Cmp 416

Quality

Quality of software systems(planning and evaluating)

How many times have you been working on your computer, and something's gone wrong? The system falls into a go-slow and starts taking an age to do anything, your word processor crashes, the whole computer stops responding... or perhaps you can't find something you have worked on for hours, or you accidentally delete or change something and can't get it back... and then there's the endless updates and patches...

Software is often not tremendously robust - in fact, it is often full of bugs. If a more tangible product, a washing machine, say, behaved like that, you'd take it back to the shop and demand your money back! However, it is the nature of software that it is very difficult - some say impossible - to release it 100% bug-free. So how can we, as developers, manage quality in software projects?

Quality management is a term much used in industry. Whole disciplines and methodologies have grown up around the search for "quality". You may have come across quality assurance, Total Quality Management (TQM), "Right First Time" and quality management systems certification such as ISO9000. The approach taken in engineering and manufacturing has been extended and adapted over time to cover software engineering.

In this, we consider what is meant by software quality, how you can define, measure and manage it, and the specific issues of quality in web software

Who defines quality?

There are many definitions of "quality". Some say "quality is delivering what the customer asks for", and most traditional approaches seek to ensure quality by specifying the requirements of a product or system up front, monitoring interim systems during the development, and then checking what was achieved at the end, against the initial requirements.

If a customer or user is then dissatisfied with the system, the developer can always claim to have fulfilled their obligations, providing they have supplied what was originally required. The user, however, may feel that the original specification was inadequate.
Whose fault is that? The user's? Or the developer's?

We have seen in the methodologies studied in earlier units how it is difficult to pin down requirementat the start of a development. If this is a difficult task for the professionals, how much more difficult is itfor users?

Professionals need to be able to evaluate the quality of their system independently of what their customers say. And, more than this, they need to consider quality from different perspectives, considering not just the customer perspective but the supplier and professional perspectives.

Developer's responsibilities

1. Adjusting customer expectations:

It is essential to find out from potential users what their needs and

expectations are, but it is the responsibility of the developer to adjust those expectations to ensure that what is proposed is feasible - that it can actually be achieved within the particular constraints of the situation.

2.Considering the future: Systems introduced into organisations have future consequences for that organisation. Systems should be designed taking into account potential maintenance issues, future

development, compatibility with technology as it evolves in the future, and how useful the technology will be as organisation evolves.

These are heavy responsibilities, but they should not be avoided. Defining quality is more than just

deciding what a system should do and if it does it!

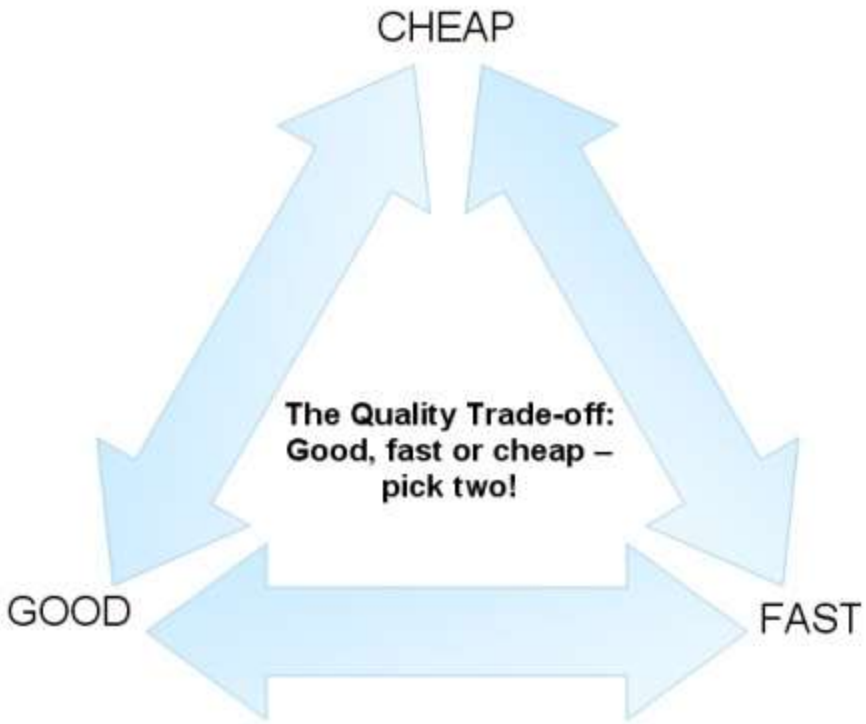
Quality issues

- **Quality issues**

A. Quality - The quality of an information system can be viewed from different standpoints:

- The quality of the system viewed as a technical object - does it work? Does it perform the functions it is designed for?
- The quality of system as a useful tool for its users - does it help the users do what they want (regardless of what it was designed to do!)
- The quality of the development process producing it - was it produced according to the quality plan? Was it a useful learning experience for the developer?
- The quality of supporting materials produced by the development team during the course of the development - project proposals, specifications, prototypes and so on.

Criteria - the criteria used for judging quality often conflict with each other.



An example often used is shown in the figure above. This shows the need to choose what is most important for a system, high quality, low cost or rapid delivery. It is generally possible to deliver two of the three, but not all three. A good, cheap system will take a long time to produce; a fast, good system will be expensive; a fast, cheap system will be less functional in some way - it may have fewer features, or be less efficient.

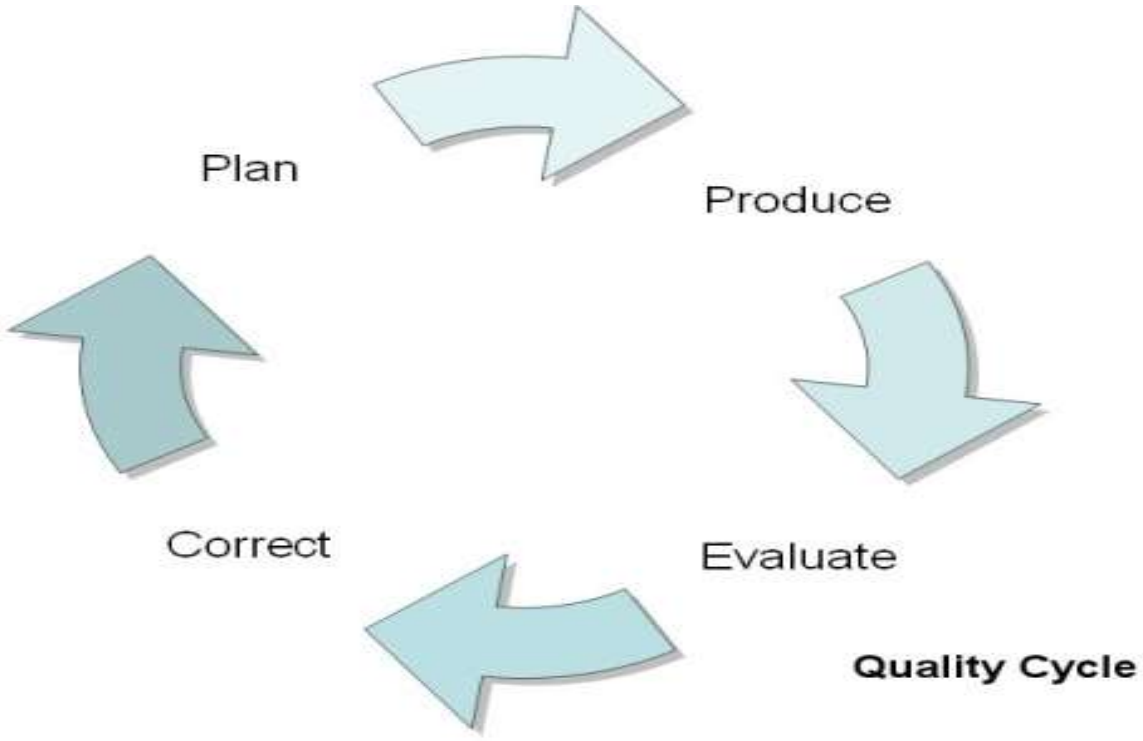
- In practice, there will need to be a trade-off between the various ideal quality requirements.
- Other trade-offs may include efficiency versus maintainability, or reliability versus speed of delivery.

B.Perspective - Different perspectives lead to differing priorities, which lead to different judgments.

- The quality of a system is judged differently by different people, so agreement on what constitutes quality must be negotiated. This negotiation will not be a one-time process. Parties may be renegotiating quality throughout the development, sometimes overtly, such as in review meetings, sometimes covertly, by their attitudes and actions.
- Short timescale projects, such as are typical in web development, have a better chance of success if the customer is willing to negotiate quality.

Planning for quality

- The general aim is to control the quality throughout the development process by using specific methods to plan, improve and control the process.
- One of the best known approaches to quality control is the "Deming Cycle". This is an iterative cycle of actions to be taken at each stage in a development. The 4 steps in the Deming Cycle are: Plan-Do- Check-Act, adapted by Dahlbom and Mathiassen (1993) as: Plan-Produce-Evaluate-Correct.



The Quality Cycle

- PLAN: Each process in the development starts by planning for quality. This means deciding at the outset what the objectives and required outcomes are, what procedures and methods will be used to achieve them, and how they will be evaluated at the end of that process. (This is the same approach for the overall development as well as for the constituent parts.)

- PRODUCE: implement the plan. The quality of the work can be improved during production by use of tried and tested methods and methodologies - for instance, those considered in units 1 and 2. Various inspections are also carried out to provide performance and status information for the evaluation.

- EVALUATE: the information and measurements taken are used to evaluate the output against the objectives, using objective measures where possible.

- CORRECT: corrective action is taken depending on the outcome of the evaluation.

The whole cycle then begins again with the next stage or process.

Benefits/difficulties of the Cycle

- *Benefits:*
- Simple cycle, easy to understand.
- Iterative process, allows for unclear requirements at the start to be refined later.
- Well tried and tested.

- *Difficulties:*
- Deciding the objectives may be difficult. People may set objectives to be easy to meet rather than what's actually needed.
- May be difficult to integrate with certain methodology e.g. RAD
- People sometimes don't like being "judged", even against objective measures

Evaluating quality

- There are many appropriate procedures for evaluating quality. It is useful to have a mixture of quantitative and qualitative measures, with objective and subjective observations, as appropriate.

- **Use of metrics**

  Metrics are the various parameters or ways of looking at a process that is to be measured. Metrics define what is to be measured. General qualities can be split into more specific, measurable, attributes. Examples include efficiency, cycle time, customer satisfaction, and number of faults logged. Using metrics has the advantage of helping to make the measurement more objective. It may be, however, that measuring still requires some subjective judgment - for instance, customer satisfaction can be "measured" by assigning points from 1 to 10, but the customer has to decide how many points to select using their (subjective) judgment.
- **Reviews**

  A formal technique, the reviewer(s) use their experience to evaluate the system against formal criteria. There will usually be a review meeting where the evaluations are discussed and acceptance or corrective action is agreed.
- **Proofs**

  Another formal technique, the idea of proofs has been "borrowed" from mathematics. In a software context, a program is proved to perform as defined in the specification.
- **Experiments**

  This is a planned evaluation of a system under the same conditions as those in which it will eventually be used. It is similar to prototyping. The judgment will be objective, with some subjective judgment also possible.

- **Example of using a quality tool to measure customer satisfaction**
- http://urlm.co.uk/
- Find out if there are other sites that measures quality.

Quality factors in software systems

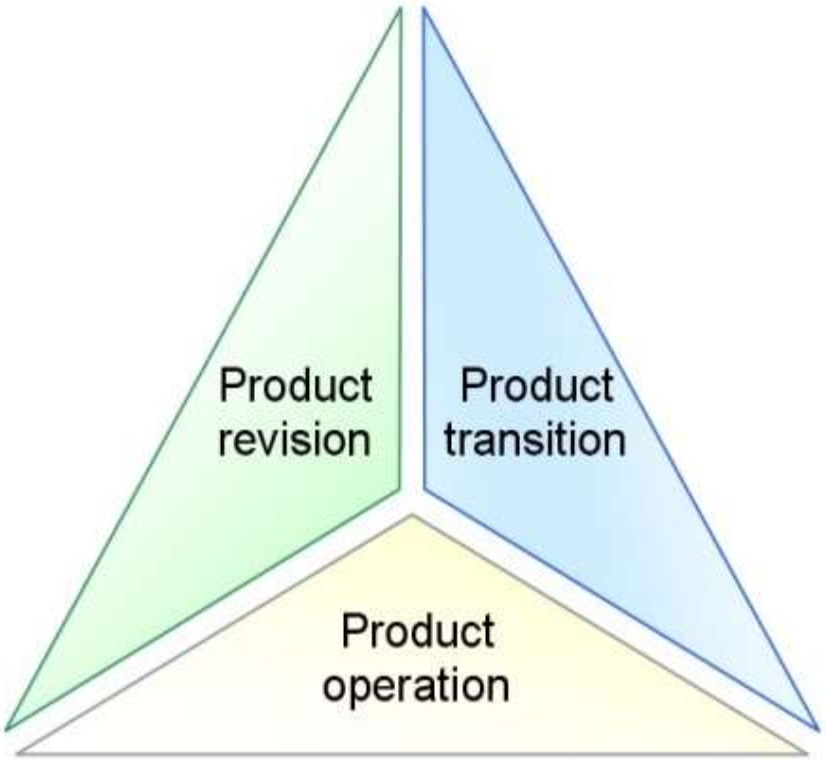
Three aspects of software quality

- In previous slides, we looked at responsibilities for quality, planning, and how quality can be evaluated. But what is "quality"?
- In the 1970s, Jim McCall recognised that users and developers had conflicting views on quality, and in 1977 he produced a model of software quality, for the US Air Force, which aimed to bridge the gap. Of course, the web had not been invented in the 1970s, but as you read the following explanation of the different elements of McCall's model, you should consider whether, and how, they are relevant to web applications

.

- McCall's model has three aspects:

- Product operation - concerns the usability of the software;
- Product revision - concerns future modification of the to meet new requirements ;
- Product transition - concerns transforming the software to work in new environments.
- These are shown diagrammatically below.



Quality Factors (McCall)

McCall's Quality Factors

- **Product operation:**

This includes aspects of correctness, reliability, efficiency, integrity and usability of the software.

- *Correctness is a reflection of how complete and consistent the software is, and how it matches the* specification and does what it is supposed to do.
- *Reliability also covers consistency, but also how tolerant the system is to errors and how it recovers* from unexpected inputs. A simple program will usually be more reliable.
- *Efficiency - an efficient program will execute more quickly and concisely than an inefficient one. This* factor is to do with efficient use of the system technology resources.
- *Integrity - this is a measure of how the system resists unauthorized access.*
- *Usability reflects how easy the program is to learn and to use.*

- **Product revision:**
- *Maintainability - this includes how easy it is to locate and fix faults, whether the system is self documenting,* and what general maintenance and housekeeping tasks (back-ups, database compression etc) need to be performed to keep it running well.
- *Testability - how easy is it to validate the system? Programs that are well-structured, simple, modular*

  and self-documenting are generally easier to test than those made of "spaghetti" code.
- *Flexibility - factors relating to how easy it is to modify the program to meet changing business needs. It* includes scalability (for instance how easily the program can be extended to cope with more users).

- **Product transition:**
- *Portability - how well the software can be transferred from one environment to another - for instance to* another operating system or hardware configuration.
- *Reusability - similar to portability, this aspect relates to how well the existing software components can* be reused in a different context. This could include reusing functions such as search algorithms or user input forms.
- *Interoperability - the extent to which the system can work alongside and with other systems. This* requires the system to use common protocols for communication and data structure, and is helped by using software and hardware standards

Three types of quality

- Dahlbom and Mathiassen (1993) look to a wider definition of quality. Quality is judged by people in three categories:
- Functionality - does the product do what it is supposed to do?
- Aesthetics - does the product look good?
- Symbolism - what does the product mean to us, or signal to others?
- These aspects are easier to understand for a software system by first looking at a different type of product.
- When someone is shopping for a pair of training shoes, they may choose them for different reasons.

- *Functionality - someone who wants to use the training shoes for running round an athletics track will* primarily be concerned with functionality. They want the shoes to be comfortable to wear, fit correctly, provide the right amount of support, flexibility and cushioning, and to be the right weight - heavier for training, lighter for racing.

Given a choice of a few pairs that meet all the functional criteria, they might then go on to consider the other factors, but functionality is likely to be most important in defining what a quality trainer means to them.

- *Aesthetics - someone may buy trainers not for fitness work, but for everyday wear. Beyond the basic* functionality of comfort, they are likely to choose them based on their appearance. Why else would people buy trainers with flashing lights in their soles when they walk? This effect is even greater with other types of footwear - party shoes for instance.
- Choices based on aesthetics are entirely personal - there are fashions and trends of course, but these then start to move into the third area...

- *Symbolism - this relates to what the object symbolises, to us and to others. The person buying the* training shoes may decide that the most important aspect of quality for them is the brand name on the trainers, even if they are uncomfortable or ugly - because everyone else is buying them.

Quality on the WWW

- The preceding sections have addressed quality in general terms, focussing then on specifics of software development and considering some aspects of quality as they apply to websites.
- In this section, we will look at research by Jeff Offutt (2002), which looks at how the unique challenges of web development make quality priorities different from the priorities in traditional software developments.

The Web then and now

The World Wide Web started life as a means of presenting information to people using simple web pages, which were mainly text documents linked together. Sites displayed information to visitors, and were often just electronic versions of brochures or catalogues. A site could be looked after by one person - the webmaster. Sites were hosted on simple Web servers, which sent the information to and from the visitor's browser.

Over the last half-decade, the internet and the Web have evolved almost beyond recognition. Web sites now run large-scale software applications, and provide entertainment, collaboration, e-commerce and many other activities. Applications consist of many different components, programs, HTML files, databases, images, scripts and more.

We no longer talk about visitors, but users. Users interact with the sites using diverse hardware and software, including mobile computing devices such as hand-held PDAs and mobile phones. The software is often distributed geographically, residing on numerous different servers. Engineering and maintaining a site now involves large teams of people with a variety of skills.
Web site engineering is now one of the largest parts of the software industry.

New discipline - new tools?

The characteristics of the new Web can be summed up in one word: "diverse".

This means that the developing discipline of web site engineering must have a whole range of tools and systems to build good quality, secure, applications - a point that is not well-understood in many cases.

Existing software tools are certainly useful, but they were developed over decades to meet the needs of traditional software engineering, and surveys have reported that they are inadequate for new needs, and little work has been done on how to ensure the quality of web applications.

The quality of Web applications is affected by several factors.

- Applications are built from numerous components from various sources - custom-built applications, off-the-shelf products, sometimes customized, and third-party products.
- The different software components have to be integrated usually without having the source code available, and they may even be hosted on remote servers owned by other organizations.
- The data flows required are complex and they must work on different types of software and hardware.
- The diversity of sources, environments and locations brings huge problems for quality assurance. To make decisions on the quality of all the diverse components, web developers need information on them - and that is often just not available.

Web quality factors and why they matter

Research suggests that the three most important criteria for web applications are:

- 1. Reliability
- 2. Usability
- 3. Security
- 4. Availability
- 5. Scalability
- 6. Maintainability
- 7. Time-to-market

These criteria are similar to those already considered earlier, but the priorities are different, due to the particular demands and challenges of web use.

1. *Reliability - whereas most traditional software does not need to be highly reliable, many* businesses' commercial success depends on web software. If it does not work reliably, they will lose customers and money.

2. *Usability - customers expect easy web transactions. Unusable sites will not be used.*
3. *Security - the consequences of insecure web sites were limited when they were just electronic* brochures, but nowadays much personal information is held by web applications and breaches of security can lead to losses in credibility and income, legal implications and large repair costs.

   Software security is a fast-growing research area.

4. Availability - customers will not wait to use a site that is down for maintenance, they will just go elsewhere. Global time differences mean that a site needs to be available at all times of day or night, all year round. Availability also includes accessibility and compatibility.

5.Scalability - the number of potential website customers that can visit a site is virtually unlimited. Applications must therefore be able to grow quickly, but robustly. Small weaknesses which were not a problem in a small application can lead to big problems as it grows.

6. Maintainability - there is a big difference from traditional software, where updates are collected over weeks and months, and released together. The update rate of websites is orders of magnitude faster than this. Updates can be implemented over days or even hours, and users get the updates immediately they are in place. Such frequent updates also have implications for compatibility - users may not update their software and hardware, so updated applications must work not just with many different types of system, but many different versions of them!

7. *Time-to-market - of course, this is still important for web systems, but for all the reasons given,* web development projects must allow time for systems to be designed and tested for security, usability and reliability if they are to get a return on their investment.

Conclusion

We are now starting to answer the question posed in first part of the of the course:

Is Internet/web development the same as, or different from, other types of computer systems development?

Certainly in the area of quality, we can see similarities and differences in the factors that contribute to quality, and in the importance that those factors hold.

The industry needs to continue developing tools and systems specifically to deal with web quality, taking

the best from traditional software development and adapting it to the new development environment.