IT Product Quality in Practice as Knowledge and Experience

Steen, Odd

Published in:
Proceedings of IRIS27

2004

IT Product Quality in Practice as Knowledge and Experience

Odd Steen:

School of Economics and Management, Lund University

Department of Informatics

Ole Römers väg 6, SE-223 63 Lund

odd.steen@ics.lu.se

***Abstract**: IT quality, e.g. software quality, use quality, information systems quality, is a current but problematic issue. The proliferation and increased complexity of information systems and software have introduced new quality problems. The current wisdom is to solve these problems trough a standardised and controlled process with conformance to the requirement specification as the main goal. The objective is to reach non-subjective quality management. On the results from an empirical investigation into IT professionals' perception of IT quality in practice, this paper argues for an alternative approach based on knowledge and assessment ability grounded in experience and reflection. The conclusion is that IT quality is a complex of aspects, perspectives, and stakeholders, and that important qualities of information systems do not lend them selves to normative approaches. This requires professionalism in the form of subjective assessment ability and the paper is concluded with some proposals for how the assessment ability can be strengthened.*

***Keywords**: IT quality, information systems quality, software quality, quality assessment, IT professionals.*

1. Introduction

This empirical paper presents the results from an investigation into IT professionals' perceptions of the concept of IT product quality and assessment ability. The reason for the investigation was to increase the understanding for how they, e.g. systems developers, systems designers, quality managers, and programmers, handle the quality concept, since lack of this understanding, in my view, makes it much more difficult to find ways to improve IT quality and IT professionals' assessment ability.

IT quality is a problematic phenomenon with many "parts", e.g. software quality (see e.g. Galin (2003)), use quality (see e.g. Dix *et al* (2003)), information systems quality (see e.g. Yeates and Wakefield (2004)), and economical aspects of IT investments (see e.g. Hedman and Kalling (2002)), and many perspectives and stakeholders. My intention in the paper is not to give the final solution to the IT quality problem, but rather to increase the understanding of IT product quality in *systems development practice* – an often neglected perspective in IT quality research. In the end it is systems development practice that results in good or poor IT product quality.

The concept of IT quality has been an interest since at least 1968, when at a NASA conference the problems of poor software at that time was called the Software Crisis (Dijkstra, 1979, Pressman, 1987). The last ten years the interest has increased and in ACM and IEEE, two major associations for software and information systems research, the number of articles and reviews concerning software quality, information systems quality, and use quality, has grown[1].

In Scandinavian research Dahlbom & Mathiassen (1993) discuss IT quality in their quite philosophical book "Computers in Context", where they argue that quality is a compromise between different qualities, which I will do also, but they do it without any empirical evidence. Braa (1995) in her doctoral dissertation discuss IS product quality as three perspectives: technical quality, use quality, and organisational quality. Apart from this, she also focuses on process quality with the notion that

[1] A search was performed in the ACM Digital Library using the following search terms (searching in "all information") for all publication years. The up to 200 best matches were published between 1993 and 2003. In percentage: [software+quality] 58%, [systems+quality] 63%, and [use+quality] 80%. The ACM Digital Library: (http://portal.acm.org/dl.cfm?coll=portal&dl=ACM&CFID=16967686&CFTOKEN=62849447).

A similar search in IEEE journals, conference proceedings, and standards resulted in: [('quality' <in> ab) <and> ('software' <in> ab)] 81%, [('quality' <in> ab) <and> ('information' <in> ab) <and> ('system' <in> ab)] 84%, and [('usability' <in> ab) <and> ('information systems' <in> ab)] 82%. (http://ieeexplore.ieee.org/search/advsearch.jsp).

process quality and product quality are interrelated since it is in the design process that product quality is more or less achieved.

So despite the increased research interest into IT quality, the IT professionals' perspectives are almost totally disregarded or not researched. Counter examples of empirical research are Stolterman's (1991) study of the hidden rationality of systems developers, Hoberg's (1998) reflective book on the importance of improvisation and precision in the systems development process, and Wilson and Hall's (1998) study of the different perceptions of software quality among IT professionals and management. More recent studies are Meggerle and Steen's (2002) study from a knowledge/knowing perspective of some systems developers perceptions of the concept of IT quality and assessment ability, and Schönström's (forthcoming) study, based on personal experience, of software development as a type of knowledge work. There are few other studies of IT professionals' perception of quality and assessment.

The small number of studies is explained by the general focus on process control to reduce subjective assessments. In recent Software Engineering (SE) and particularly in SPI (Software Process Improvement) the focus is on process improvement (Schönström, forthcoming) as a way to produce higher quality. There has obviously been a shift in focus to process quality. Also, in Software Engineering (SE) and Human Computer Interaction (HCI), two major research fields contributing to the IT quality understanding, the focus has been on the concept of quality, metrics, standards, techniques, and methods, with little interest in the systems development practice and the professionals' perception.

Hence, there is a need to investigate the practical side of quality and to pick up the IT product quality perspective again, but with the notion that the understanding and perception of product quality influences the achieved quality (and not only that design influence product quality but that quality perception influences both product and design). The purpose of this empirical paper is thus to address the following two research questions:

- What perceptions do systems development professionals have of IT product quality and quality assessment ability? This question is investigated from a practical knowledge perspective as described later.

- And how does that influence the concept of IT product quality and quality assessment and control? The empirical conclusions from the investigation will give tentative answers to these questions.

According to Budgen (2003) is software development a *design* rather than an engineering activity, and deals as such with ill-structured problems more than with structured ones. My conclusion is therefore that software design and systems development, in order to handle ill-structured problems, must be thought of as being based on experience based practical knowing and skill (Polanyi, 1966, Rolf, 1995,

Johannessen, 1999) and this will be my guiding principle in this paper. I will however only use Johannessen (1999) in the paper.

The paper is structured as follows. In section two the concept of IT quality and the lack of investigations into perceptions of IT quality in practice are discussed briefly. In section three is a short description of the investigation of 18 systems developers' perception of IT quality and assessment. The results from the investigation are presented in section four. The paper is concluded in section five with a discussion of the empirical findings.

2. IT Quality

In the early days of computing the efficient use of CPU time and working memory was the main focus and the primary measure of quality was efficiency in terms of memory and CPU usage (Budgen, 1994). This situation led to a drive for "clever" and efficient solutions (Dijkstra, 1979) which did not promote good software quality in terms of e.g. readability and maintainability.

The first step taken towards better software quality was what later would be called structured programming (Yourdon, 1979) and structured design (Stevens et al., 1979). Parnas (1979) described principles for better maintainability, reusability, and information hiding, which would increase maintainability.

These ideas and the Software Crisis led to the discipline Software Engineering, which is a term that implies software development based on theoretical and practical foundations in other engineering disciplines (Shapiro, 1997) and thereby a more systematic and engineering-like process. The efforts led to new ideas about software quality, apart from efficiency, such as modularity, abstraction, and metrics.

Basically in SE there are two different approaches to reach good quality: a scientific based on mathematics, and an engineering-like based on pragmatics. Although some claim that the problems of poor software quality is attributable to scientific weakness, and hence inadequate application of mathematics, in Software Engineering (Baber, 1997), the mathematical approach seems to be dominating (Meggerle and Steen, 2002). Software quality must be objectively measurable and independent of a subject's interpretation (Pressman, 1987, Kan, 1995, Sommerville, 1996). Different mathematical models and methods for measuring quality were thus developed (Shapiro, 1997).

During the 1960's use quality also became a problem when computers were being used by others than technicians in labs. This situation led to efforts to make more usable programs, through ground-breaking work in the early 1960's like Ivan Sutherland's Sketchpad, word processing and mouse pointer by Douglas Engelbart at Stanford University (Dix *et al.*, 2003), Alan Key's Smalltalk in the mid 1970's (ibid.),

and WIMP[2] interfaces, interaction metaphors, and direct manipulation in the 1980's (ibid.). A new field of research and development, Human Computer Interaction, emerged. The HCI field has over the years developed from machine-like thoughts about man to a more complex blend of behaviouristic, cognitive, emotional, and social aspects (Meggerle and Steen, 2002).

Software Engineering, with its constructional perspective, and HCI, with its use perspective, do together constitute much of the knowledge about IT quality at large.

The focus in contemporary research on quality, especially in SE, is however generally on *process* quality rather than *product* quality, which is no surprise given that the control of the process to reach high product quality has been in focus for more than 20 years. Many of the ideas from manufacturing industry, such as total quality management (TQM) and total quality control (TQC) (Crosby, 1979), and standards such as ISO-9000 (Oskarsson and von Schantz, 1987, Sanders and Curran, 1994, Ince, 1995, Sommerville, 1996, Galin, 2003) have found their way into the IT industry also. These ideas all stress the importance of Quality Assurance (QA), and thus the implementation of a quality system, to control the development and manufacturing process rather than the product.

This focus on the process is understandable and acceptable, but may also lead to an even but poor product quality (through a well-controlled process) and a less focus on product quality as such. For the advocates of process control in systems development, IT quality seems often to be an already solved problem and the control of the process can start with standard measures of software qualities in an effort to standardise the process (se for instance Burr and Owen (1996)). That the control perspective has not solved the problem of poor quality of IT products is quite obvious considering the multitude of problems today's users still face when using the technology. The problems of, for instance, maintainability, modifiability, security, and stability are partly due to increased software complexity, partly due to *poor construction* of software products. Poor *use quality* is also still a problem even though great developments have been achieved in this area. Mitchell Kapor's more than ten year old design manifesto[3] still holds true to many end-users:

> The lack of usability of software and the poor design of programs are the secret shame of the industry [...] Computing professionals themselves should take responsibility for creating a positive user experience. [...] By training and inclination, people who develops programs haven't been oriented to design issues. This is not to fault the vital work of programmers. It is simply to say that the perspectives and skills that are critical to good design are typically absent from the development

[2] Acronym for Windows, Icons, Menus and Pointing devices.

[3] First appearance in print in Dr. Dobbs Journal in 1991.

> process, or, if present, exists only in an underground fashion (Kapor, 1996, p. 3).

Interesting in this quote are the words about computing professionals, perspectives, and skills. In the design of IT products it is ultimately the systems developers, usability engineers, and programmers that influence quality through their work and their ability. To find ways to improve IT quality it is thus important to understand how they perceive IT quality and how they handle it in practice – that is, to understand systems developers' knowledge and skill in relation to IT product quality.

3. Investigation into It Quality in Practice

To investigate how the concept of IT quality can be perceived in practice, a case-study at three systems development organisations was carried out. These organisations were in-house IT departments of two major Swedish corporations and a regional office of one major European IT consultancy firm. Two broad research questions were investigated:

- Systems developers' knowledge of the concept of IT quality as such, i.e. what it means to them
- Their perception of knowledge in action in relation to IT quality, i.e. how it is handled in practice

3.1 Conceptual Frame of the Investigation

The research questions are of a knowledge nature and the two research questions are framed in the knowledge theory of Johannessen (1999) who interpret the late Wittgenstein.

This theory contradicts the logical positivists' claim for an objective and true reality, where language and knowledge corresponds with a given reality through either direct empirical evidence or logical language operations. Johannessen suggests that there is indeed a type of knowledge in the logical positivist sense, which is termed stated knowledge, but also kinds of knowledge that cannot be stated and framed and that are shown in action, where formal rules and exact words are insufficient, requiring indirect communication through metaphors, examples, and action. Those types of knowledge are termed respectively skill and familiarity.

From this perspective the first research question deals with verbalised and stated knowledge about the concept of IT quality, e.g. quality definitions, standards, measures, and attributes; what could be viewed as positivistic and objective knowledge. The second research question deals with skill and knowledge-in-action, and familiarity (as a kind of knowledge) with situations where this skill is meaningful,

and is hence based on a more subjective view of knowledge. These two questions were further divided into six themes:

- Definition of quality
- Characteristics of respectively good and poor quality
- The importance of quality
- Quality assessment ability
- The evolvement and forming of the assessment ability
- Communication of a quality assessment

Theme number one and, to a lesser extent, number two and number six do specifically address verbalised and stated knowledge about IT quality as attributes and definitions, that is: How is quality defined? How are the definitions, if any, used? What words are used to denote quality? Is there a special kind of language to communicate an assessment of quality?

Theme number two through number five address IT quality as the result of knowledge-in-action, tacit knowledge and experience, i.e. the kind of knowledge that is difficult or impossible to frame in externalised descriptions or norms, such as definitions, standards, or measures. Questions asked here are for example: Is it necessary to make trade-offs between different qualities? How is quality assessed? How is the assessment ability developed? Is it influenced by cooperation with other systems developers? How is the perception of good or poor quality of a system communicated to another systems developer, if possible?

3.2 How the Investigation Was Conducted

The six themes require quite in-depth reflection and pondering and therefore a qualitative research design was adopted. The chosen research method was the qualitative interview (Kvale, 1997) and 19 were conducted[4] with systems developers ranging from programmers to quality managers.[5] Within each of the six themes a number of interview questions like the ones above were formed in a questionnaire (cf. Meggerle and Steen (2002)) for the complete questionnaire), used for in-depth, semi-structured and tape recorded interviews.

[4] 19 interviews with 18 persons.

[5] 5 women and 14 men were interviewed. The most experienced had 25 years in the practice and the least experienced 4 months. The average was 7 years of experience with the standard deviation of 7.09. The median was 4.5 years. The most experienced interviewees had worked with a broad range of systems development issues; programming, systems design, implementation, maintenance design, project management. The least experienced had mainly worked with programming. Based on their at that time current assignments, 2 of the interviewees were quality managers, 5 project managers, 2 functionality designers, 6 programmers, and 3 systems designers.

The interviews took place at the interviewees' work places and the time required for the interviews was between 30 and 90 minutes. All interviews were conducted before analysis and then prepared for analysis by transcription from audio tapes to text documents with numbered units for better traceability and chain of evidence (Yin, 1994).

In the first stage the 320 pages of transcripts were deductively coded (Miles and Huberman, 1994) using a coding scheme developed from the three knowledge types: stated knowledge, skill, and familiarity (cf. Meggerle and Steen (2002)) for the complete coding scheme). This coding was conducted by the two researchers separately and individually and through this procedure the coding reliability was strengthen (Miles and Huberman, 1994) and the ratio of coding similarity was above 90 percent.

In the second stage the units of analysis in the coded interviews were clustered under each separate code and sense concentration (Kvale, 1997) was performed on each unit to expresses the essence of the unit in fewer words to reduce the text.

In the third stage summarising interpretations of the interviews, based on the sense concentration, were produced. After this final stage an interview transcript of about 20 pages was reduced to two to three pages.

The strength of qualitative investigation, according to Kvale (ibid.), is the possibility of multiple conclusion, and therefore validity (and reliability following Enerstvedt's (1989) argumentation) is based on research skill and avoidance of "black-box" methods descriptions. A paper is not the proper place to avoid the "black-box" since it would require too much space. There is however also the opportunity of respondent validation (Kvale, 1997) and the validity of this investigation is thus strengthened by the respondents giving feedback on the interview transcripts.

Generalisation of qualitative studies are discussed by e.g. Stake (1994) and Yin (1994), and following Stake the generalisation of this investigation is naturalistic and leads to expectations rather than formal predictions, with the aim of verbalising and transforming tacit knowledge into stated knowledge.

4. Empirical Findings

The findings are purely empirical, based on empirical conclusion drawing, and the presentation will be organised around the six themes and will through this be linked back to the two main research questions:

- Systems developers' knowledge of the concept of software quality as such, i.e. what it means to them

- Their perception of knowledge in action in relation to software quality, i.e. how it is handled in practice

Translated citations from the sense concentration will be used through out the discussion. The numbers indicate interview and unit, e.g. I1-58 means interview one, unit 58.

4.1 Definition of quality

The systems developers in the study had problems defining the concept of IT quality. On one hand the ISO definition of quality[6] is to some extent applicable, but it does not give enough detailed information, apart from the fact that quality is characteristics of a product and that stated or implied needs should be met. A quality definition of this kind is too broad, ambiguous, and general to be made operational. A more specific definition, e.g. that the error rate of software should fall below one error per thousand lines of code, is easy to make operational but is on the other hand too specific and excludes everything but the software as such.

The difficulty of defining the concept of quality is due to the intrinsic complexity of the concept, since quality assessment and perception is related to perspectives, roles, and stakeholders and to a large part is subjective in nature:

> Quality is subjective and not everyone has the same quality notion (I1-58). There are aspects of quality that cannot be verbalised (I1-29). X think that it is possible to find a general definition of quality in a dictionary or to produce a definition, but in practice it is something that you feel and can be hard to decide (I15-79). Quality is defined from various roles and perspectives (I3-17).

Quality can be perceived from an organisational, a technical, and a user perspective, with different quality goals. From an organisational perspective quality means functionality and efficient support of operations, not specifically maintainability and usability of software. From a technical perspective quality means software quality, not specifically efficient operational support or usability. From a user perspective the most important quality is user experience. The available resources for development complicate the picture even further:

> Quality can be viewed from three different perspectives: the system should do what the customer wants, the system should be well built for the required environments, and the code should be maintainable (I12-10).

[6] The totality of characteristics of a product or service that bear on its ability to satisfy stated or implied needs Hägerfors (1995).

Depending on what role you perform within a perspective, quality is also perceived differently. As for maintainability, not just does the customer's view influence this quality but also the developer's. If a developer lacks experience from systems maintenance, he or she will not perceive, or maybe more correctly, acknowledge maintainability or readability as important qualities. A user in the role of super-user in a systems development project has another understanding of the total project and perhaps a differing quality perception than the other users-to-be, with a slightly limited knowledge of the project. Thus, perspectives are also associated with roles performed within the development organisation and the user organisation.

There are basically three stakeholders, the customer, user, and developer, and these perceive and assess quality differently:

> Quality can be viewed from a number of different perspectives. From the users' view is quality functionality, how easy it is to learn how to use a product and if it fulfils theirs needs. From a technicians perspective quality is a matter of performance, stability, and that the product works well (I11-16). Quality is not what you, the supplier, think but what the customer think you have accomplished and if that meet their needs (I6-36).

A developer might regard quality from a technical perspective, e.g. readability of code and maintainability, as more important than a customer does, since developing for high maintainability normally is more costly in both time and money. Probably most important to a customer is quality from an organisational perspective since this quality, or lack thereof, is evident in a way that maintainability is not. Readable and understandable code is connected to maintainability which is connected to modifiability, which in turn is required to have a system that can be modified to changing organisational needs. The systems developers know these are crucial qualities but may have problems conveying the importance of them to the customers, since these qualities costs to build for but are not readily evident to others than maintenance designers.

Then, who decides if the quality of a system or program is good or not? In the end it is the customer, but the developers have an influence too since not all qualities are assessable by customers. Quality is therefore in reality a joint assessment or a mixed picture from the stake holders:

> Quality is the combined picture of the involved stakeholders' viewpoints (I8-9).

4.2 Characteristics of Good and Poor Quality

The most important qualities of all are that the systems and programs should be functional, useable, and maintainable tools supporting operations and activities, with functionality as the most important quality:

> A system should have simplicity; it should be a business operations adapted tool. Work should govern how the tool should function. Poor quality means that the tool does not fit the user's daily work (I4-26).

A program of poor quality is for instance instable, hard for the users to understand and learn how to operate, and poorly written and hence hard for maintenance designers to understand and maintain. A program of good quality is of course the right opposite of this.

A system of good quality shares many of the virtues and characteristics of a good quality program, but does also show good interoperability between its constituent parts, i.e. the programs, and with other systems. Because a system, in a way, represents a frozen image of the organisation at a certain point in time, is it important that it is possible to modify the system according to changing needs. A system of good quality is thus modifiable to a degree that a poor quality system is not.

The assessment of quality is mostly based on the requirements specification, which provides the agreed level of quality. Interestingly the systems developers regarded this level as moderate and perceived good quality as surpassing the requirements and delivering a greater contribution than the customer expected, without overdoing the design and providing more functionality than necessary:

> Good quality means that the system delivers a substantial added value compared to the specification, e.g. that it will be significantly faster and the programs easier to use. Furthermore it should be maintainable so it will not take a long time to develop further (I14-26).

4.3 The Importance of Quality

Quality is of course very important and no quality can result in no customers. At the same time the importance of quality should not be overstated since quality has no merit in it self, but is related to money and other resources:

> According to X quality has no end in it self, but must be assessed by someone from a number of criteria (I4-6).

Quality is also a matter of handling expectancies and image, in the sense that you would expect better quality of a luxury Mercedes than of a standard Ford car, since the customer who buys the Mercedes is prepared to pay more and thus expects more. The same holds true for systems.

4.4 Quality Assessment Ability

Quality is seldom or not at all measured and the systems developers did not know of any measurement methods such as McCabes Cyclomatic Complexity measure (McCabe (1976), in Shapiro (1997)):

> Even if the company has standards for measuring quality does X think that quality measurement is very difficult (I6-16). Quality is not measured at all (I2-38). Maintainability is not measured but assessed through experience-based knowledge (I7-22).

The systems developers thought that they measured too little but also found it difficult to construct measurable criteria, i.e. metrics, and measurement methods that are standardised enough to allow values to be compared between projects. Without this comparability, metrics and measurement would be much less interesting.

There is also a difference between measurable and non-measurable criteria according to the developers. They made a distinction between "hard" technical qualities that can be more easily measured, such as response time in seconds, and "soft" qualities, for instance acceptability, which they perceived as almost impossible to measure.

> There are some measurable criteria and some vague where it is a question of judgement (I2-45). Quality is subjective and hard to measure (I1-114).

Somehow quality has to be assessed and this is mostly done by testing at different stages and levels in the development process to see if the system under development fulfils the requirements specification. Occasionally formal software reviews are also carried out, but other formal, systematic, and structured approaches, like metrics and measurement, were unknown of and viewed a bit suspiciously.

The ability to assess quality is thus not a question of using formal and "objective" methods and metrics, but to large part subjective valuations. The same way as the requirements specification is a means to represent the least acceptable or moderate quality, methods, guidelines, and standards are also only means to a developer, not substitutes for an experienced developer. Proper use of these structured means will normally result in acceptable quality, but good quality or excellence is the result of personal professional experience and "Fingerspitzengefühl[7]", specifically when it comes to "softer" qualities. The experience-based feeling is also very important for judging when to stop designing a system, since design does not have a natural end point and can be carried on for ever (almost).

> There are aspects of assessment for a developer that lies outside the normative. Methods and such are frameworks that are complemented by experience of what is good and favourable, which is individual to X and belongs to the hidden knowledge (I3-46). The assessment ability includes knowing how far to go to reach simplicity and when to stop, since a change can be pursued infinitely according to X. The hard thing is to reach enough simplicity and that is part of the assessment ability (I8-32).

[7] That is German for "fine feeling" or sure instinct.

Important aspects of the assessment ability are good knowledge about the actual program or system and the context it is used in. The most important aspect of the assessment ability is however experience and feeling, what can be termed tacit knowledge. The tacit knowledge is the personal opinion about quality, based on experience and certain personal frames which are adjusted to the workplace and colleagues:

> Each individual has a personal frame of reference for quality assessment, a filter influenced by social interaction. Hence is the social interaction very important, because it adapts the individual's filter and perspective of truth. This frame of reference does not turn out in standards and statistics but has great influence on the tacit knowledge (I6-43).

4.5 The Development and Forming of the Assessment Ability

Formal education and various courses, though they can give a basic knowledge about quality, are less important in developing the assessment ability:

> Part of the assessment ability is in the start developed by reading or taking a course (I9-57). Rules and standards for quality assessment can be learned in education (I6-42). Experience from work and various projects is a great part of assessment ability and determinant for good or poor quality, and this exceeds what can be framed in education or guide lines (I8-51).

To some extent new technology has an influence through new technical possibilities and thus new ways to design systems and programs. The relation between quality and new technology is however not obvious or natural, since for instance a well designed but "old-fashioned" user interface can be better than a new but "flashy" one. Quality theories are also not important in the development and forming of assessment ability:

> It is the personal experiences that develop the assessment ability, not quality theories (I7-51). You acquire a repertoire of experiences which is important for the vague and intuitive feeling concerning quality, that which cannot be measured (I1-91).

This is not surprising considering the great importance placed on personal, professional experience discussed above

Since experience is the most significant source for assessment ability, feedback from customers and users is important. This feedback is usually of the negative kind and criticism; positive feedback or commendation is much rarer. Words about flaws, bugs, and errors are always communicated to the developers, while silence indicates no problem and hence acceptable quality for customers and users:

> If the users experience poor quality, X will get feedback to learn much from and have experience of what causes problems with the users (I10-39).

The negative feedback is however most important for development of assessment ability, since it gives valuable insights into how users perceive quality and is detailed in a way positive feedback is not. Detailed feedback is a source for learning, while commendation is a source for self esteem but harder to draw lessons from. The positive feedback is however important too, because it is a base for conclusions about quality and knowledge about what worked well and can be reused in later projects:

> Feedback from users experiencing good quality helps build knowledge that can be used for other systems or in other projects, if there is time to reflect upon what it is in the solution and the situations that is experienced as positive and good quality (I12-46).

Another important input to experience is cooperation and exchange of perceptions and knowledge with other developers, mainly colleagues. More experienced developers can act as coaches for newly employed and less experienced ones and sometimes knowledge transfer is organised through dedicated groups or individuals responsible for communicating insights from a certain knowledge domain. The developers also frequently ask for opinion from each other about solutions they are working on or consult each other for solutions to already solved problems:

> To X it is important not to put your self in a corner, but to constantly engage in an ongoing dialog about good solutions that can be used again, how things should be solved in a program, and what solution there already are (I15-79). Through interchange [with colleagues] of experiences, which is important, and discussions, you learn and form assessment ability (I5-51).

Different kinds of input to develop experience are thus important and seem to require time for reflection, which was regarded as a way to develop the assessment ability and become better developers. Time for reflection was however something that the developers did not have enough of, which they found problematic. The culture of either the profession or the workplaces that were investigated seems to be such that reflection upon the concept of quality and its meanings is not an important part of work. The systems developers by and large wanted more time for reflection, but the pressure to keep deadlines and commence new projects severely limited the possibility:

> X is positively sure that you could be a better professional if you were given time to reflect upon the quality of your products, but follow-ups and reflection is never given enough time and resources and is therefore never given any priority (I12-81).

Another aspect in relation to reflection, is that the developers and their organisations did not them selves investigate the quality of their systems and programs after delivery to customer. As discussed above, presumably good quality led to silent customers and users, a silence that definitely not contribute to development of experience through reflection upon perceived good quality. An explanation of this situation is the already mentioned lack of time for reflection, but also the focus on problems which was perceived as a characterising the nature of the profession.

4.6 Communication of a Quality Assessment

The interviewed systems developers did, in their own opinion, not talk about or discuss quality generally or conceptually. This is explained by the lack of reflection, and time for it, upon quality and users' and customers' opinion about a system, as discussed above. All discussion and reflection is on a particularity level in relation to a specific project to solve an emergent problem. Discussions to learn more about quality, qualities, perceptions, and assessment did thus not take place on a conceptual level, though some found it necessary:

> X is of the opinion that reflection upon and discussion about quality can be supported partly by the company acquiring a general systems development strategy, partly by courses and seminars that provide a more theoretical knowledge about the quality concept and various ways to measure quality (I12-58).

The systems developers did also not consider there to be any specific and exact concepts usable in conveying a quality assessment to another person, e.g. another systems developer, despite that they used quite a few words and concepts to describe quality and qualities. They explained this surprising situation such that they did not really use special quality concepts, but talked about quality in every day language, using every day words such as good/bad, complicated/easy, logical/confusing:

> Have no general concepts for discussing quality, e.g. maintainability, but the aspect can be discussed in other words and varying terms (I8-84). There are few words to describe the user interface and hard to describe it without showing it (I2-71).

As a consequence of this, they did not have a special quality language used to describe quality in a product.

> You talk about aspects of quality, but perhaps not in the form of a language (I2-60). X does not think that there is a certain language with special terms to talk about good or poor quality and opinions, instead everyday words are used such as good/bad, nice/bad, complicated/easy, and logical/confused (I15-56).

Some regarded standards, guide lines, and methods as parts of a quality language, but the majority of the interviewees lacked an accurate professional language with exact concepts for communication about quality, a language they would welcome:

> X would really want a language to talk about quality and opinions, since it would make things simpler in the sense that it would have really expressed something about quality [...] (I14-68, 69). A quality language with common concepts would be good and could facilitate talking about quality, so that you talk about the same thing (I3-64).

Not all knowledge, as was discussed earlier, can be framed and stated verbally, but must rely on indirect communication through metaphors and examples. Examples of systems and programs were however not used by the systems developers, at least not in a structured way to convey a quality assessment, despite that they thought it hard to convey an impression of a system without trying it out and showing it:

> You do not use good examples [or exemplars] in a structured way, but you have them any way (I4-58).

Closest to an example in this respect was the projects at hand because the discussions about quality circled around them, not around the concept of quality.

5. Conclusions

From this investigation it is clear that IT quality is a complex of qualities relating to perspectives, stakeholders, and roles, and has to be perceived both from an organisational, a technical, and a user view. Quality will always be a blend and a compromise between various qualities, with different importance to different stakeholders, and available resources for development. It is thus very difficult, lest not say impossible, for the investigated IT professionals to express a functional and general definition of quality that can be used for assessments and descriptions, or for structured approaches, such as methods and metrics, to comprehensively control quality of software. Instead, they place much emphasis upon the subjective and experience-based assessment ability, the ability that can give excellent, rather than moderate quality as expressed in requirement specifications.

It is thus surprising that their experience is developed in an underground, or at least unsystematic, fashion with little time and attention devoted to quality as a concept, assessment ability, and reflection. Likewise, the support, in the form of systematic input from customers and users, language, and examples (or better, exemplars) for reflection, is poor.

A reason for this might be that the common sense, or philosophy, in research and development in dominating fields, especially in SE, is that the process should be controlled and the product measured objectively as conformance to requirements

specifications; assessment should not be subjective. The responsibility for quality should be external in relation to the individual. These efforts have however not succeeded yet, even though they have contributed significantly to both research and practice, and it might be due, among many things, to the notion of requirements specifications as acceptable, not good, quality coming forth as one interesting finding in this paper.

I think that the solution to the problem of IT quality cannot be solved through control of a rational and objective process alone, since quality obviously is too complex to define and capture in objective, externalised, and operational concepts – the practitioners' skill and assessment ability are, in the end, decisive of the quality of IT and should be viewed as necessary parts of professionalism instead of obstacles. To find ways to strengthen and develop professionalism and combine this with methods, standards, and process control, i.e. normativism, seems to be the right path to walk. The implications of this are that there should be much more focus on systems development as knowledge and knowing both in research, education, and practice.

In the practice the reflective ability has to be trained and acknowledged as important and given time. A common professional quality language made up of the more or less precise terms, or quality attributes, which can be found in relevant fields, should be used to drive part of the reflection. A more systematic use of examples should also serve as aids in reflection. But above all, there has to be time for the professionals to reflect, talk, and investigate quality.

In research too much emphasis is placed on engineering and control and too little on ability and knowledge in action. It would be quite interesting to study if a knowledge-in-action oriented approach combined with a quality assurance approach would result in better quality, and thus fertilise each other.

In education a broader grip on the IT quality issue must be taken. IT systems are not just about constructional qualities or usability, but a compromise between many conflicting demands and wishes. Students have to learn about standards, metrics, and methods, as well as about reflection, knowledge, and assessment.

6. References

Baber, R. L. (1997): Comparison of Electrical "Engineering" of Heaviside's Times and Software "Engineering" of Our Times. *IEEE Annals of the History of Computing*, vol. 19, no. 4.

Braa, K. (1995): *Beyond Formal Quality in Information Systems Design.* Dissertation, Research report No. 202, Department of Informatics, University of Oslo, Oslo.

Budgen, D. (1994): *Software design.* Addison-Wesley, Wokingham.

Budgen, D. (2003): *Software design.* 2nd ed. Addison-Wesley, Harlow, England

Burr, A. & Owen, M. (1996): *Statistical methods for software quality : using metrics to control process and product quality.* International Thomson Computer Press, London.

Crosby, P. B. (1979): *Quality is Free : the Art of Making Quality Certain.* McGraw-Hill, New York.

Dahlbom, B. & Mathiassen, L. (1993): *Computers in Context – The Philosophy and Practice of Systems Design.* NCC Blackwell, Cambridge.

Denzin, N. K. & Lincoln, Y. S. (eds.1994): *Handbook of qualitative research.* Sage Publications, Thousand Oaks.

Dijkstra, E. W. (1979): The humble programmer. In Yourdon, E. (ed): *Classics in software engineering.*

Dix, A., Finlay, J., Abowd, G. D. & Beale, R. (2003): *Human-computer interaction.* 3rd ed.. Pearson, Upper Saddle River, NJ.

Enerstvedt, R. T. (1989): The Problem of Validity In Social Science. In Kvale, S. (ed.): *Issues of Validity in Qualitative Research.*

Galin, D. (2003): *Software quality assurance.* Pearson Education Limited, New York.

Hedman, J. & Kalling, T. (2002): *IT and business models : concepts and theories.* Liber Ekonomi Abstrakt, Malmö.

Hoberg, C. (1998): *Precision och improvisation – om systemutvecklares yrkeskunnande.* Dialoger, Stockholm. (in Swedish)

Hägerfors, A. (1995): *Att samlära i systemdesign.* Studentlitteratur, Lund. (in Swedish)

Ince, D. (1995): *Software Quality Assurance – a student introduction.* McGraw-Hill Book Company, London.

Johannessen, K. S. (1999): *Praxis och tyst kunnande.* Dialoger cop., Stockholm. (in Swedish)

Kan, S. H. (1995): *Metrics and Models in Software Quality Engineering.* Addison-Wesley, Reading.

Kapor, M. (1996): A Software Design Manifesto. In Winograd, T., Bennet, J., De Young, L. & Hartfield, B., (eds.): *Bringing Design to Software.*

Kvale, S. (1997): *Den kvalitativa forskningsintervjun.* Studentlitteratur, Lund. (in Swedish)

Kvale, S. (ed.1989): *Issues of Validity in Qualitative Research.* Studentlitteratur, Lund.

McCabe, T. J. (1976): A Complexity Measure. *IEEE Transactions on Software Engineering*, vol. 2, nr. October, p. 61. (referenced in Shapiro, 1997)

Meggerle, T. & Steen, O. (2002): *IT-kvalitet i praxis: systemutvecklares kunskap om och syn på kvalitet.* Doktorsavhandling, inst. för Informatik, Lunds universitet, Lund. (in Swedish)

Miles, M. B. & Huberman, A. M. (1994): *Qualitative data analysis : an expanded sourcebook.* 2nd ed. Sage Publications, Thousand Oaks.

Oskarsson, Ö. & von Schantz, C. (1987): *Kvalitetssäkring av programvara.* 2nd ed., Mekanförbundets förlag, Stockholm. (in Swedish)

Parnas, D. L. (1979): On the criteria to be used in decomposing systems into modules. In Yourdon, E. (ed.): *Classics in software engineering.*

Polanyi, M. (1966): *The tacit dimension.* Doubleday, Garden City, N.Y.

Pressman, R. S. (1987): *Software engineering : a practitioner's approach.* 2nd ed., McGraw-Hill, New York.

Rolf, B. (1995): *Profession, tradition och tyst kunskap.* 2nd ed., Nya Doxa, Nora. (in Swedish)

Sanders, J. & Curran, E. (1994): *Software quality : a framework for success in software development and support.* Addison-Wesley Pub. Co., Reading, Mass.

Schönström, M. (forthcoming): Software Development as Knowledge Work - A Conceptual Framework. In Hedman, J., Kalling, T., Khakar, D. & Steen, O. (eds.): *(in progress).*

Shapiro, S. (1997): Splitting the Difference: The Historical Necessity of Synthesis in Software Engineering. *IEEE Annals of the History of Computing*, vol. 19, no. 1, pp. 20-54.

Sommerville, I. (1996): *Software Engineering.* 5th ed. Addison-Wesley, Wokingham.

Stake, R. E. (1994): Case Studies. In Denzin, N. K. &Lincoln, Y. S. (eds.): *Handbook of Qualitative Research.*

Stevens, W., Myers, G. & Constantine, L. (1979): Structured design. In Yourdon, E. (ed.): *Classics in software engineering.*

Stolterman, E. (1991): *Designarbetets dolda Rationalitet – en studie av metodik och praktik inom systemutveckling.* Doktorsavhandling, Institutionen för Informationsbehandling, Umeå Universitet, Umeå. (in Swedish)

Winograd, T., Bennet, J., De Young, L. & Hartfield, B., (eds. 1996): *Bringing Design to Software.* Addison-Wesley Publishing Company, New York.

Wilson, D. N. & Hall, T. (1998): Perceptions of software quality: a pilot study. *Software Quality Journal.* no. 7, pp. 67-75.

Yeates, D. & Wakefield, T. (2004): *Systems Analysis and Design.* 2nd ed. Pearson Education, Harlow, England.

Yin, R. K. (1994): *Case Study Research – Design and Methods.* 2nd ed., SAGE Publications, Thousands Oaks.

Yourdon, E. (ed. 1979): *Classics in software engineering.* Yourdon Press, New York.