FEABHAS

Understanding Quality

Glennan Carnie

Understanding Quality

Quality has become a bit of a catch-all word. How often do we hear comments like:

"We must improve our software quality"

"The first thing that always gets sacrificed is quality"

Glance through a typical requirements specification, project plan, test specification or quality plan and you'll see them littered with the phrase "high-quality software".

But what does this *actually* mean?

"Quality" is one of those definition-free words that everyone *thinks* they understand, and bandies around liberally to convince others they are a conscientious and rigorous engineer. Without having a true understanding of "quality" – that is, *what* it requires you to do, *why* you should do it, and the *consequences* of doing it – it remains a meaningless buzzword.

Defining 'Quality'

Let's start with a simple definition. Many naïve organisations take quality to mean "absence (or a very low number) of bugs" in their product. This definition, though, ignores customer satisfaction or engineering rigor; both of which must surely have bearing on the "quality" of your product.

So what do the 'Quality Experts' think? Surely as 'experts' they must have a definition?

ISO 9000:

> *Degree to which a set of inherent characteristic fulfils requirements.*

Genichi Taguchi (developer of Tagachi methods for manufacturing):

1. *Uniformity around a target value.*
2. *The loss a product imposes on society after it is shipped*

Peter Drucker (Inventor of the phrase "Knowledge Worker"):

> *Quality in a product or service is not what the supplier puts in. It is what the customer gets out and is willing to pay for.*

American Society for Quality:

1. *The characteristics of a product or service that bear on its ability to satisfy stated or implied needs.*
2. *A product or service free of deficiencies.*

It seems quality appears to defy a single consistent definition.

One thing all these definitions imply is that quality is beyond a mere system characteristic. It is both customer-facing and internal-facing; it influences both the product produced and the processes used to create the product.

For practical purposes four definitions of quality are required, inter-related but very different; and not necessarily mutually-inclusive. This model incorporates all the definitions given above.

Compliance Quality

Compliance Quality focuses on conformance to specification. That is: how many of the (formally stated) requirements does the product fulfil? The more requirements the product complies with the better quality it is.

Engineering Compliance quality

Whatever development process you choose to adopt – be it waterfall, iterative, agile or even ad hoc – you must have methodologies (that is, guidelines, techniques and practices) for the following:

> **Synthesis** – how, when and why you create and transform project artefacts such as design documents, source code, test files, etc.
>
> **Verification** – how you demonstrate traceability; how you demonstrate conformance.
>
> **Validation** – the techniques you use to ensure you are solving the correct problem in a sensible fashion at each stage; that your thought process is valid.

Compliance Quality is *verification-oriented* quality. Many (if not most) processes focus on Compliance quality as it is relatively easy to measure, control and enforce. Testing and compliance matrices are the classic tools for measuring Compliance quality in a product.

The obvious(?) drawbacks of Compliance quality are that it places few demands on synthesis or validation methodologies. It is assumed that the requirements specifications are always complete, correct, consistent and unambiguous. Practical experience tells a very different story: if specifications even exist (and this is far from guaranteed on many projects!) they are often less-than-excellent – patchy, inconsistent, unintelligible, and unhelpful.

The value of Compliance quality

Customers typically expect a high level of compliance to their specifications and are not likely to pay more for better compliance.

Customers rarely penalise for failure to comply with specifications (although it's not unheard of); but such a practice is likely to lead to loss of future business.

There is a business model that focuses on Compliance quality with the implicit knowledge and expectation that requirements will be poor and/or incorrect. By delivering *exactly* what the customer asks for (rather than what the customer actually needs – see later) the development organisation can claim the 'moral high ground' and charge the customer for post-delivery 'fixes' to overcome all the mistakes in the original specification.

Perceived Quality

Let's imagine a scenario: You've won a competition (congratulations!). The prize is a C-segment car. You have a choice of vehicle: A Mercedes *C-class*; or a Daewoo *Nubira*.

Which do you choose? And *why*?

Most people select the Mercedes. The reason given is often “because it’s worth more money”. When pressed as to why the Mercedes is worth more money the answers trend toward:

- It’s better built
- It’s more reliable
- Brand recognition
- More desirable styling
- Mercedes is an aspirational marque

If the customer can see, smell, feel or taste the results of engineering effort we call this 'Perceived quality'. For example, when buying a car, the one with the more lustrous paint, deeper carpets, softer leather, more sound deadening and fewer squeaks and rattles we tend to think of as 'higher quality' than another, similarly functioning car.

In software terms Perceived quality typically manifests itself through the user interface. A product with a slick, glossy user interface is often considered a ‘better’ product than one with a clunky, cumbersome, or just plain ugly interface.

Perceived quality is *sensual* quality; and is often quite subjective.

Engineering Perceived Quality
Perceived quality is a synthesis activity. Aesthetics, ergonomics, layout, materials selection, etc. are all important elements of the specification and design of a product.

Perhaps more importantly, though, is establishing just what constitutes Perceived quality *to your customer.* What is a key element to one customer may be seen as superficial or unwarranted to another. The collection of key product attributes I sometimes call the product’s ‘handshake’ – that is, the set of properties that make an impression (positive or negative) on first acquaintance with the product.

The value of Perceived Quality
A customer will often use Perceived quality as a differentiator between similar products; and often pay extra for it.

One curious effect of Perceived quality is that the customer will often believe a product functions better, has fewer flaws, or is more useable than a less-well-perceived, but otherwise similar product.

Intrinsic Quality

Intrinsic Quality is the inherent 'goodness' of a system. That is, not what the product is/does, but how well it has been designed and constructed. If you like, Intrinsic quality is a measure of the engineering rigour that has been put into the product.

In the case of software-oriented products Intrinsic quality tends to manifest itself in architectural robustness and resilience to change.

It is challenging to define what makes ‘good’ software since we so rarely see it. On the other hand most engineers are used to seeing (and maintaining!) what they consider ‘bad’ software. Often

engineers cannot qualify what makes the software bad; it just is. Typically, though, bad software suffers from one or more of the following maladies:

It is difficult to understand.

The code is difficult to read; variable names are unhelpful; comments are missing (or worse, incorrect); code is badly structured and laid-out; etc.

It is fragile.

Fragile code will break at the slightest provocation. Fixing a bug in one part of the code can cause a cascade of other bugs in other parts of the code; often quite un-related to the problem.

It is rigid.

Rigid code has been designed to fulfil one requirement – and that is all it will ever do. If the requirements change the software must be almost completely re-written to support the new behaviour.

It is immobile.

Engineers often build really useful software modules. However, if the useful module is so tightly intertwined with the structure of the program it's in, the effort involved in unpicking the useful code is so great is often impractical to even try to re-use it somewhere else. Sometimes it's just easier to write it again from scratch.

Engineering Intrinsic Quality.

Intrinsic quality is a synthesis activity; or more correctly, it is the amount of rigour (and effort) we put into specifying, designing, and implementing our products.

If the above qualities define bad software then the opposite must define good software. Thus good software could be defined as:

Comprehensible

Good software must be easy to read and understand. There should be no surprises in the code; its purpose and implementation should be obvious and equivalent.

Flexible

It should require little effort to adapt existing software to new requirements.

Robust

Software should be tolerant to invalid inputs. Performance should be unhindered by incorrect inputs, or should be degraded in a known, controlled, way. The impact of change should be controlled and limited.

Mobile

Software should be easily adapted to new environments, platforms or problem domains, without requiring major modification.

The value of Intrinsic Quality

Intrinsic quality is closely allied to the idea of *Technical Debt*. Technical debt is the name given to the concept that technical flaws, omissions, structural weaknesses, etc. in a product, if not fixed immediately, will cost considerably more to fix in the future; and the longer the debt remains, the more it will cost to remove it.

A company will put effort into the design and architecture of their systems to give them greater flexibility in the future. Engineering rigour, and designing for change and maintainability reduces (but cannot complete eliminate, unfortunately) the impact of technical debt. That is, the higher the Intrinsic quality of a product, the less it will cost to maintain, modify and extend it during its life.

Note Intrinsic quality benefits the development organisation, and is largely invisible to the customer; thus very few customers are willing to pay for such work. Intrinsic quality is therefore an upfront cost to the development organisation, which has to be balanced against the reduced future costs of product maintenance.

Fitness Quality

Finally, we have 'Fitness-For-Purpose Quality'. Typically, we buy products / systems because they either a) fulfil some aspiration; or b) solve some problem we have. Fitness-For-Purpose Quality is a measure of how well the system fulfils the NEEDS of the customer. That is, how well the system / product helps the customer meet their personal responsibilities/ aspirations.

Engineering Fitness Quality

Fitness quality is a validation activity. During validation we ask “are we building the right product?” Implicit in this is the question “what is the right product?”

Fitness quality is about understanding the needs of the project’s stakeholders and their *success criteria* – that is, those aspects of the product that make them ‘happy’ (or, if you prefer, those aspects of the product that, if flawed, will leave the stakeholder deeply *unhappy*). The more success criteria you can fulfil the better the Fitness quality of your product.

Of course, in any real project it is next to impossible to fulfil all the stakeholder’s success criteria. Many are unachievable, many are contradictory or conflict; in many cases the success criteria of a stakeholder are not reflected in the requirements submitted by that stakeholder. Fitness quality is therefore always an exercise in compromise.

The value of Fitness Quality

A product that fulfils a customer's needs is far more likely to be considered to be a 'high quality' product than one that doesn't meet their needs. Or, conversely, a product that doesn't meet your needs - irrespective of how nicely constructed it is! - will not be as satisfying as one that does.

Although Fitness-For-Purpose Quality doesn't necessarily lead directly to higher sales cost, it may lead to improved future sales. Products that meet customers' needs are far more likely to generate repeat business or brand loyalty than those that don't.

Focussing on Quality

All these different definitions of quality are inter-dependent and in any commercially-viable product all will be present to some degree: All products will satisfy the some (if not most) of their requirements; all products have *some* Intrinsic quality; all products will satisfy their customers to some level. The problems arise when companies attempt to *improve* their quality and over-focus on one aspect of quality at the expense of the others.

Putting the emphasis on Compliance quality

In many projects - for example, safety-critical systems - Compliance quality is a contractual obligation. In such projects the developer is required to demonstrate – via appropriate verification techniques – that they have applied due diligence during their development.

Projects with high Compliance quality requirements tend to cost a lot because of the amount of effort required to produce all the compliance evidence. This can be a problem if you are working in a price-sensitive market.

And of course it is perfectly possible to be diligent, systematic and rigorous and still produce a product that doesn't fulfil the customers' needs and is totally unmaintainable.

To avoid too much focus on Compliance quality you must understand exactly what your customers require in terms of evidence, which in many projects is remarkably little. And also recognise that a lot of design documentation is for the benefit of the customer, not the benefit of the developer.

Putting the emphasis on Perceived quality

The Perceived quality of a product is what makes it initially desirable. A customer is likely to choose a desirable product over a mundane product; and even pay more for it. (This is providing you give the customer the right desirable properties)

Over-emphasis on Perceived quality can lead to problems with 'style over substance'. Such products can end up lacking functionality or engineering rigour, meaning they can be difficult to maintain or extend, or may not be adaptable to new markets.

Remember the rule that 80% of a customer's needs are fulfilled by 20% of the system's functionality; not everything the customer asks for is needed or required to satisfy them.

Putting the emphasis on Intrinsic quality

Engineers like to do a good job. It's what motivates them. Without control, though, there is a tendency for them to 'gold plate' their systems, building in flexibility and extensibility where none is required; and often at the expense of customer functionality or compliance rigour.

This is nicely illustrated by one of the stories from Geoffrey James' "*The Tao of Programming"*:

> *A novice programmer was once assigned to code a simple financial package.*

> *The novice worked furiously for many days, but when his master reviewed his program, he discovered that it contained a screen editor, a set of generalized graphics routines, an artificial intelligence interface, but not the slightest mention of anything financial.*
>
> *When the master asked about this, the novice became indignant. ``Don't be so impatient,'' he said, ``I'll put in the financial stuff eventually.''*

The key to avoiding over-emphasis on architectural factors, or 'Death by Abstraction', is to understand the technical risks to your project. Not every part of your system will change in the future; some functionality may never change. The technical risks of the system should identify these potential areas of uncertainty, and the design should take these factors into account.

Putting the emphasis on Fitness quality.

Fitness quality probably has the most impact on a customer's impressions of your product: Compliance quality is about delivering (and demonstrating) a specified set of functionality and system qualities; Intrinsic quality ultimately benefits the development organisation (which may, or may not, be passed on the customer) by building more flexible and robust architectures; Perceived quality makes the product desirable; but Fitness quality is about making the product *useful*. When all the other quality measures are forgotten people want useful products that make their lives better in some way.

Fitness-for-purpose will have a massive impact on the Perceived quality or a product. Poor utility can shift perception of a product from 'excellent' to 'too much style over substance' very quickly.

In many ways Fitness quality is the polar opposite of Compliance quality. Compliance quality focuses on satisfying the needs of the process; Fitness quality focuses on satisfying the needs of the customer.

Agile development evangelists have long argued that process-driven approaches to development have consistently failed to produce useful products. They also argue that we have been very poor at capturing the needs of our stakeholders in requirements documents. Agile methodologies, in response, have emphasised customer satisfaction at the expense of documentation.

If you are not careful you can end up over-emphasising Fitness quality at the expense of other quality measures.

Architectural robustness (Intrinsic quality). 'Organic Architecture' (to coin the phrase used by the Agile community to describe the ad hoc development of a system's large-scale design) can lead to rigid, unmaintainable systems, wholly unsuited to long life-cycles.

With few, or no, formal requirements it may be difficult to demonstrate compliance.

Ad hoc, incremental developments may lead to uncohesive, scrappy designs that may affect future customer's perception of the system.

Which quality is right for me?

The key to quality is to understand the impact different qualities have and also understand which are important to your product and business.

In other words: *Quality is a business choice*.

For most commercial projects Fitness-For-Purpose quality is king – if you don't satisfy your customers you won't sell products, and that's the key to staying in business. In safety-critical systems, Compliance Quality supersedes all – it is vital that every effort is made to demonstrate that all possible care was taken when developing the product. Rarely in software does Intrinsic Quality dominate; in fact the opposite is more likely true. Intrinsic quality is an overhead cost that many companies do not wish to bear. However, it is worth understanding that most software projects spend between 75% and 95% of their life in a maintenance phase. If your product will be in service for a long time it may well be worth investing in its architectural robustness.

Conclusion

Quality therefore forms the framework upon which your entire business is based. How you fulfil the demands of the various types of quality you make business-level decisions that affect the whole culture of your company.

It's that important. And it's why you *must* understand it.