What is a High Quality Software System?

by John Brinkworth

This article first appeared in

SQM issue 27.

There is a growing focus within the software quality community on processes and process improvement. In this article I take a complementary approach, namely a product one. What are we trying to create by improving our processes? The fundamental answer is higher quality software, but what is a high quality software system? Most SQM readers would consider they have an intuitive feel for this but how can it be identified and structured? The aim behind this article is to offer a fresh perspective on the different aspects of a high quality software system.

The straightforward definition of a high quality system is that it meets its users' requirements. Requirements can be classified in a number of ways. There is the classic split into functional and non-functional. Functional seems almost too easy to state - the system does the tasks the users ask it to perform. Non-functional yields a longer list - they are the attributes that have a bearing upon the system's ability fulfil its functional requirements.

There is also a split between user-transparent and user-opaque. A user-transparent feature is one that is directly obvious to a user and thus one that they would naturally consider when assembling their requirements. Conversely a user-opaque one would not be included by a user without prompting from a developer, and indeed such requirements may not strike the user as being important at all. Indeed, sometimes they seem irrelevant, sometimes self-evident and sometimes only important if they are absent.

The features are described in this article in no particular order, since there can be no precedence or absolute minimum. Definitions here are given in their narrowest sense, yielding a fine granularity of features. Other equally valid taxonomies might class some features as sub-features of others. Every system is different and will need a unique mix of features. No system will exhibit all of them, not least since some may conflict (most obviously efficiency can be affected by a number of features such as maintainability, security, auditability etc.).

I describe and categorise the features below, under four general headings relating to functionality vs. non functionality and transparency vs. opacity.

Functional: User-Transparent

Core Functionality - The system must perform the tasks its users require and perform them correctly. This feature can be augmented by requiring the system to perform the tasks elegantly rather than clumsily, as a simple example data should only need to be input once. The core functionality may include a requirement to interface to other systems.

Self-Checking - For certain tasks it may important that their outcome be assured to be correct. In such situations a system may need to have the ability to check that it has performed a task correctly, for a calculation this might involve doing it in two different ways and raising an exception report if the outcomes differ by more than an acceptable tolerance.

Fail-Safe - The self-checking facility may drive a fail-safe feature that prevents a system from executing subsequent steps if an error is detected in earlier processing. An example of this would be a system that uses checksums to determine the validity of data transferred into it from other systems and only processes the data if the checksums tally.

Safety-Critical - If a life rather than a livelihood depends upon a system, it must function in such a way that the safety of the individuals who rely on the system is always the paramount concern.

Behaviour Reporting - This can range from a system that doesn't report what it's done at all, through one that reports everything (overwhelming any user or operator), to one that reports on an exception basis. A mixture of on-line exception reporting with full reporting to files can give the most effective and comprehensive service.

Terminology - The use of specific workplace jargon and terminology is not a problem provided users understand it or, if necessary, are trained in it. However, if terminology is used inconsistently within the system, or between the system and others it interfaces to, then the likelihood increases that users will find the system difficult or confusing to use.

Forgivingness - The system may need to include the capability to reverse an action done in error. Such an action would probably need authority from a more senior user before being executed, and would need to be fully audit-trailed. At the extreme there may be a need for the system to be idiot proof, for example a touch-screen point-of-sale terminal in a public location like an airport may not have a keyboard visible thus not giving anyone the opportunity to see what happens if they enter CTRL-C.

Clarity - This is closely related to the elegance of the design. A well designed system will support the users' intuitive understanding of the tasks they are working on. From the users' point of view the components of the system should cleanly map onto the tasks being performed by the users. Navigation between the components is likely to reflect the flow of work the users follow when performing the tasks. It should not be artificially restrictive, neither forcing users through unnecessary screens on the way to the one they want, nor forcing users only to take one route. The system should allow for different experience levels and different ways of performing the same task where appropriate within the business.

Functional: User-Opaque

Data - The integrity and quality of the data have a fundamental bearing on the benefits the organisation using the system will derive from it. A functionally correct system that is loaded with incomplete, inconsistent, or corrupt data will give erroneous results. For some systems the manual checking of such data by the users may be adequate, others may prefer to incorporate the fail-safe features mentioned above.

Operability - If the system is one that has to be controlled by a Data Centre then the functionality they require also needs to be provided. The target is often minimum intervention, which tends to be achieved using features such as multiple levels of error reporting with each level of criticality highlighting itself more than the previous one. An addition to the logging of all major actions taken by the system and the operators, and a separate log of error messages, is the controlled, and regular archiving of such log files. There may also be a requirement for clearly defined operator procedures covering all eventualities, not just normal running but backups, troubleshooting, controlled stopping and restarting, and disaster recovery.

Testability - A system with a clear definition that enables effective tests to be derived stands a greater chance of being error free. In addition, a system that facilitates testing by providing full audit trails of its actions increases the number of tests that can easily be performed, so increasing the chances that most errors have been found before it goes into production.

Non-Functional: User-Transparent

Performance - This is the ability of the system to perform its tasks within the time frame required by its users. Tasks might be screen-based response to data input, or the scheduled generation of an overnight report, or the control of some item of hardware (such as the flaps of an aeroplane or a valve in a water treatment works). The granularity of the requirements may be quite fine within a system, some elements needing to respond very quickly with others being less time-critical.

Capacity - Whereas expandability may be overlooked by users unless they are prompted, capacity is normally an issue that will come to the fore. Capacity is the ability to handle the initial expected data volumes and processing workload.

User Interface - This can be assessed by the extent to which the system forces its users to work in a way that is not effective for them. High usability involves ensuring that the system uses the right interface devices (same screen and keyboard as the user already uses for accessing other systems, mouse, hand-held device, touch screen etc.). It is closely related to the functional attribute of clarity, however a system can be clear in its structure but still poor in the quality of its user interface.

Assistance - The existence and the quality of the documentation that the user needs will directly affect the utility of the system. The availability of the documentation will have an impact - on-line help is in many cases more convenient than paper based manuals, for one thing its far less likely to go missing. If available the documentation will also need to be comprehensive but segmented in two different ways. One way will segment it to address the different groups who have access to different parts of the system. The other way will segment it to address individuals who have different levels of knowledge and experience of the system, thus offering various ways of learning about the system. In the ultimate case assistance for a system can be extended to include the provision of a support team available whenever the users need help.

Availability - This relates to when the users can access the system, and may be influenced by their geographic dispersion (especially international or multi-time zone) if the system is multi-user. It can also be affected by the maintenance schedule of the system, whether it will need to be taken down for backups or archiving. Hardware maintenance will also affect the availability. There may be a need for a service level agreement to ensure that all parties are aware of when the system must be available. The system's reliability can be determined by measuring the extent to which its actual running meets the planned availability targets, that is how much time is lost to unplanned downtime.

Archives - If the system is one that retains details of all its transactions, then as the amount of data grows performance levels may be reduced and storage capacity limits reached. The system may need to include an archiving capability to remove data that is no longer required. This may completely remove transactions or may distil them into a more concise record retaining only the key elements. The removed data may be retained elsewhere: in a database that could be reloaded into a copy of the system; in a separate database of reports; as reports held in a document image processing system; or simply on microfiche. The degree to which detailed audit trails are maintained of the archiving operation will depend on the criticality of the data. As volumes of data that the system processes vary the time data is retained before being archived may vary. Equally the overall retention time for the archived data may be dictated by external agencies.

Security - This is the ability of the system to allow only genuine authorised users to access its information and features. Requirements can range from minimal for packages installed on a standalone PC to stringent for safety- or business-critical systems. It should be remembered that security can also encompass physical measures that control peoples' access to a system's user interfaces and hardware components.

Non-Functional: User-Opaque

Portability - There may be a need for the same system to run on different platforms, however this requirement more often arises from a technical/marketing viewpoint for a product when a vendor wishes to make the same product available to as much of the market as possible irrespective of the hardware they use.

Efficiency - This is the user opaque equivalent of performance. If there is a requirement for particularly high performance the system will have to make very efficient use of the resources at its disposal (such as: processor, main memory, swap memory, disk space, etc.).

Safety - There is a fundamental requirement that the system be safe in a physical sense. That is, there are no problems with the overloading of electrical equipment, there are no exposed wires, and no cables providing trip-hazards. Health and Safety legislation may also need to be considered.

Expandability - The underlying internal design of a system will determine how expandable a system will be. At the start of a system's lifetime it is not easy to predict the growth in the volumes of data to be processed, or numbers of users that may require access to a multi-user system. The modularity of the components, be they processors, logical and physical storage units, or multiple instances of the same process, will determine whether the users will be constrained by limits that are arbitrary and avoidable. The constraint can be absolute, e.g. a maximum number of users since all access routes are in use, a maximum size to a database table, or relative e.g. no more than a given number of users since the response times will degrade to an unacceptable level.

Configurability - There are two aspects of configurability that may be present in a system. It may come in different versions for different users at the same installation. That is different users see different parts of the system when running the same executable - this is very closely related to access rights, security and permissioning. The other aspect of configurability is that of a system which is a product installed in more than one location with different features at each location. In this case changing the functionality users to which have access may involve just redefining a parameter whilst the system is still running, or stopping the system and rebuilding it with different configuration options, or a completely fresh installation from the developer.

Maintainability - This is a measure of the system's amenability to change. A well designed system will be quicker to fix-meaning that the time from discovery to resolution for a problem will be minimised. If it is known at the outset that further enhancement is likely then a high quality system will be as modular as possible so as to minimise the number of changes needed to add extra features. This reduces the amount of time taken to enhance the system, making its development life cycle more responsive to the needs of the users. Maintainability also encompasses the ability of the system to carry on running whilst an upgrade or a fix is installed.

Reusability - Sitting closely with maintainability, this again is a reflection of the design. There may be a requirement that certain aspects of the system's processing will be needed in other systems. The system thus should be developed in a manner so that the elements can be easily reused. This is met naturally by using an object oriented approach, but can also be done to a degree when using a well modularised, structured design.

Conclusion

The timing of the system's delivery is also critical. Many systems support enterprises which are targeting particular windows of opportunity. Systems development involves prioritisation and compromise. To echo DSDM's Third Principle - a fully featured system that is available too late is of less utility than one that meets the high priority business requirements and is available when the enterprise needs it.

Most users really don't care about the details of how software is created, they just want it to help them on the job.

We currently tend to judge software quality from a process perspective, assuming that the right process will result in appropriate quality software. We should not loose sight of the product perspective. We need to apply critical thinking and consider not only how we put a system together but whether its any good when we've finished.

The author

Articles index

QiC Index

Tesseract index

About these pages

URL for this page: http://www.avnet.co.uk/tesseract/QiC/articles/Brinkworth/27.html

Last updated 8th February 1996.