Lecture 2 – What is Quality?

Objectives

The purpose of this lecture are to offer a number of different views of quality and how each of these relates to different views of software and software quality.

Our Definition of Quality

For this course, the definition of quality that we have chosen is:

Conformance to requirements, both explicit and implicit.

This is just one view of quality. In this lecture we will be considering several other views.

Garvin[1]'s Five Views of Quality

Who do you think is the best footballer in the Premier League?

How would you define “best”?

It is far easier to identify the best team in the Premier League because there is league system that awards points for wins and draws. But does the best team necessarily consist of the best players?

What is my ideal car? Well, I long for a Ferrari 550 Marinello (in red, of course). Is the Ferrari 550 Marinello a quality car? In the sense that embodies all that I desire in a car (performance, looks, comfort, image, sheer style, etc), yes.

There is some ideal car that I’m measuring other cars against and the Ferrari 550 Marinello comes closest to my ideal. Others might prefer a Rolls-Royce, a Bentley, a Bristol or an Aston Martin. This is what Garvin calls the *Transcendent View* of quality. The Ferrari 550 Marinello comes closest to my ideal – in software quality terms we strive towards an ideal, but we may never be able to achieve it.

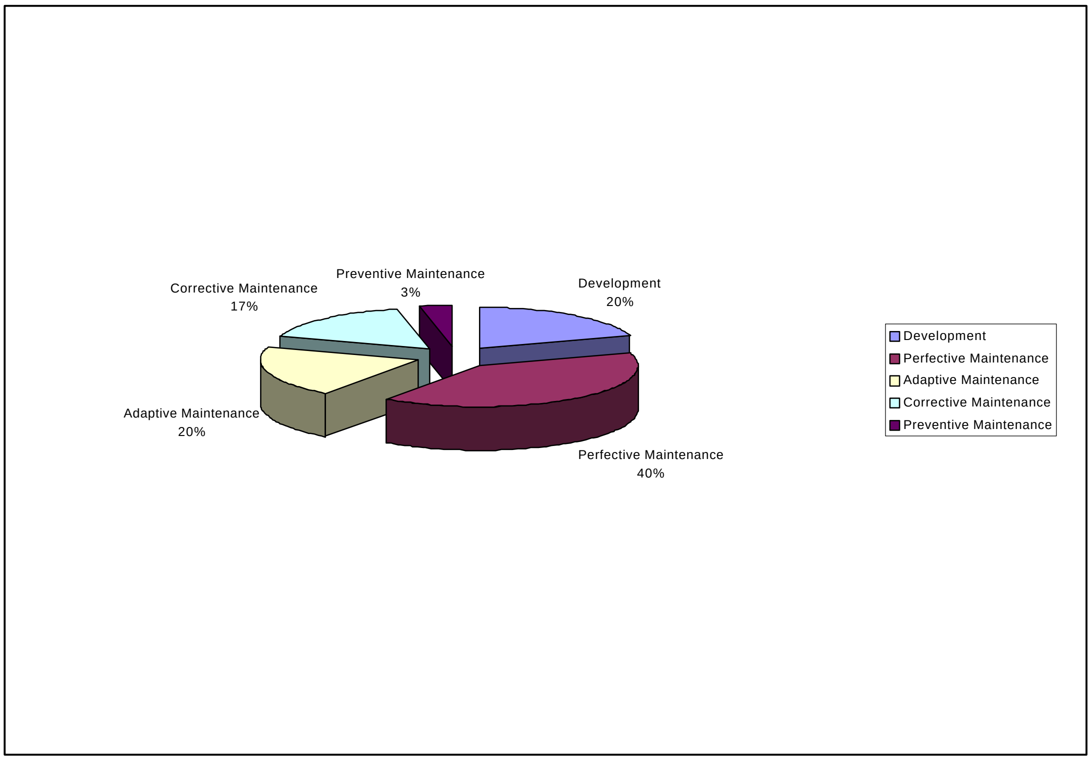
But if all I want is a car to take me to the supermarket on Saturday mornings, my Ferrari will do the job, but then so will a Ford Fiesta. So is the Fiesta a quality car? Well it does the job and taking into account running costs, some might argue that a Fiesta is more fit for purpose than the Ferrari. This is what Garvin calls the *User-Based View*. If you are a user and if all you want to do is a simple analysis of sales figures on a month-by-month basis, then Microsoft’s Excel is probably fit for purpose; if you want to carry out some sophisticated computation, then it probably isn’t fit for purpose.

Garvin’s *Product-Based View* looks at inherent product characteristics, for example build quality. The Subaru Impreza came out on top for mechanical, electrical and bodywork

[1] D. Garvin, “What does Product Quality Mean?”, *Sloan Management Review*, Vol 4, 1984.

reliability in a recent J D Power survey for BBC's Top Gear programme. Some have argued that building in quality in this way can ramp up the overall manufacturing cost. Others, such as Philip Crosby, have argued that that by putting in the investment during manufacture, savings can be made later by not having to fix defects (costs of failure) and from the enhanced reputation that such a product achieves ("Quality is Free"). This is a particularly important view for software developers because of the high costs of software maintenance.

Recent surveys suggest that developers count on the 80:20 rule for estimating effort required for software maintenance: 20% of the effort goes into developing a system, and 80% into its maintenance (Pfleeger[2]). Pfleeger goes onto report that in a survey carried out by Lientz and Swanson[3], 25% of the effort of software maintenance goes into preventive and corrective maintenance.



In other words as much effort goes into trying to correct and prevent errors after a piece of software goes into operation as went into designing and constructing it in the first place. If the software has been built to a high standard in the first place, we should save on maintenance costs later, not to mention saving on the costs of failure, for example the $500M of the Ariane 5 disaster.

If the *Product-Based View* of quality asks, "are we building the product right?", the *Manufacturing View* asks "are we building the right product?" The *Manufacturing View* is concerned with conformance to specification. Using our example of motor cars, we have to ask whether the car is built to a specification. Of course every manufacturer will have a specification for their products if for no other reason that it would not be possible to build a production line to assemble them without one. All Formula 1 racing cars have to conform to a very detailed specification (see http://www.fia.com/homepage/selection-a.html for details). This specification is rigidly enforced by FIA stewards and teams will be penalised if they deviate from it in the slightest detail. It should be noted, however, that mere conformance to

[2] Shari Pfleeger, *Software Engineering: Theory and Practice*, Prentice Hall, 1998.
[3] B. P. Lientz & E. B. Swanson, "Problems in Application Software Maintenance", *Communications of the ACM*, 24 (11): 763 – 769, 1981.

specification will not result in products with higher reliability or fewer faults. I’m sure that we can all come up with examples of cars, and other products (including software products) which have poor reputations.

Whereas the user of a system will largely be interested in the *User-Based View*, the client (the person who pays for the development) will be more interested in the *Product-Based* and *Manufacturing Views*.

Garvin offers one other view of quality: the *Value-Based View*. This is about providing as much quality as the customer is willing to pay for. In the car industry this is of major concern to manufacturers of the smaller mass produced cars. Many manufacturers allow customers to specify their own cars. For example, ABS required, but not electric windows. In software terms this is like buying the standard edition of a software package, rather than the “professional” edition with all the bells and whistles. Some car models have a very good reputation for value for money, for example the Nissan Micra. Others’ reputations, for example, Lada, although inexpensive, have not fared so well, and are still regarded as cheap and nasty.

Surprisingly perhaps, Formula 1 engines are built to survive the Grand Prix race and no more. Building a very strong engine can increase its weight; in order to make the cars competitive, they must be lightweight, but making the engine too light weight might result in an increased danger of an engine blowing up. The F1 constructors have to find the compromise between engine performance and reliability over the length of a Grand Prix race.

In the software industry, the *Value-Based View* manifests itself in the eternal triangle of cost versus functionality versus time to deliver. What this tells us that is that it is possible to satisfy two of the three factors, but not all three.

The *Product-Based* and *Value-Based Views* present us with another dilemma: do we build up to a standard, or down to a cost? Much will depend on the type of application (safety critical, mission critical, general purpose, etc), the clients (budget, expectations, IS strategy etc), the users (expectations, expertise, experience, etc) and the development team (culture, experience, etc).

The *User-Based* and *Manufacturing Views* can be thought of as Explicit and Implicit system requirements. The explicit requirements are those things that the system must do, eg maintain customer records, move railway level crossing barrier up or down, add x and y, etc. Implicit requirements those desirable properties of software that the user and client take for granted, things they expect but don’t specifically ask for, eg wheels not falling off when you hit a pot hole in the road, spare parts still available after 6 months, paint not peeling off after a heavy downpour, etc. We will be discussing implicit requirements in greater depth in lecture 3.

Crosby’s View of Quality

Philip Crosby[4] argues that quality means “conformance to requirements”. So if a Ferrari 550 Marinello conforms to all its specified requirements, then it is a quality car. But then, by the same token, if a Lada conforms to all the specified requirements of a Lada, then it too is a quality car! What Crosby offers is an objective test of quality, but the problem for the software engineer is that much of the quality requirements of a piece of software are implicit or implied. For example, clients and users expect the software to be easily modifiable, they do not expect the software to behave as though its specification was written on tablets of stone.

[4] Philip Crosby, *Quality is Free: the Art of Making Quality Certain*, Signet Books, 1982.

Whereas when you buy a car, you have a much better idea of what to expect and many implicit requirements of cars, such as safety features and reliability are covered in law.

The problem seems to be that "quality" is still very much a subjective term, and we often apply the transcendent view, which Crosby rejects. Linda Macaulay[5] cites Kano et al[6]'s three types of requirement:

- Normal requirements: these are what we typically get just by asking clients and users what they want.

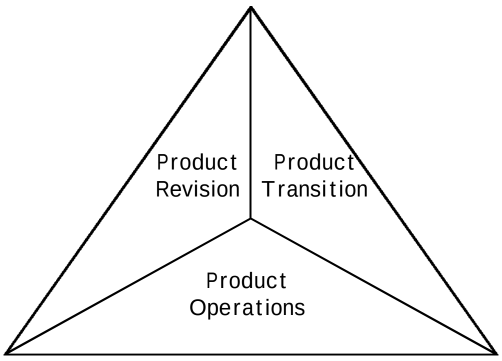
- Expected requirements: these are often so basic the clients and users may not think to mention them – until we fail to deliver them. Their presence in the system meets expectations, but does not satisfy customers. Their absence however is very disappointing. You would be very disappointed if you had bought some software that ran perfectly well under Windows 95, but will not run under Windows NT.

- Exciting requirements: The presence of these provoke the reaction "Wow. It even does this …". Features can succeed in satisfying clients and users so well that they boast about their software.

This view is closely related to explicit and implicit requirements discussed earlier. But what about the exciting requirements? If they had not been specified would that detract from the quality of the software? It depends – if you are the user they might well enhance your opinion of the software, but if you are the client and are having to pay for something you didn't ask for, probably it would diminish your opinion of the software.

McCall's Model of Quality

Another model of quality offered by McCall considers software quality from three perspectives (see figure below).



<u>Product Revision</u>

McCall considers three dimensions to this perspective:

- Maintainability - *Can I fix it?*
- Flexibility - *Can I change it?*
- Testability - *Can I test it?*

[5] Linda Macaulay, Requirements Engineering, Springer, 1996
[6] Kano, N., Seraku, N., Takahashi, F. & Tsuji, S., "Attractive and Must-Be-Quality" (in Japanese), *Hinshitsu*, Vol 14, No 2, 1984.

Product Transition

Again McCall considers three dimensions to this perspective:

- Portability – *will I be able to use it on another machine?*
- Reusability – *will I be able to reuse some of the software in other applications?*
- Interoperability – *will I be able to interface it with other components?*

Product Operations

Here McCall considers five dimensions to this perspective.

- Correctness – *does it do what I want?*
- Reliability – *does it do it accurately all of the time?*
- Efficiency – *will it run as well as it can?*
- Integrity – *is it secure?*
- Usability – *is it easy to use?*

Clearly each of these perspectives is important for software quality, but in this course we will be mainly concentrating on Product Revision. We will be discussing software quality in an environment where requirements are not clearly understood by the user and where a prototyping development life cycle is used. Code will need to be easily created, changed, tested and fixed – hence our concentration on Product Revision. It should be added, however, that we will also be considering the other perspectives from time to time.