


DESIGN QUALITY

Objectives

design support models


OBJECTIVES

- To understand the quality of software systems;
- To understand how design affects software quality;
- To understand the quality attributes of software design.

SOFTWARE QUALITY MODELS

- INTRODUCTION:
- Quality is one of the most elusive concepts that one may have.
- Different people may have different views on what is quality and how to measure the quality of a product or service.
- Even the same people may have different views on quality from time to time.
- According to the general theory of quality management, the complex and multifaceted concept can be described from five different views .

USER

- From a user's point of view, quality is 'fitness for purpose'. This view of quality evaluates the product or service according to whether it meets the user's needs.
- It, therefore, can be highly personal.
- The value-based view of quality is concerned with the ability to provide what the customer requires at a price that they can afford.
- Therefore, quality depends on the amount that a customer is willing to pay for it.

PRODUCTION

- From the manufacturing point of view, the quality of a product is the conformance to specification.
- It see quality as whether it is constructed 'right the first time', therefore, the costs associated with rework during development and after delivery can be avoided.
- It focuses on the development and construction process and leads to quality assessment that is virtually independent of the product itself.
- As software designers, we take the product view of quality to study what quality attributes the software should have.

Hierarchical models

- Quality attributes are often classified into a hierarchical structure to highlight the relationship between them.
- For example, McCall divided software quality attributes into 3 groups as shown in Figure 2.1.

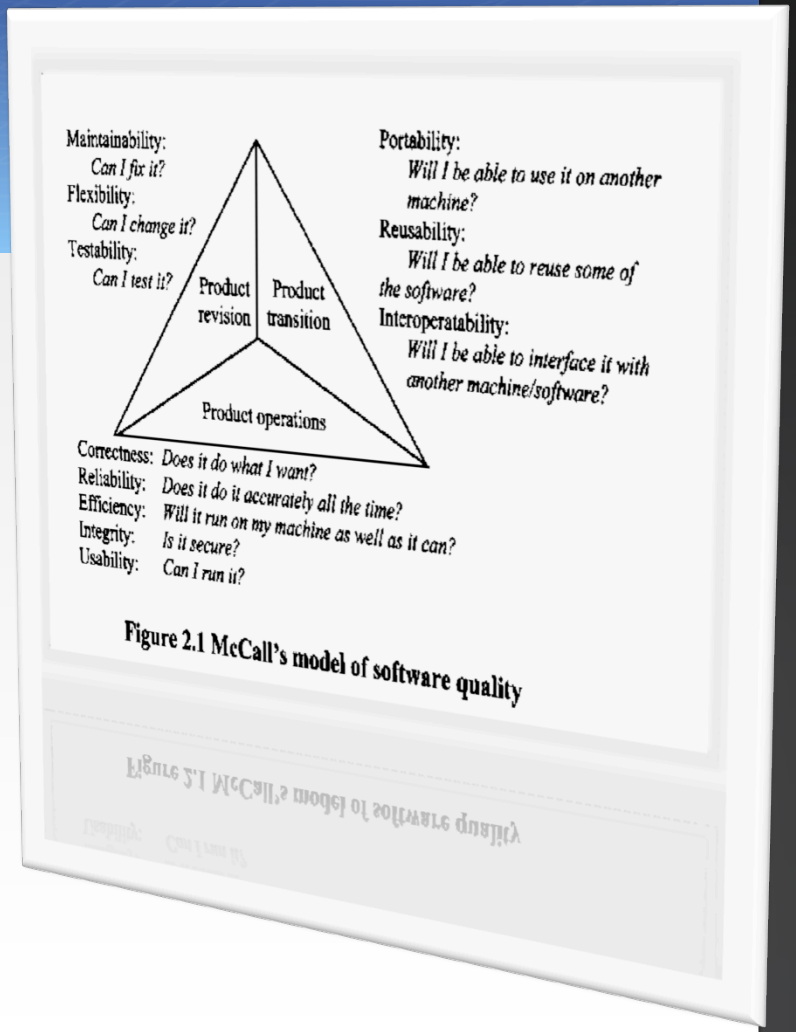
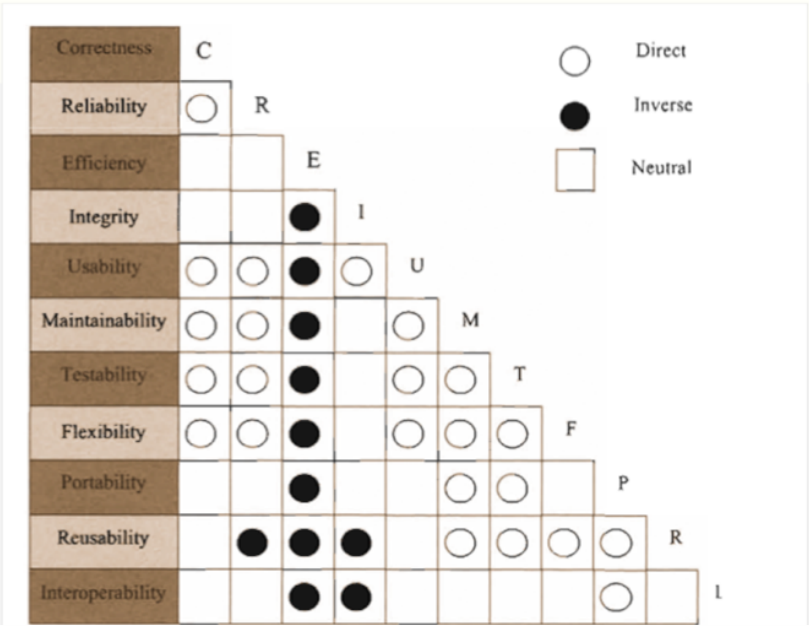


Figure 2.1 McCall's model of software quality

Relational models



Neutral,inverse,direct relation-ships between quality attributes.

Relationship model

- Integrity vs. efficiency (inverse): The control of data access will need additional code, leading to a longer runtime and more storage requirement.
- Usability vs. efficiency (inverse): Improvement of HCI will need more code and data, hence the system will be less efficient.30
- Maintainability and testability vs. efficiency (inverse): Compact and optimised code is not easy to maintain and test, and well-commented code is less efficient.
- Flexibility, reusability vs. integrity (inverse): Flexible data structures required for flexible and reusable software increase the data security problem.
- Flexibility and reusability vs. maintainability (direct): Maintainable code arises from the code that is well structured; meantime, well-structured maintainable code is easy to reuse in other programs.
- Portability vs. reusability (direct): Portable code is likely to be easily used in other environments. The code is likely well-structured and easier to be reused.
- Correctness vs. efficiency (neutral): The correcmess of code has no relation with its efficiency. Correct code may be efficient or inefficient in operation.

Gillies' relational model of software quality criteria

Criteria	R	E	I	S	U	F	EI	P	U	A	T	T	A	U	C
Reliability	O	+	+	O	O	O	O	O	O	O	O	O	O	O	O
Efficiency	O	–	–	–	–	–	–	–	O	–	+	–	–	O	–
Integrity	+	–	+	+	O	–	–	+	+	–	O	O	–	O	+
Security	+	–	+	+	O	–	–	+	+	–	O	O	–	O	–
Understandability	O	–	O	O	+	O	O	O	O	–	–	O	O	O	O
Flexibility	O	–	O	O	+	O	O	O	O	–	–	O	–	O	–
Ease of interfacing	O	O	+	+	+	+	+	+	–	–	O	O	O	O	O
Portability	O	O	+	+	+	+	+	O	O	–	O	O	O	O	–
User consultation	O	–	O	O	O	+	+	O	O	–	O	O	O	O	O
Accuracy	O	O	+	O	+	O	O	O	+	–	O	O	O	O	O
Timeliness	+	O	+	+	+	O	O	O	+	–	O	O	O	O	O
Time to use	–	O	–	–	O	–	–	–	–	–	–	–	–	O	–
Appeal	+	+	O	O	–	–	–	O	O	O	+	+	–	O	–
User flexibility	+	+	O	+	+	+	+	+	+	+	+	+	+	+	+
Cost/benefit	O	–	O	O	O	+	O	+	+	+	O	–	O	O	O
User friendliness	+	+	+	+	+	+	O	O	O	+	+	+	O	+	+
	+	O	O	O	O	O	O	O	O	+	O	–	O	+	O

O no relationship or that the relationship is heavily context-dependent.
\+ direct
– inverse

Gille's relationship

- In a study of software quality in six big organizations,
- Gillies developed six hierarchical quality models, in terms of the criteria used by both users and developers of software.
- Gillies also gave a relational model, illustrated in above diagram. As many as 16 pairs of quality attributes appeared in his model.
- His studies demonstrated that the relationships were often not commutative, which means although attribute A may reinforce attribute B, attribute B may not reinforce attribute A. i.e., a--->b but not b--->a
- Gillies claimed that his relational model was not project dependent.
- They are claimed to be applicable to all software systems. Therefore, they ignored the issues related to the specific features and application domains. For example, there is no weight associated to the quality attributes to express their relative importance.

- The value of a quality attribute for a given system can also change.
- Therefore, a good design made ten years ago can become less satisfactory now and might be regarded as out of date. Such phenomena are not unique for software designs. In fact, they are more general for all types of designs as stated in Mayall's axioms .
- This is equally true for software systems.
- For example, for the software that controls a nuclear power station, safety is perhaps the most important quality attribute, while, for a word processor, it is not an issue at all. Moreover, the importance of a quality attribute in assessing a particular software system may change as time passes.