Some elementary questions on software quality control

by Jacco Wesselius and Frans Ververs

In view of the growing impact of the application of computer software on human welfare, the quality of software is a topic at issue. In this paper some elementary questions relating to the subject of quality control in software development are addressed. What do we mean when we speak of software quality? What obstacles should be removed in order to obtain quality control? What are directions for research regarding quality control in software development? A central issue in this paper is the notion that complete objectivity in quality assessment cannot be achieved. It will be argued that the consequences of this should not be ignored if we want to make any progress towards the achievement of quality control. The result of our exploration into quality has been that three distinct components of quality can be identified: an objectively assessable component, a subjectively assessable component and a non-assessable component. We have argued that it would be unwise to limit our attention to any single one of these, although only the first is suited to be engineered.

1 Introduction

Almost anyone who is asked about his* feelings on the quality of delivered software products will give evidence of some scepticism. Especially when the post-delivery (maintenance) costs of software are considered, the quality problem in software development comes to the surface.

Before attempting to improve the quality of software products by changing the methods currently employed in software development, it might be fruitful first to address some elementary questions on software quality. The most elementary question we will address is concerned with terminology: what do we mean by 'quality'? We will address this question by confronting several views on quality that can be found in the literature on general quality issues. Starting from the findings of our exploration into the meaning of quality, we will ask the following question: what problems need to be solved before we can reach 'a state of quality control' in software development? Knowing which problems need to be solved, we can ask for the direction of the path that leads to quality control in software development. Founded on the answers to these questions, we will conclude with a summary of directions for future research, related to the achievement of 'quality control in software development'.

* Throughout this paper 'he', 'him' etc. are interchangeable with the female equivalents and *vice versa*.

2 Quality engineering

Our life is pervaded with 'quality'. It is the subject of everyday experience. Yet the exact meaning of 'quality' seems to be somewhat fuzzy; we all know more or less what it means, but when asked for the exact meaning many people will remain silent or will vaguely utter things like 'excellence', 'reliability', 'it cannot be defined', 'everyone knows' etc. Before trying to improve 'software quality', it is essential to obtain a more substantial understanding of the notion. In this Section we will therefore seek a more precise terminology.

The Concise Oxford Dictionary of Current English [1] describes quality as

> **'qua'lĭtў** (-ŏl-) *n.* & *a.* 1. *n.* degree of excellence, relative nature or kind of character, (opp. *quantity; of good high, poor quality, is made in three qualities; quality matters more than quantity,* in the ~ of, as); general excellence (has ~, is excellent). 2. faculty, skill, accomplishment, characteristic trait, mental or moral attribute, (give a taste of one's ~, show what one can do; *has many good qualities, the* DEFECTS *of his qualities, the qualities of a ruler, the quality of inspiring confidence* of *of courage*). 3. (arch.) high rank or social standing (people of ~, the ~, the upper classes). 4. (Logic). (Of proposition) being affirmative or negative. 5. (Of sound, voice, etc.) distinctive character apart from pitch and loudness, timbre. 6. *a.* possessing high degree of excellence; concerned with maintenance of high quality (*quality control*). [ME, f. OF *qualité* f. L *qualitas -tatis* (*qualis* of what kind; see -ITY)]'

The first option is most likely the meaning we think of intuitively; we want to make the very best products, and we strive for excellence. But what do we mean by *the best*? By what measure do we put products in a hierarchy of quality? Does 'The Best Product' even exist, i.e. can we ever create a product *everyone* judges to be the best? Before we can ever create such a product (if ever!), we have to make sure that everyone means the same when speaking of 'quality'. Interpreting 'quality' in accordance with the first dictionary definition therefore would not offer good prospects of improving 'software quality'.

In his well known book 'Quality is free' [2] Crosby puts it like this:

> 'The first erroneous assumption is that quality means goodness, or luxury, or shininess, or weight. The word "quality" is used to signifify the relative worth of things in phrases as "good quality", "bad quality", and that brave new statement "quality of life". "Quality of life" is a cliché because each listener assumes that the speaker means exactly what he or she, the listener, means by the phrase. It is a situation in which individuals talk dreamily about something without even bothering to define it.'

We agree with him, and we will therefore reject the first dictionary definition.

What we are concerned with is the following: we want a product to have certain characteristics. The second dictionary definition therefore is much more useful; it starts identifying the characteristics that are of interest. Since we are not discussing 'good and bad' in a moral or artistic sense, we can narrow down the number of characteristics quite considerably. The characteristics we are concerned with all have to contribute to the usefulness of the product; we are dealing with utensils, and everything has a specific purpose.

> 'All human societies make use of natural and artificial materials and forces to provide *products* which consist of:
> *Goods*: e.g. milk, clothes, houses, vehicles.
> *Services*: e.g. electrical energy, bus rides, health care, education.
>
> An essential requirement of these products is that they meet the needs of those members of society who will actually use them. This concept of *fitness for use* is a universal. It applies to all goods and services, without exception. The popular term for fitness for use is *quality*, and our basic definition becomes: *quality means fitness for use.*'
> [3]

What needs to be added to the above definition is that we are not really interested in fitness for use in general when we are creating a product. We have to develop a product that is fit for a specific purpose. We should say 'quality means fitness for a *specified* use'.

If a product is to be fit for a specified use, it has to have certain vital attributes. Juran and Gryna [3] call these *Quality Characteristics*.

> 'This wide variety of uses means that products must possess multiple elements of fitness for use. Each of these elements is a *quality characteristic* which is the fundamental building block out of which quality is constructed. The quality characteristic is also the means by which we translate the term "fitness for use" into the language of the technologist.
>
> Quality characteristics can be grouped into various species such as:
> *Structural*: e.g. length, frequency, viscosity.
> *Sensory*: e.g. taste, beauty.
> *Time-oriented*: e.g. reliability, maintainability.
> *Commercial*: e.g. warranty.
> *Ethical*: e.g. courtesy, honesty.'

Defining quality in terms of a set of product characteristics necessary for the product to be suited for specified use comes very close to a definition that can be found in the US DOD's standard for Software Quality Programs (DOD-Std-2168) [4]:

> '*Software Quality.* The degree to which *the attributes* of the software enable it to perform its *specified end-item use.*'

From this definition, it would follow that all products that are fit for a specified purpose have equal quality. This seems to be contra-intuitive; something seems to be missing. The missing part is what Juran and Gryna call 'quality of design' [3]:

> 'As a human society acquires affluence, there emerges a spectrum of purchasing power — some members are wealthy, others are poor. In response to this variation in purchasing power, the products of goods and services evolve variations in quality of those goods and services. These variations are popularly called *grades*. The more technical name is quality of design, since the differences in grades are intentional. For example, all automobiles provide the user with the service of transportation. However, the various models differ as to size, comfort, appearance, performance, economy, status conferred, etc. These differences are in turn the result of intended or designed differences in the size, the styling, materials, tolerances, test programs, etc.
>
> In contrast, quality of conformance is the extent to which the goods and services conform to the intent of the design. This extent of conformance is determined by such variables as:
> Choice of processes: i.e. are they able to hold tolerances?
> Training of the supervision and the work force.
> Degree of adherence to the program of inspection, test, audit, etc.
> Motivation for Quality.
>
> A good deal of confusion arises when the word "quality" is used without making clear whether the speaker is referring to quality of design or quality of conformance. Generally, higher quality of design can be attained only at an increase in costs, whereas higher quality of conformance can often be attained with an accompanying reduction in costs.'

So, since the product has been developed for some specified use, we can state a list of characteristics vital to the quality of the product. These characteristics determine a minimum

set of characteristics that any product developed for the specified use should have. The extent to which a product has the characteristics vital to its application (i.e. does it serve its purpose?) is called 'quality of conformance'. There may, however, be characteristics that are not vital to the product's usability but that may cause customers to prefer product A to product B. This difference in customer appreciation of the product stems from differences in the 'quality of design'. *Quality of conformance* thus indicates how well the developer succeeded in developing a product with the required characteristics, whereas *quality of design* indicates how well a customer appreciates the developed product.

In his well known books 'Quality is Free' [2] and 'Quality Without Tears' [5], Crosby stresses the necessity of objective quality judgement and therefore of clinging to quality of conformance. Crosby even defines quality as 'Conformance to requirements'; what is more, conformance to well defined, measurable requirements. The quote [2], from earlier in this paper, continues.

> 'That is precisely the reason we must define quality as "conformance to requirements" if we are to manage it. Thus, those who want to talk about quality of life must define that life in specific terms, such as desirable income, health, pollution control, political programs, and other items that can each be measured. When all criteria are defined and explained, then the measurement of quality of life is possible and practical.
>
> In business the same is true. Requirements must be clearly stated so that they cannot be misunderstood. Measurements are then taken continually to determine conformance to those requirements. The nonconformance detected is the absence of quality. Quality problems become nonconformance problems, and quality becomes definable. All through this book, whenever you see the word "quality" read "conformance to requirements".
>
> If a Cadillac conforms to all the requirements of a Cadillac, then it is a quality car. If a Pinto conforms to all the requirements of a Pinto, then it is a quality car. Luxury or its absence is spelled out in specific requirements, such as carpeting or rubber mats. The next time someone says someone or something has "lousy quality", interrogate that person until you can determine just exactly what he or she means'.

This strict definition of quality requires the statement of a complete set of objectively verifiable requirements. However, to us it is not clear that it is always possible to state such a *complete* set of requirements. It may be possible to make all stated requirements objectively verifiable, but yet there will always be factors that only come to mind after a specific problem has been encountered. From these factors, requirements additional to the ones stated will emerge [6].

> 'The ballistic missile warning systems of the United States (and presumably those of the Soviety Union) regularly give false alarms of incoming attacks. [...] These false alerts stem from causes as varied as natural events, in one case a moonrise, in another a flock of geese [...]
>
> That is why the radar reflections of the rising moon fooled the North American Air Defense system; moons were not among the pre-defined building blocks'.

In our opinion, a product that creates a disaster in the situations that were not considered when specifying the requirements can hardly be called a quality product. Similarly, a product that is being used beyond its specified limits should not create a disaster even though correct behaviour may not be required. If a naval air defence system destroys a civil airplane, something is wrong with the system, even if it happens under conditions that have not been foreseen when requirements were specified. We therefore argue that quality goes beyond the scope of specified conditions; it extends to a system's behaviour under all possible conditions. Since requirements only address a system's behaviour under well defined conditions, they will never capture all aspects relevant to a product's quality.

In addition to the above-mentioned problems regarding complete objectivity in quality assessment, there is yet another problem. Even when the product is 'perfect' at the moment, the needs of the user will change inevitably. A quality product should be adaptable to these changes. Yet, how do we specify the adaptability to changes we do not yet know? Or as Deming [7] puts it:

> 'The problem inherent in attempts to define the quality of a product, almost any product, were stated by my master, Walter A. Shewhart [Walter A. Shewhart, *Economic Control of Quality of Manufactured Product* (Van Nostrand, 1931; American Society for Quality Control, 1980; reprinted by Ceepress, The George Washington University, 1986), Ch. 4]. The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price that the user will pay. This is not easy, and as soon as one feels fairly successful in the endeavour, he finds that the needs of the consumer have changed, competitors have moved in, there are new materials to work with, some better than the old ones, some worse; some cheaper than the old ones, some dearer.'

In addition to this, it is important to realise that the quality of a product will always be judged according to the judge's experience. New requirements originate from new experience. The requirements that are in someone's mind are strongly biased by the kind of relationship he has with the product; different relationships bear different requirements in the mind of the user [7].

> 'What is the quality of a textbook, or of any book that its authors intended to carry a message of some kind? To the printer, quality is determined by style of type, legibility, paper, freedom from typographical error. To the author, and to the readers, quality requires clarity and importance of message. To the publisher, sales are important in order that he may stay in business and publish another book. The reader requires, in addition, some spark of learning, or of entertainment. The quality could be high in the eyes of the printer and high in the eyes of the author, yet low for readers and for publishers.'

So, we have to deal with different views of the product, all introducing their own requirements.

From the above-mentioned aspects of quality, we can see that there is more to quality than 'conformance to requirements'. We have to deal with unexpected situations, unpredictable changes and people who have different relations with the product. It would be unwise to limit quality to 'conformance of requirements', knowing that some important aspects will not be easily captured in this way. When we abandon Crosby's strict definition, we also leave complete objectivity behind; we then accept that we have to deal with the subjectivity of quality and design.

In addition to stated requirements, we now also have to deal with requirements that are in people's mind but that may never have been stated. A definition of 'quality' therefore should include 'user expectations'. In his book 'Total quality control' [8], Feigenbaum defines quality in terms of user expectation.

> 'Product and service quality can be defined as:
> The total composite product and service characteristics of marketing, engineering, manufacture, and maintenance through which the product and service *in use will meet the expectations of the customer*'.

An analogous definition is given by ISO in ISO 8402-1986 [9].

> '**Quality**: The totality of features and characteristics of a product of service, that bear on its ability to satisfy *stated* or *implied* needs.'

In the latter definition not even the user's need is fixed. Quality is not only judged according to stated but also to *implied* needs. The consequences thereof can be grand; a product that provides for the stated needs perfectly may yet be of 'low quality' because some implied need has not been provided for. This may give rise to a distinct sense of unfairness; 'we did everything we were told to, yet are accused of doing a lousy job'. Creating quality turns out to be more than doing exactly what we have been told to; creating quality plainly means '**pleasing the customer**' (i.e. all future users of the product, including marketing people, maintenance people, end-users, etc.).

We have now arrived at a point where the subjective component of quality has become dominant. The way Peters describes quality in 'A Passion for Excellence' [10] and 'Thriving on Chaos' [11] addresses the subjectivity of quality. To Peters, quality is nothing more or less than 'customer perception'. This can create an even stronger sense of unfairness [11].

> 'Caution: The Customer's Perception of Quality Can Be Perverse
>
> I have owned a GMC truck for over a year. Nothing major has gone wrong with it — no dropped transmission or oil leaks. It's worse than that: About eight little things have gone wrong, such as the failure of a light which indicates whether or not the truck is in four-wheel drive and a loose, rattling latch on the glove box.
>
> The problem is that even collectively these irritating problems aren't enough to warrant the inconvenience of a visit to the dealer. Therefore, and here's the rub, each time I get into the truck, it's as if a brightly lit map of GM's failure to attend to details flashes before my eyes.
>
> Frankly, in terms of my perceptions, GM would have been better off if the transmission had failed. I would have gotten it fixed immediately, it would have been done with, and my memory of it would have dimmed.'

This quotation illustrates the unpredictable and subjective component of quality. It also illustrates that the customer is the ultimate judge of quality. Despite the manufacturer's effort to develop a product that has 'enough quality' in his eyes, a customer may be displeased with the product. As the quotation illustrates, this may even be the case when the product is suited for specified use and possibly well engineered. Using a catch-phrase, we could conclude that quality control has to be 'customer-oriented'.

The conclusion of our little exploration of 'quality' is that we can identify three distinct components of 'quality'.

☐ **An objectively assessable component:** does the product have all the characteristics stated in the requirements specification? These required characteristics stem from two sources:

- the specified use of the product is intended for;
- personal preferences of the customer that have been entered into the set of stated requirements.

☐ **A subjectively assessable component:** do the characteristics of the product comply with the preferences and expectations of the individual customer? In case we are capable of creating a product that conforms with the stated requirements, this will raise the question of whether the set of stated requirements is correct (i.e. conform the customer's expectations) and sufficiently complete?

☐ **A non-assessable component:** does the system behave according to our expectations in situations that have not been foreseen? Can the product be adapted to emerging needs that we do not yet know?

Concerning our aim to engineer quality (i.e. using scientifically sound methods and techniques to achieve the desired quality in a product) only the first component seems to offer good prospects, for objectivity is an important prerequisite to engineering. The latter two components enter into the area of 'fuzzy-requirements', where assessing compliance objectively is impossible. To enhance the scope of engineering quality the first component should be preponderant. We therefore need to

- specify required characteristics in a way that enables objective assessment of the product;
- move as many characteristics as possible from the second and third component of quality to the objectively assessable component. Regarding the second component, this means that we should abstract from the individual customer and use a larger customer population to determine 'average appreciation' of the product. This way the component can be made objective. If the product is to be tailored to the needs of just a small group of customers, this approach may, however, not offer a real solution.
- offer means to ensure that the characteristics entered

into the set of required characteristics comply with the expectations the customer has regarding a product that is suited for the specified use.

In short, we should try to get as clear a picture as possible of the image the customer has of 'the ideal product for the specified use' to enable objective assessment of the product's quality. In addition to all this, it is important that we can implement a product to comply with the stated requirements, using no more than the available resources. The latter relies on the availability of valid software engineering techniques. Software engineering techniques offering means to ensure quality of conformance should be enhanced and made practically applicable.

3 What separates us from Engineering Quality?

Before we can engineer quality, the problems that need to be solved are how do we obtain a sufficiently complete, correct and consistent image of the customer's expectations and wishes concerning the product we are to develop; and how do we apply software engineering techniques to develop a product that complies with the thus formed image? Quality engineering therefore consists of

- obtaining a requirements specification that can serve as the basis of a potentially successful software development process (Quality of Design); this means fighting incompleteness and subjectivity.
- offering a framework for effectively applying useful software engineering concepts and techniques in order to comply with the stated requirements (Quality of Conformance).

It should be noted that offering a framework for applying techniques does not bring quality. The value of the framework depends, for a large part, on the usefulness of the techniques the framework encompasses. The success of research efforts on software quality engineering therefore depends, for a large part, on the usefulness of current and future software engineering concepts and techniques. Research into software quality engineering and into general aspects of software engineering are therefore strongly intertwined.

In addition to the problems of subjectivity and incompleteness, we have to deal with some other problems. Solving these problems would be a major contribution to the achievement of a 'state of quality control' in software development.

3.1 Quality is indivisible

The engineering approach is (in addition to its striving for objectivity) characterised by its reductionistic approach. Solving a problem is accomplished by dividing it into smaller problems that can subsequently be solved more easily. If we make the right division, and if we assemble the solutions to the subproblems properly, solving the smaller problems solves our original problem.

If we intend to solve quality problems this way, we are bound to fail. The quality of a product cannot be determined from the quality of its parts. The quality of a chair cannot be defined by the quality of its legs, its seat, its back and its elbow rests. Putting together quality parts may result in a product that is completely unsatisfactory. Of course, when we want to create a quality chair, it is wise to create quality legs, a quality seat, a quality back and quality rests, but it is no assurance for the quality of the chair. The quality of the chair can only be judged after all the parts have been put together, when someone sits on it. Someone can judge the quality of the chair by sitting on it without examining the quality of its legs.

Judging the quality of the individual parts before assembling a product is a means of controlling quality, but it can never be a means of assuring the quality of the resulting product. However, creating quality parts is a prerequisite to creating a quality product.

3.2 Software lacks tolerances

Dealing with a software system means dealing with a system with a discrete nature. This means that tolerances are far less applicable than in engineering disciplines dealing with the behaviour of continuous systems. Tolerances are only applicable regarding the non-discrete characteristics of software, such as performance, availability etc.

3.3 The introduction of new functions

Very often the introduction of an automated system has a large impact on the organisation into which it is being introduced. This means that changes will take place after the product is introduced; existing needs may be changed, and new needs may emerge. This means that, even if the product was intentionally developed to automate an existing function, newly introduced functions may have to be part of the product's functionality. The contents of these new functions will hardly ever be clearly defined.

3.4 No generally accepted theory

In many engineering disciplines a well founded and generally accepted theory exists. This theory constitutes the foundation of the engineering activities that take place. By offering a common terminology, it allows communication and measurement.

In software engineering, new terms and concepts are being introduced at a high rate. A stable and accepted terminology is lacking, and furthermore, a theory of software engineering on which new techniques could be founded is not at hand. Because of this, it is almost impossible to assess the value of newly proposed techniques. This hinders acceptance of valuable techniques and allows useless techniques to be entered into software engineering practice.

3.5 No well calibrated sense of quality

In direct line with the previous problem lies the problem of lacking a feeling of good quality. In many disciplines, it may be true that people lack the theoretical background to perform their job most effectively; nevertheless, in most cases, they know what they are doing. They know when they are doing a good job; they know what it means to create 'a quality product'. In most disciplines, people have a sense of craftsmanship that makes everyone look in the same direction when it comes to the creation of a quality product.

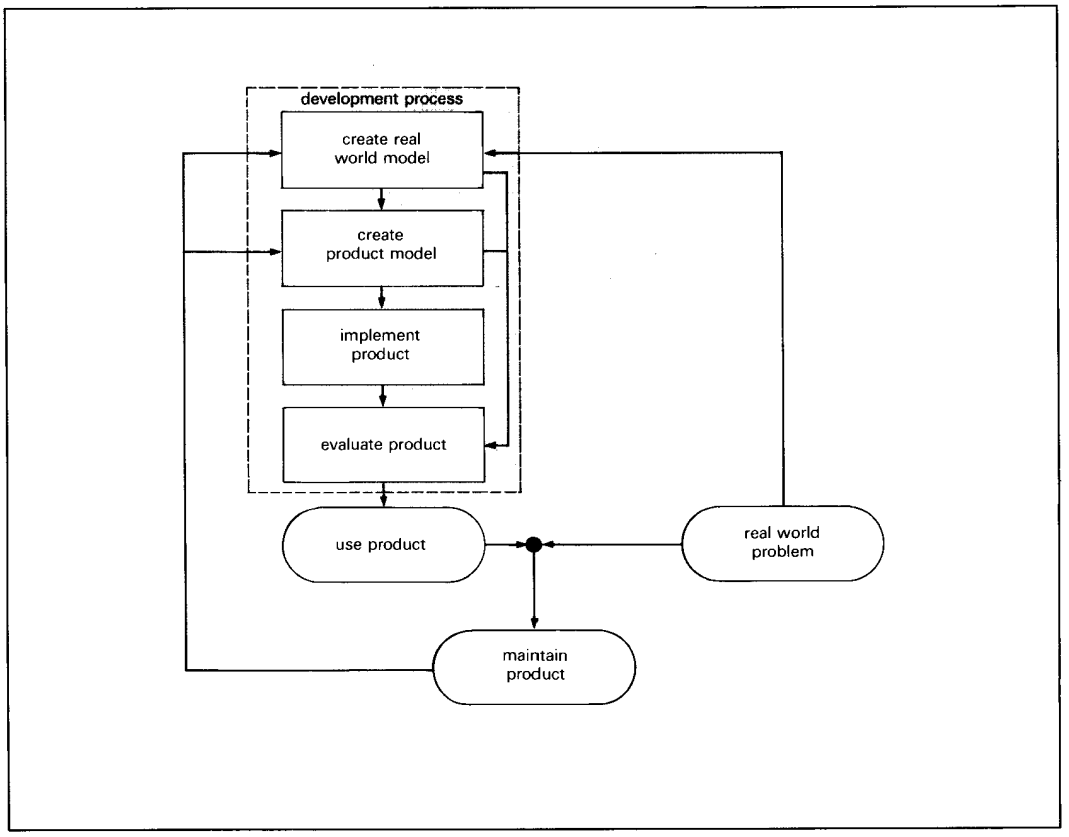

Fig. 1 The ideal development process

In software engineering, there seems to be no well developed sense of quality. It proves extremely difficult to discuss the quality of a product with a colleague. We all seem to have our own preferences which are hardly ever supported by relevant data. This means that we do not know which decisions to make when it comes to the development of quality products.

3.6 Means of navigation

It has been generally accepted that for a traveller it is important to know where he is, where he wants to go to and what roads are available. The analogue notion has been accepted in almost any engineering discipline. Data are being collected to determine the effectiveness of the currently employed techniques. Based on these data, aspects that need improvement can be identified and new techniques can be proposed and evaluated.

In software engineering, however, there are generally no means to determine the current state and compare it with the state we were in yesterday or the state we want to be in tomorrow. We often do not know how to measure the effectiveness of the software development process. This means we cannot track our progress and we cannot plan our future. We should be collecting statistics to determine the impact of the measures we have taken. Without them, the value of efforts can only be guessed, and process control is a suburb of Utopia.

The success story of Hewlett-Packard in the application of process measurement [12] shows that it is possible (although difficult) to carry out measurement in software development. It also shows the benefits of doing measurement.

From this enumeration we conclude that a general improvement in education and the development of a well formulated and well founded theory of software engineering, in general, is needed. In addition to the efforts spent in the achievement of quality control mentioned at the beginning of this Section, effort needs to be spent in the achievement of general improvements in these areas.

4 Towards Quality Engineering

If a complete set of required characteristics of a software product could be specified in advance, it would be feasible to employ some 'ideal development process' that would lead us in a straight line from the specification of clear requirements to a quality product. As we have seen, this approach is not realistic. The fact that completeness in requirements specification, and consequently complete objectivity in quality assessment, is beyond our reach, should, however, not incline us to abandon specifying at all. Neither should it induce a self-indulgent sense of self-pity, rather it should be a challenge; it should trigger a process of continuous improvement. Because the customer and the user are the ultimate judges of quality, we have to develop methods that

stimulate customer and user participation in order to get the specification of their needs as complete as possible. This means we have to adapt the 'ideal development process' to make it practically applicable. In this Section, we will sketch several ways in which the 'ideal development process' could be modified to make it more applicable.

An effort to be performed in parallel with the rearrangement of the development process should be trying to objectify the assessment of quality, by developing techniques that capture as much of the required characteristics as possible. The road towards quality control thus leads in two directions.

- Rearranging 'the ideal development process' in order to accommodate the fact that quality is not completely objective.
- Enlarging the objective component to make 'the ideal development process' more applicable.

Because quality is a 'human thing', we should remember that people, rather than methods and techniques, create quality or junk. It should therefore be clear that no process model nor any technique in the world can create quality products, unless people are motivated do so. Quality can therefore only be created when the people who create products are fully aware of the need for quality and are given means to create quality by their management. Project management should therefore be 'obsessed with quality' to make it clear to the people creating products that they back them in their striving for quality. Management should not try to enforce quality by enforcing methods, techniques and standards; neither should it support quality only passively. Management should participate in the creation of quality actively.

In the next Subsection, we describe what we called 'the ideal development process' and the changes required to adapt it to both the incompleteness in requirements specification and the subjectivity inherent to quality assessment. We also address the topic of enlarging the objective component of quality to make application of 'the ideal development process' more feasible.

4.1 The development process

Above we referred to 'the ideal development process'. By 'ideal' we meant that this development process could be used if quality could be assessed objectively, by comparing the product's characteristics with a complete set of required characteristics. The model would look like the model sketched in Fig. 1. It splits the development process into four stages.

☐ **Create a model of the real world.** This means try to capture the problem completely. The problem, and the environment in which the problem exists, should be characterised completely.

☐ **Create a model of the product.** This means state a complete set of required characteristics which the product should have. The validity of the stated requirements can be proven, by relating them to the problem and its environment which have been characterised in the previous step.

☐ **Implement the product.** Founded on the complete set of required product characteristics, we could apply software engineering techniques to realise a product.

☐ **Evaluate the product.** After a product has been developed, its validity can be assessed, by comparing the characteristics of the product with the characteristics that have been stated during the first and second steps of the development process. Since the required characteristics have been stated completely, this evaluation would be a complete evaluation of the product's quality. If the evaluation shows all required characteristics to be present, the product has the required quality and is therefore fit for use.

Because post-development activities (maintenance) will be performed, the characteristics needed in order for these activities to be performed efficiently and effectively have to be included in the set of product requirements too.

The applicability of this development process model is limited by the potential discrepancy between the developer's perception of the problem he is to solve and the real problem. Rearranging the development process therefore may be necessary in order to connect the development process to reality more tightly.

A process model describes the steps to be taken when developing a product, together with the moments when the quality of the product developed thus far should be assessed. Assessing the quality of the product takes place in two different ways.

- **Verification:** assesses the quality of the step from which the product resulted. This means we compare the characteristics of the product with the statement of required characteristics that served as input of the development step.
- **Validation:** assesses the quality of the product itself. The characteristics of the product are related to the problem it was intended to solve. This means the product is judged in relation to the real world.

Verification judges the product, assuming the statement of required characteristics is complete and correct. Validation judges the product in its real environment and is therefore a means to assess the completeness of the statement of required characteristcs. If great trust can be placed in the completeness and correctness of this statement, validation should only be important in the very first stages of the development process. Knowing that complete achievement of completeness and correctness is an illusion, however, we should make sure that validation gets sufficient attention in the process model we apply. From this point of view, the following well known models of the software development process will be analysed.

- ☐ the stagewise or waterfall model;
- ☐ a rapid prototyping model;
- ☐ a model of evolutionary delivery;
- ☐ incremental development;
- ☐ the spiral model.

4.1.1 The stagewise or waterfall model: the waterfall model [13] resembles 'the ideal process model'. This model (Fig. 2) is characterised by splitting the development process in strictly separated stages and by a forward coupling between the various stages. It is apparent that using this model stresses the necessity of assuring the correctness and completeness of the output of every singular stage. Letting the next stage take place, when the correctness and

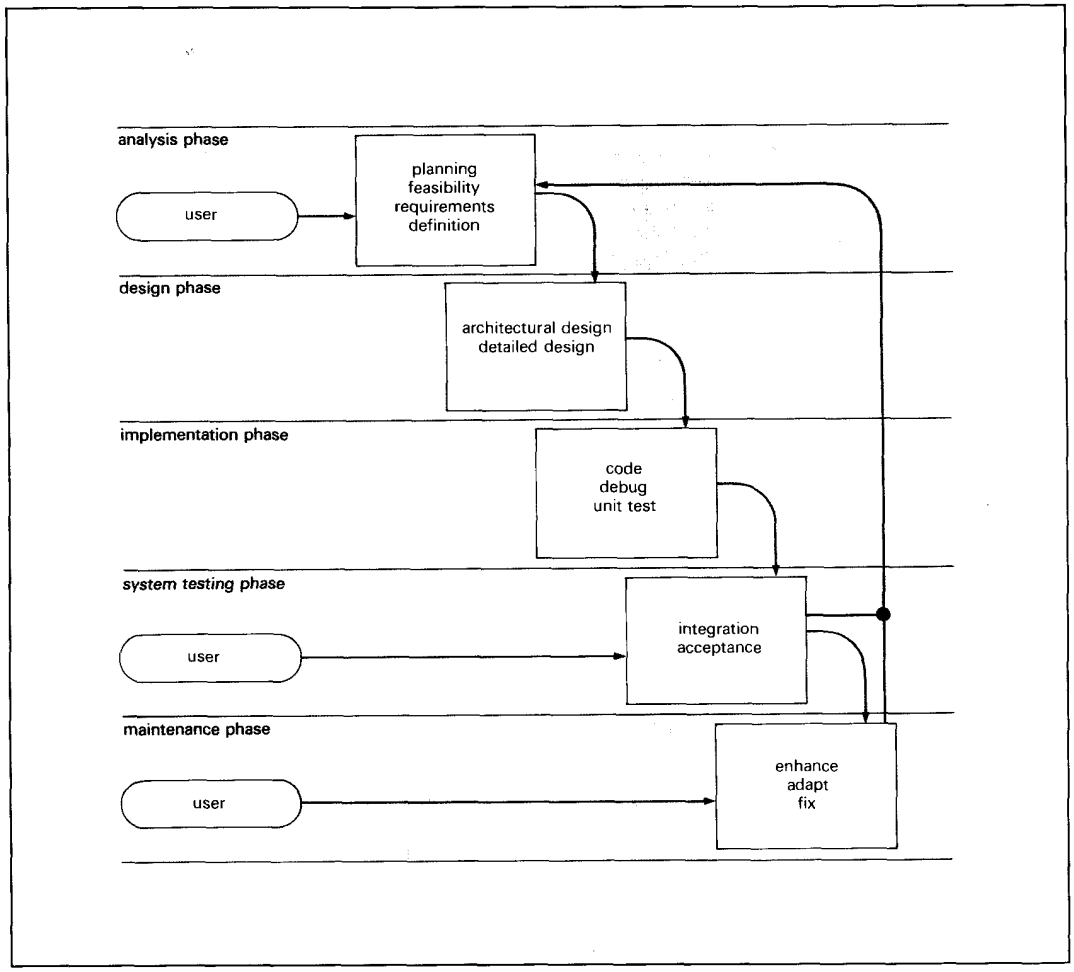

Fig. 2 The stagewise or waterfall lifecycle model

completeness of its input are not absolutely certain, might cause expensive problems later on.

Because imperfection turns out to be unavoidable, backward coupling has been added to the model. When mistakes are discovered during the performance of a certain stage, it is possible to interrupt this stage to rework one of the previous stages. Backward coupling is not introduced for its positive contribution to quality, but rather for its necessity to compensate for human imperfection.

Since rework is very expensive, specifying the requirements completely before any further development activities is necessary when this model is being used. However, as we have seen before, it is impossible to state the product requirements completely and using this model turns out to be somewhat problematical.

The 'user feedback loop' is very large in this model. Only when validation takes place, can user problems can be spotted. Quality determination thus takes place at the very end of the project, and this model thus lacks the possibility of real quality control.

4.1.2 A rapid prototyping model: applying the waterfall model offers very limited possibilities for achieving quality control. Rapid prototyping (Fig. 3) is one way to restructure the first stage of the development process while shortening the vital feedback loop. The first stage of a development process then consists of building one or more so-called 'prototypes'. When prototypes are built, the user feedback loop becomes much shorter. This lets us gain some control over quality in the first stage of the development process.

After the first stage has been finished, however, a waterfall-like development process takes place. The output of the requirements definition stage thus once again has to be correct and complete. It is clear that, after having built several prototypes, our understanding of the user's requirements should have increased but, as we have seen in Section 2, complete specification is an illusion. Not even prototyping can make that change. After completion of the requirements definition stage, control of quality is once again lost.

Since rapid prototyping changes only the first stage of the waterfall model, it should not be considered to be a different process model. It thus shortens the vital feedback loop for one of the process stages, but it does not allow control of the quality during the complete development process.

4.1.3 A model of evolutionary delivery: since user feedback is most valuable when based on hands-on experience with a working product, evolutionary delivery emphasises the need for working products. Every step of the development process must be completed with the delivery of a working system which can be used by the customer.

Since *frequent* feedback is required, the development of evolving new products should not take very long. A reasonable step size is approximately five percent of the total development effort. The development process therefore will be split into at least 20 steps,* which all cost less than approximately five percent of the available resources. The product resulting from a step should be usable to the customer and should be an improvement compared to the system used before completing this development step. If the resulting product turns out not to be an improvement, this is detected quickly and only, at most, five percent of the available resources have been spent. These are not wasted, however, since much insight into the requirements of the customer will have been gained.

Another important feature of evolutionary delivery is that it enables frequent measurement of a working system. This makes it possible to measure progress relative to product characteristics and requirements (i.e. if these have been stated in a way that enables measurement). As we have noted in Section 3, knowing the progress we have made and knowing how far we have yet to go are prerequisites of process control.

Evolutionary delivery relies on the assumption that any step in the software development process can be started from an existing system, which can be replaced by a slightly modified system that suits the customer's demands better. There are three aspects to this assumption.

- A working system exists (not necessarily an automated one);

> 'Many people wrongly assume that the first step cannot be delivered until a rather large, critical minimum new system is designed and built. This is not true if you start by using the existing system — however much you dislike it [14]'

- Small changes can bring improvement (even up to the final level of product completion).
- The people using the system can adapt to a large number of small changes to their working environment, without a significant decrease of performance.

To us, it is not absolutely clear that these assumptions are always valid.

Evolutionary delivery offers a grip on the subjectivity in quality assessment by means of very frequent feedback from the customer and the user. The main difference between rapid prototyping and evolutionary delivery lies in

* This number has been taken from Gilb [14].

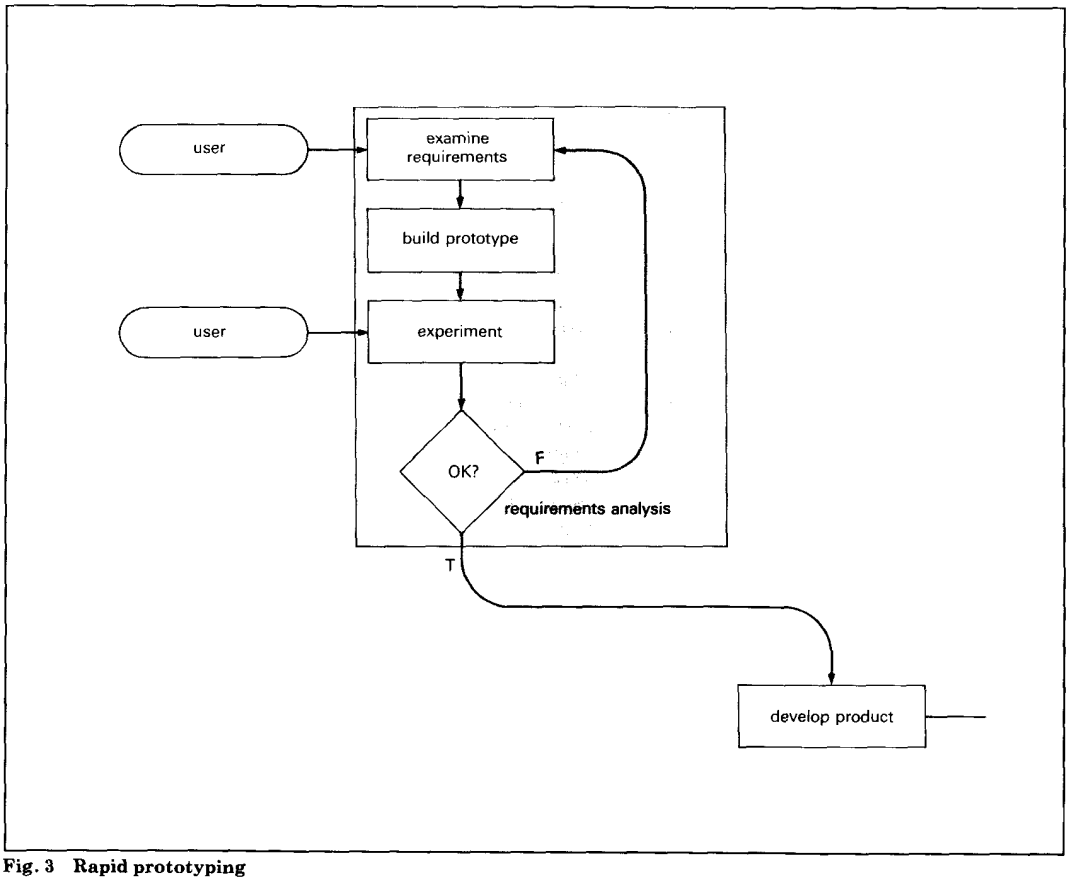

Fig. 3 Rapid prototyping

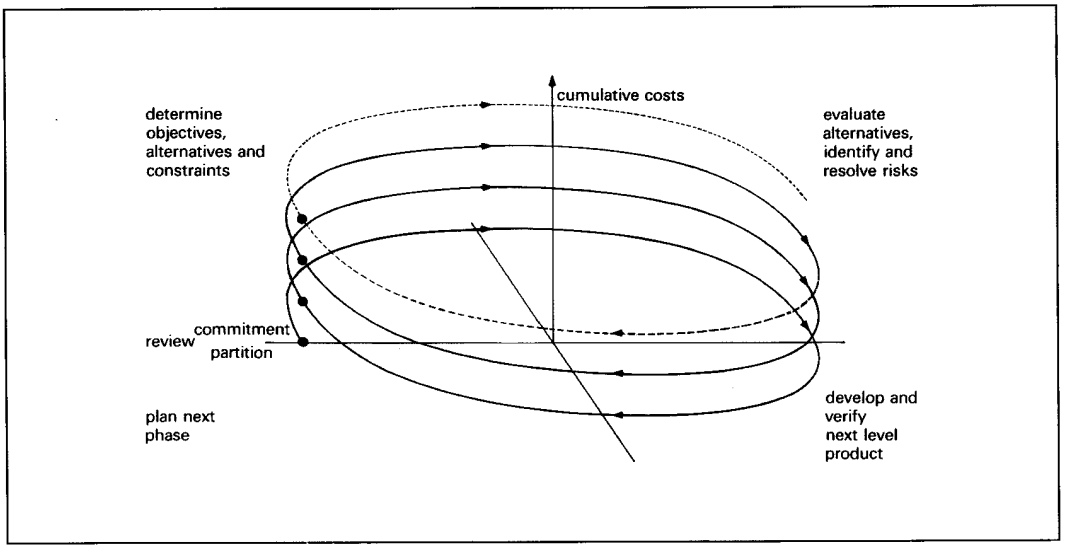

Fig. 4 The spiral model

the fact that the latter allows frequent feedback during all stages of the development process, whereas the former only offers feedback in the first stage of the development process.

4.1.4 Incremental development: incremental development has one characteristic in common with evolutionary delivery; it is beneficial to deliver a product that is usable to the customer as quickly as possible. However, the way incremental development achieves this is less general. The deliverables of incremental development are intended to be final constituents of the final product. Each delivery should enhance the system by adding new functionality. In evolutionary delivery the deliverables are not intended to be final. The final product is not built by adding new functionality with each delivery but by slightly changing existing system characteristics. Incremental delivery emphasises the functionality of the system, whereas evolutionary delivery emphasises the attributes (the qualities) of a system.

Incremental delivery thus offers prospects for validation by providing feedback based on *de facto* experience with a part of the system, but its scope is less general than the scope of evolutionary delivery. It may, however, be less of a problem to introduce incremental development in the practice of software engineering, because incremental development offers seemingly more opportunities for project planning.

4.1.5 The spiral model: the spiral model [15] has been introduced in order to resolve some of the problems that stem from the application of the other models, and to add the advantages of the other models onto one new framework for software development. In order to achieve this, the spiral model uses different drivers to those of the other models.

> 'The major distinguishing feature of the spiral model is that it creates a *risk-driven* approach to the software process rather than a primarily *document-driven* or *code-driven* process' [15].

The spiral model models the software development process as a sequence of progressive cycles. Each cycle consists of four steps (Fig. 4).

- ☐ Determine objectives, alternatives, constraints.
- ☐ Evaluate alternatives, identify, resolve risks.
- ☐ Develop, verify next-level product.
- ☐ Plan next phase.

At the end of every cycle a review takes place. This review covers all the products of the previous cycle, including the plans for the next cycle.

> 'The review's major objective is to ensure that all concerned parties are mutually committed to the approach for the next phase' [15].

An important feature is that risk considerations have a great impact on the activities performed during a cycle. This means that certain activities can be skipped if there seems to be no risk involved in doing so, and other activities may get a lot of attention if the risks involved are great.

Because the order of activities is not fixed, the spiral model is a generic model which encompasses both extremes; evolutionary delivery and the waterfall model. The spiral model offers the possibility of using a mixture of different models in order to adapt the development process to the specific risk sources of the current project.

A focus point in using the spiral model is the question 'how much is enough?'. Since risk, and not documentation, is the criterion for selecting the next activity, blindly striving after completeness is not required. Completeness should only be pursued to a level where the risks incurred by incompleteness are no longer significant. This does not only apply to documentation but equally well to quality assurance, verification and validation, planning, analysis, configuration management etc.

Since every cycle involves determination of the risks the project runs, quality objectives can be integrated into the development process. Once quality objectives (like user

friendliness, performance, portability etc.) have been stated, the risk of not meeting the objectives is determined at the end of every cycle. If the risks of not meeting stated quality objectives become a threat to the success of the project, these objectives will be addressed in order to avoid losing control of quality.

4.2 *Objectifying quality*

The second direction of the road that leads towards a 'state of quality control' in software engineering is concerned with enlargement of the objective component of quality, in order to make the 'ideal development process' more applicable. Enlarging the objective component means capturing more of the necessary product characteristics in the specification of required characteristics and making sure that compliance with them can be assessed objectively.

Objective assessment of a product's quality is only possible when the criteria by which the product is to be judged are specified clearly. These criteria can be divided into two levels; general criteria that apply to software products regardless of their specific purpose, and criteria that stem from the specific purpose of the product. The former can be captured by standardisation of the deliverables of the software development process, and the latter by product specification.

4.2.1 Standardisation: the first step towards enhanced objectivity is standardising the deliverables of the software development process. If measurement of the impact of the application of a specific standard indicates a positive contribution to the quality of the products, application of a standard offers a set of criteria, compliance with which would indicate the achievement of quality.

Because a standard is not directed towards the purpose a particular product is supposed to be suited for, compliance with the standard does not mean that the product will be a quality product. Standards mostly deal with internal characteristics of the product that are relevant to the developer, but that are far less relevant to the customer. Standardisation will therefore, in the first place, enhance the quality of a product in the eyes of the software engineer; it will be more structured, requirements and design decisions will be more traceable, documentation and code will be more readable etc. These characteristics have impact on the ease with which the developer can develop and maintain the product. This indirectly influences product characteristics relevant to the customer; the time a system will be 'down' after a system failure, the time and effort needed to have the product adapted to changed needs etc. Standardising the deliverables of the software development process may therefore enhance the quality of a product by offering internal criteria, compliance with which can be judged objectively.

If products comply with a standard, they have some common characteristics. These common characteristics offer some common terminology and understanding, and thereby enhanced possibilities for carrying out measurement. This is a beneficial side-effect of the application of standards in software engineering.

Many standards and guidelines for software development exist.

- **ANSI (American National Standards Institute)/IEEE standards:** these standards relate to the contents of the documentation that will be written during the software development process. A list of topics that need to be addressed and a standard table of contents are contained in these standards for various documents. Regarding the development process itself, as little as possible is prescribed. In addition to the standards, guides have been written. These guides offer suggestions regarding the way the prescribed topics should be addressed.
- **DOD standards:** the US Department of Defense standards prescribe the deliverables of the software development process, the layout and the contents of these documents, and the procedures that should be followed when informing the customer of the results of a development step. These standards are more strict than the ANSI-/IEEE standards, in that they offer less possibilities for tailoring the standards and in that they address the organisation of the development process.
- **ISO (International Standardards Organisation):** the series of ISO-9000 to ISO-9004 do not, in detail, address the way a product should be developed. They address the organisational issues related to the documentation of the measures taken for quality assurance. These measures should offer the customer confidence that the developer is capable of delivering a quality product. In their initial state, these standards are not very well suited to be applied in software development. For this reason ISO-9001 has been tailored to accommodate the specifics of software development (n33 from ISO/TC176/SC2/wgs, February 1989).

The above-mentioned standards are just a small selection of the huge collection of existing standards.

4.2.2 Specification: the criteria for quality assessment offered by standards are general criteria. Judging the objective component of quality requires, however, that the product's quality is assessed by determining compliance with stated requirements that stem from the customer's preferences and the product's intended use. These criteria are much more specific.

In software engineering, attempts have been made to design notational methods to specify the required functions of a software system unambiguously. One can think of methods like VDM [16, 17], and Z [18]. These notations specify the functions of a software system by means of a mathematical model. Other methods, such as RML [19] and Telos [20], offer means to model the problem that the system is supposed to solve and means to specify time-related behaviour of the system. Because the syntax and the semantics of specification notations are well defined, no debate on the meaning of the product specifications should arise. Checking the conformance of the product with the specifications thus can be done objectively, and if necessary, conformance or nonconformance with the functional requirements can even be proven mathematically.

The above-mentioned specification methods thus enhance the objectivity of quality and are a valuable means of enhancing quality control. Proving that the implementation complies with the specification will, however, not ensure the usability of the product because quality of design cannot be proven to exist, and because these methods have only limited capacities for specifying non-functional requirements.

What needs to be developed are methods with more facili-

ties for specifying non-functional requirements. Non-functional requirements should be specified in combination with a method for measuring whether a product complies with these requirements. This method should enable assessment of compliance to be performed objectively. This way the set of unambiguously specified requirements will capture more of the total required characteristics. Examples of notations for specifying non-functional requirements are Gilb's Planguage* and the Quality Requirements Specification Subsystem described by Walker and Kitchenham [21]. Future methods for specifying non-functional requirements would be most valuable if they could be tied to methods for formal software development and functional specification, such as VDM and Z.

As we have seen, quality of design cannot be proven. It therefore is necessary to validate the specified requirements. Methods for requirements specification should therefore offer means to provide the customer with information about the specified requirements in a way that enables him to give valid feedback.

5 Conclusion

The result of our exploration into quality has been that three distinct components of quality can be identified: an objectively assessable component, a subjectively assessable component and a non-assessable component. We have argued that it would be unwise to limit our attention to any single one of these, although only the first is suited to be engineered.

In order to achieve a 'state of quality control', we should

- arrange the software development process in a way that permits frequent validation, compensating for the problems we have with the subjective component of quality;
- develop methods for unambiguously specifying as many of the required characteristics of the product as possible. Thus, we can limit the impact of the subjective component, by stating as many of the initially subjective requirements in a way that allows objective assessment of compliance with them.

Regarding the unpredictable component of quality, little has been said because there is little that can be done. The only thing to be said is that we should be aware of the many aspects that are relevant to the system's behaviour and that have not yet been considered. Once these aspects have come to mind, we should try to transform them into clear criteria by which the product can be judged objectively. We should realise, however, that completeness in requirements specification is an illusion.

Research regarding quality control in software development should aim at developing a software development process in which frequent customer feedback and the application of formal methods are joined together. We should look for the crossing of the two roads that lead towards a 'state of quality control', because walking these roads separately solves only one of the problems. It should be our aim to control 'quality', not just 'quality of design' or 'quality of conformance'.

* GILB, T.: 'Planguage: a result-driven planning language handbook' (obtainable from the author).

6 Acknowledgment

We would like to thank Professor Jan van Katwijk for providing feedback on earlier drafts.

7 References

[1] SYKES, J.B., (Ed.): 'The concise Oxford dictionary of current English' (The Clarendon Press, 1982), 7th edn.
[2] CROSBY, P.B.: 'Quality is free' the art of making quality certain' (McGraw-Hill, 1979)
[3] JURAN, J.M., and GRYNA, F.M. Jr.: 'Quality planning and analysis: from product planning through use' (McGraw-Hill, 1980), 2nd edn.
[4] DOD-STD-2168: 'Defense System Software Quality Program'. Military Standard of The Department of Defense of The United States of America, Draft, 1st April 1987
[5] CROSBY, P.B.: 'Quality without tears: the art of hassle-free management' (McGraw-Hill, 1984)
[6] ORNSTEIN, S.M., SMITH, B.C., and SUCHMAN, L.A.: 'Strategic computing', *Bull. Atom. Sci.*, 1984, **40,** (10), pp. 11–15
[7] DEMING, W.E.: 'Out of the crisis: quality, productivity and competitive position' (Cambridge University Press, 1982)
[8] FEIGENBAUM, A.V.: 'Total quality control' (McGraw-Hill, 1986), 3rd edn.
[9] ISO-8402: Quality vocabulary'. International Standardisation Institute, 1986
[10] PETERS, T.J., and AUSTIN, N.K.: 'A passion for excellence' (Random House, New York, 1985)
[11] PETERS, T.: 'Thriving on chaos: handbook for a management revolution' (Harper & Row, New York, 1987)
[12] GRADY, R.B., and CASWELL, D.L.: 'Software metrics: establishing a company-wide program' (Prentice-Hall, 1987)
[13] ROYCE, W.W.: 'Managing the development of large software systems: concepts and techniques'. Proc. IEEE Wescon, 25th–28th August 1970, Los Angeles, California
[14] GILB, T.: 'Principles of software engineering management' (Addison-Wesley, 1988)
[15] BOEHM, B.W.: 'A spiral model of software development and enhancement', *IEEE Comput.*, 1988, **21,** (6), pp. 61–72
[16] JONES, C.B.: 'Software development: a rigorous approach' (Prentice-Hall, 1980)
[17] JONES, C.B.: 'Systematic software development using VDM' (Prentice-Hall, 1986)
[18] SPIVEY, J.M.: 'The Z notation — a reference manual' (Prentice-Hall, 1987)
[19] GREENSPAN, S.J., BORGIDA, A., and MYLOPOULOS, J.: 'A requirements modelling language and its logic', *Inf. Syst.*, 1986, **11,** (1), pp. 9–23
[20] KOUBARAKIS, M., MYLOPOULOS, J., STANLEY, M., and JARKE, M.: 'Telos: a knowledge representation language for requirements modelling'. Technical Report CSRG-222, Computer Systems Research Institute, University of Toronto, December 1988
[21] WALKER, J.G., and KITCHENHAM, B.A.: 'Quality requirements specification and evaluation' *in* KITCHENHAM, B.A., and LITTLEWOOD, B. (Eds.): 'Measurement for software control and assurance'. Proc. Centre for Software Reliability Conf. on Measurement for Software Control and Assurance, 1987, Bristol, UK

The paper was received on 4th June 1990.

The authors are with the Delft University of Technology, Faculty of Technical Mathematics and Informatics, Section Software Engineering, Julianalaan 132, P.O. Box 356, 2600 AJ Delft, The Netherlands.