

## ISO 9126: A workable definition of software quality

In order to become a mature discipline, software engineering must be able to define its quality lexicon in the most concrete and comprehensive terminology, applicable in a wide variety of domains. Only when explicit quality criteria are part of the overall specification, design and implementation process, it will be possible to predict and guarantee the project result. A solid basis for the measurement of software quality is the ISO/IEC 9126 quality standard. This pragmatic model comprises a collection of quality attributes, metrics and examples of measurement protocols.


An important aspect of this ISO 9126 quality model is the distinction between the perspective of the developers and the stakeholders (especially the end-users of the software product). The internal and external quality is described according to six ´basic´ quality features, namely functionality, reliability, usability, efficiency, maintainability and portability. Furthermore, the perspective of the end user is addressed with quality attributes such as effectiveness, productivity, safety and satisfaction.

Specifying software quality for a product that has still to be developed is difficult for the purchaser or supplier. The purchaser needs to communicate requirements for the product to be developed in an unambiguous way. The supplier needs to be certain that the requirements are clearly understood. Both must be able to assess with confidence whether it is feasible to create a product with the appropriate level of software quality.

Consequently, the ISO 9126 model will serve to eliminate any misunderstanding between purchaser and supplier. This improvement in communication will do away with any rework required as a result of the software product not meeting the purchaser's requirements. Both the time taken to deliver the specified software product and the cost of development will diminish as a result of adherence to the ISO 9126 standard.

Also with existing products or running projects, the ISO 9126 model will help to take advantage of quality-driven engineering. For a quality improvement program to be effective and profitable it is absolutely necessary to establish a named relationship between the problems encountered in real-life cases and explicit quality attributes of the software system under scrutiny. The ISO 9126 model provides the key definitions and measurement protocols that can be shared between stakeholders in order to obtain the buy-in and support to work on quality issues which are directly related to problems in the field.

## The assessment of software quality

With today's complexity and proportions of software systems, it is impossible to meaningful quality measurements without professional tooling. Telelogic's quality assurance tool Logiscope™ measures certain quality aspects of source code that can be correlated with formal product quality attributes defined by the ISO 9126 model. Using sophisticated software metrics, Logiscope™ is able to underline the relationship between certain characteristics of the source code and properties of the final software product. This is useful in predicting whether the ultimate system will possess the required level of quality, as well in analyzing the flaws of existing systems in order to solve legacy problems and/or rectify error-prone practices in the development team.

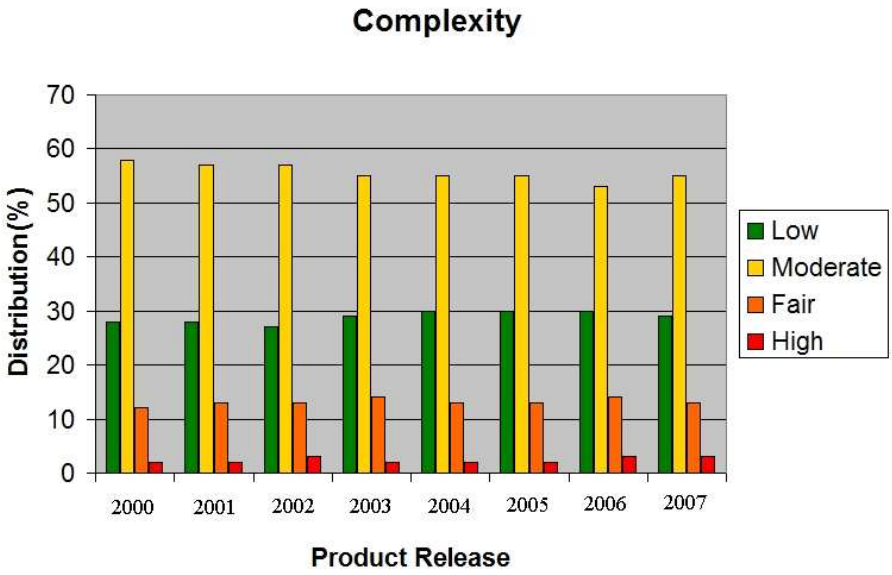
A common challenge for source code analysis tools such as Logiscope™ is that they must be effectively geared to the problems at hand. Churning out of an overwhelming stream of information, cumbersome interpretation of data, production of a long list of false positives, etc. – if these items are on your list, you have the ingredients to spoil a fruitful deployment of any quality tool. Furthermore, it is impossible that a tool will give you all relevant information directly out of the box on a silver plate. Hence, the learning-curve to really understand what is being measured and to fine tune and adapt the tool's features is equally important for a lasting utilization of the tool. And finally, since resources are scarce in most projects, the availability of an expert in both the tool and the domain is a decisive factor for a successful introduction of structural measurement of source code quality.

The selection of the appropriate metrics to obtain critical information about the software system under investigation is a vital activity, as illustrated by the following examples:

*Case description*:
A software system with a high business value significantly resists modification and evolution. It appears to be increasingly difficult to meet new and constantly changing business requirements. Innovation speed has fallen back to a worrying level, and a quality-driven renovation of the current software is urgently needed. Time and resources are precious, so the challenge is to focus on the most relevant quality aspects in order to restore the competitive position of the product. The steady increase in complexity of the application is identified by the software engineers as the root cause of the lack of extendibility of the software. Therefore, to enable identification of software modules as prime candidates for refactoring, a source code analysis with Logiscope™ Audit was proposed.

*Strategy:*
Based on a tentative examination of the 800 KLOC C++ source code, and a series of interviews with software engineers involved, an initial suite of applicable software metrics was put together. Subsequently, the software archive of the eight-year old software system was analyzed and for each applied metric a trend analysis was performed.

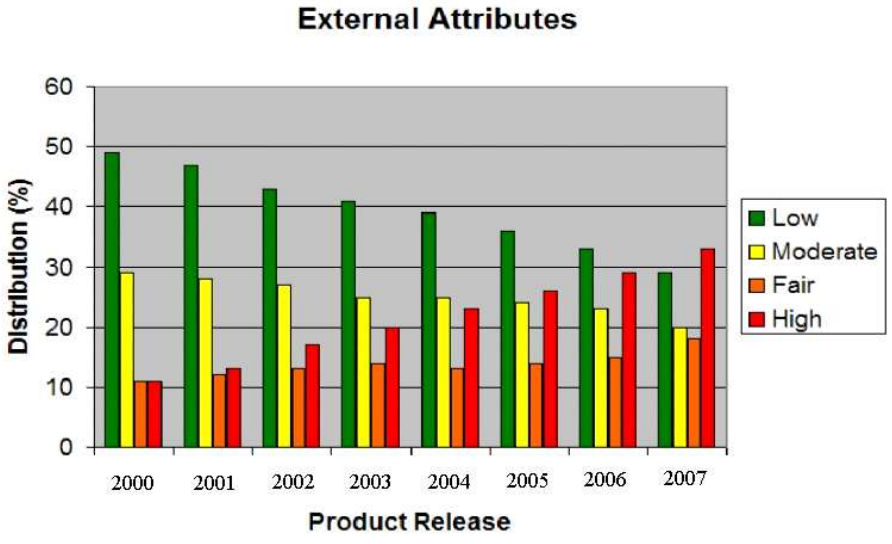
*Results:*
First, the complexity of the software was assessed using a combination of sophisticated complexity metrics. Logiscope™ Audit was configured in such way, that the outcome of the complexity measurements is subdivided in four categories, indicating the distribution of low to high complexity for the individual annual software releases:




The result of the trend analysis clearly shows that over the years, the complexity per unit size of source code has remained invariably at an acceptable level. This observation was quite remarkable, and in variance with the initial hypothesis that a gradual increase in complexity has caused the deterioration of the extendibility of the software system.

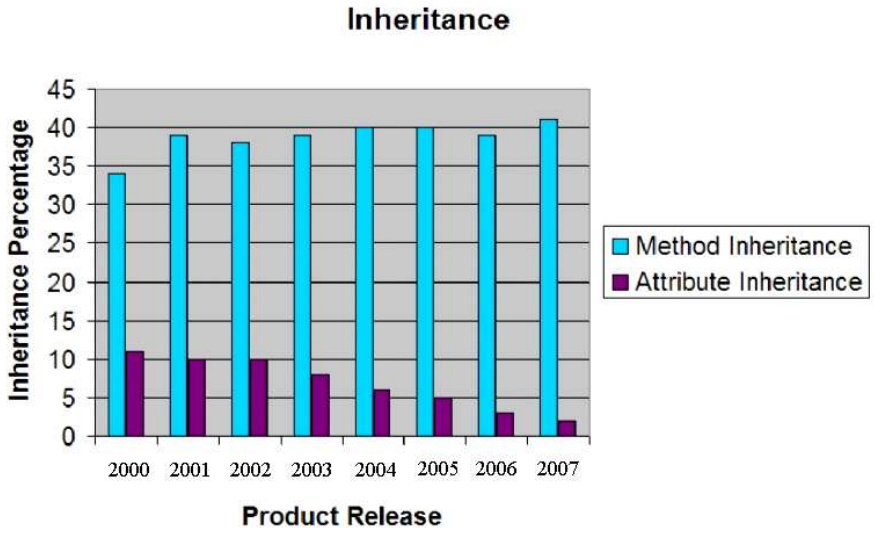
In search for another explanation and a cure for the reduced innovation speed, the next trend analysis concentrated on the usage of external attributes. This metric is directly related to the fundamental principle of data hiding or *encapsulation*. The outcome of this metric indicates to what extend software units are using public information. An excessive dependency on global and public variables will directly translate into a very tight coupled system which is difficult to change.

Again the outcome of the External Attributes measurement is divided in four categories by Logiscope™ Audit, indicating the distribution from low to high dependency on external attributes for the individual annual software releases:



As depicted in the above diagram, the percentage of software units with an unacceptable high level of usage of external attributes has steadily increased over the years. The obvious conclusion was that this excessive binding to public variables indeed caused the increased resistance of the software system against modification and extension.

To further investigate this striking violation of the first paradigm of object-oriented software construction, a suite of object-oriented metrics was used in the analysis of the software archive. Since the encapsulation principle is eminently reflected in the inheritance of methods and attributes from parent objects to derived objects, this aspect was analyzed in the first place. In contrast with the previous measurements, the results are not categorized, but the overall, average value is presented:




It appears that the inheritance of methods is within the acceptable range (20 - 80%), but as depicted in the above diagram, the values for attribute inheritance are very low and also show a downward trend. Normal values for attribute inheritance are between 30 – 60%, so these results are quite exceptional. However, this observation is in line with the previous finding with respect to the excessive use of external attributes. When information is not passed via the obvious mechanism of attribute inheritance, the inevitable alternative is to use global variables.

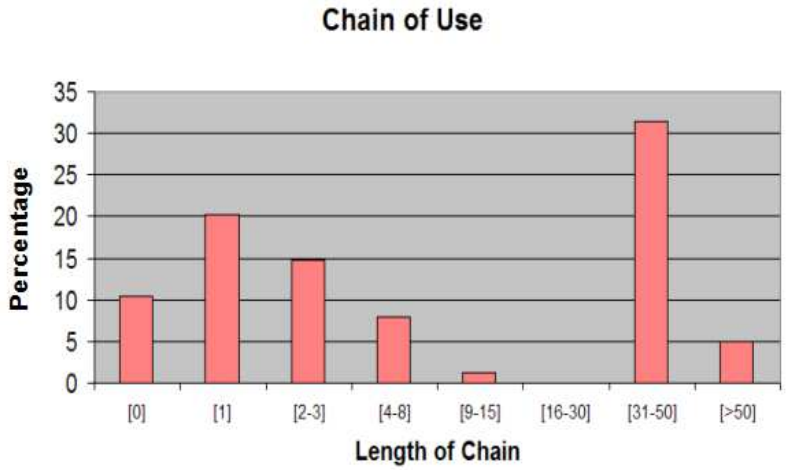
This last measurement was quite helpful in finding the cure for the worsening of innovation speed of the software system. The poor programming practice of using global variables for information sharing has evidently remained unnoticed for a number of years, ultimately leading to a serious legacy problem and loss of market share. A targeted training in object oriented principles featuring encapsulation and inheritance quickly rectified the ineffective programming practice that crept into the development team. At the same time, the metrics used to pinpoint the problem in the first place, were subsequently used to monitor each software release to ensure conformance to the new programming rules.

It is not always needed (or even possible) to conduct a trend analysis on a software archive, as the following example demonstrates:

*Case description*:
A software system has towering maintenance costs and critical functions with unacceptable low performance. Renovation of the core of the current software is urgently needed. It is vital to pinpoint the most serious problems with the current software system in order to preserve the market share. Therefore, a source code analysis of the core software modules with Logiscope™ Audit was performed.

*Strategy:*
Largely based on earlier experience and the reports from software engineers, a suite of advanced software metrics was deployed to do a quick scan of the current version of the 900 KLOC C++ source code. Focus was on the performance issue.

*Results:*
A quite relevant measurement was the 'Chain of Use' metric, which determines the maximum length of the dependency chain for a given class. The histogram for the Chain of Use is given in the following diagram:




A long chain of use is an indication of dispersion of information, for instance caused by erosion of the original partitioning of functionality within the application. A class with a chain of use more then 5 classes may contribute to poor understandability and changeability.

The uneven distribution of the length of chain appears to be quite out of the ordinary: at the left side of the graph a more or less normal histogram is displayed, but at the right side a large contribution of classes with an unusual long chain of use are presented. From this data it is easy to see that a large percentage of classes have a length of chain above 50. In order to carry out some function, a long cascade of other classes needs to be activated. Closer examination revealed that a large number of these classes did hardly anything more than just passing the bucket. A more popular way to describe such a system is using the well-known phrase 'spaghetti code'.

What started as a straightforward universal communication mechanism between layers has unwittingly expanded to an ineffective 'relay station'. Needless to say that the combination of all these service hatches contributed also to the deterioration of the performance. And also in this case the root of the problem was an unnoticed, bad programming habit. The quick scan with Logiscope™ provided the key to solve the legacy problem in a fundamental manner.

## Software Quality Management

Quality-driven software development will help you to prevent or solve many common problems in software development projects, such as late delivery, increasing number of problem reports, legacy issues, instability caused by software faults, persistent and expensive problems with performance or connectivity, etc. However, as underlined by the above examples, a correct selection of the applied metrics is essential to have the full business-related benefits of source code analysis. To In addition, successful deployment of a quality assurance tool such as Telelogic's Logiscope™, requires a combination of domain knowledge, programming experience and familiarity with the tool.

The selection of the most relevant ISO 9126 quality attributes is often a stumbling block. Therefore, **I**mprovemen**T** has developed the *Software Quality Workshop™* to reach agreement between stakeholders about their views, expectations and demands regarding the quality of the software product. The deliverable of this one-day workshop is a shortlist with the five most significant product quality features, including domain-specific definitions and indicators.

After attending the workshop, participants will be familiar with the concept of software product quality and the ISO 9126 model, and they all will utilize a consistent terminology with respect to product quality. The workshop will promote consensus and agreement between disciplines and stakeholders with regard to the most important quality requirements. The workshop is the fundament to introduce or to intensify any quality program.

A logical follow-up of the Software Quality Workshop is the *Software Quality Audit ™*, an in-depth analysis an existing software product, using the advanced code analysis techniques offered by Telelogic's Logiscope™. The result of this assessment is a concise overview of the quality level of selected product features.

Where software quality problems are caused by bad programming habits, one may consider enforcing coding standards and rules by using Logiscope™ Rule Checker to prevent software from entering the archive without proper quality verification.

**I**mprovemen**T** is using Telelogic's *Logiscope™* for static source code analysis and rule-checking, and has accumulated expertise in applying this tool in a variety of software development projects.

To give you a head start in advanced quality management, **I**mprovemen**T** and Telelogic have put together a bundle with the following components:

- Software Quality Workshop™
- Synopsis© of the ISO9126 Quality Model
- Set-up and demonstration of Logiscope™ in your environment
- Quick scan of your source code (C++ or Java) with Logiscope™
- One-year licence of Logiscope™ Audit + Rule Checker.

The above package is currently offered for the introduction price of € 3495,- ex. VAT

For more information, please contact Erik Philippus, senior consultant at **I**mprovemen**T**, or Noud van Mullekom, principal consultant at Telelogic.