Software: Its Nature and Qualities
(Goals of Software Engineering practice)

Dr. ZhiQuan (George) Zhou
Associate Professor

Discussion

- Interview questions:

1. How to design and test a vending machine?
2. What makes good quality software?

Discussion

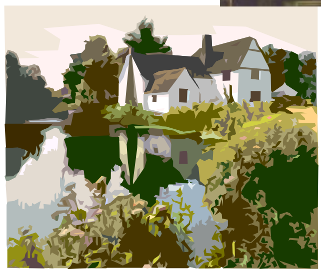
- What is software?
- Your experience with software?
  - Good?
  - Bad?
  - Software qualities?

Outline

- Software engineering (SE) is an intellectual activity and thus human-intensive
  - Requires engineering rather than manufacturing
- Software is built to meet a certain functional goal and satisfy certain qualities
- Software processes also must meet certain qualities
- Software qualities are sometimes referred to as "ilities"

Software product

- Discussion:
  - How are sw products different from traditional types of products?



Software product

- Different from traditional types of products
  - Intangible
    - difficult to describe and evaluate
  - Malleable
    - We can modify the product (as opposed to its design) "easily"
    - Software's malleability sometimes leads people to think that it can be changed easily. In practice, it cannot

Software product

- Different from traditional types of products (cont.)
  - human intensive
    - involves only trivial "manufacturing" process ▪

Software Qualities
Part I

sw qualities "ilities"

- These qualities will become our goals in the practice of SE
- To achieve those goals, we need to apply SE principles (to be discussed later)
- The presence of any quality needs to be verified and measured

Classification of sw qualities "ilities"

- Internal vs. external

Discussion: Do users and developers have the same view?

Classification of sw qualities "ilities"

- Internal vs. external
  - External→ visible to users
    - E.g. reliability
  - Internal→ concern developers
    - E.g. verifiability

Relationship between the two?

e.g. the vending machine?

Classification of sw qualities "ilities"

- Internal vs. external
  - External→ visible to users
    - E.g. reliability
  - Internal→ concern developers
    - E.g. verifiability
  - Internal qualities, which deal largely with the **structure** of the software, help developers achieve the external qualities
    - E.g. verifiability --> reliability

Classification of sw qualities "ilities"

- Product vs. process
  - Our goal is to develop software products
    - Deliverables: to be handed over to the client at the end
    - Intermediate products: used in the process of creating the deliverables
  - The process is how we do it
  - Their **relationship?**

Classification of sw qualities "ilities"

- Product vs. process

  - **Process** qualities are closely related to **product** qualities.
    - How?

Classification of sw qualities "ilities"

Key points:

- Internal qualities affect external qualities
- Process qualities affect product qualities .

Representative Qualities

- Important qualities of software products and processes

group discussion

Representative Qualities

- Correctness, reliability, and robustness
- Performance
- Usability
- Verifiability
- Maintainability
- Reparability
- Evolvability
- Reusability
- Portability
- Understandability
- Interoperability
- Productivity
- Timeliness
- Visibility

$\Bigg\}$ Important qualities

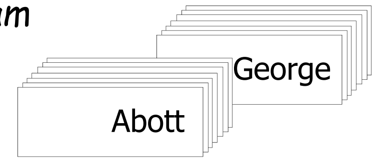
Correctness

- Software is correct if it satisfies the functional requirements specifications
  - assuming that specification exists!
- If specifications are formal, since programs are formal objects, correctness can be defined formally
  - It can be proven as a theorem or disproved by counterexamples (e.g. testing)

e.g. consider a calculator, OS, Google

Constructing a "correct" program

Example

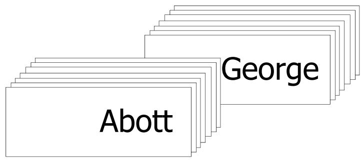


- A searching problem:
  - A sorted deck of cards
  - Suppose you are also presented with one additional card on which is printed the name of a student X
  - Task: write a program to split the deck of cards into two parts in such a way that (a) all of the cards in the first part precede X in alphabetical order and (b) none of the cards in the second part precedes X in alphabetical order.
  - We are effectively looking for the position to insert the new card.

Your solution?

Cited from: Roland Backhouse, *Program Construction: Calculating Implementations from Specifications,* John Wiley & Sons, 2003

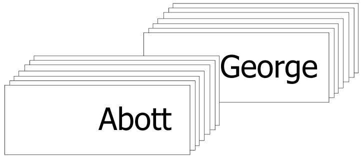
A searching problem (cont.)



- First step: to ensure that you have a clear understanding of the problem
  - For programming purposes, the demands on clarity and un-ambiguity of the problem specification are much **higher** than if the task is to be performed manually by a person, when one can rely on common sense and intelligence.
  - Any point needing clarification?

Roland Backhouse, *Program Construction: Calculating Implementations from Specifications,* John Wiley & Sons, 2003

A searching problem (cont.)

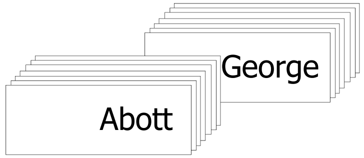


- Any point needing clarification?
  - One of the two parts may become empty
    - "no cards" is still a deck (that is, an empty deck)
    - Practical effect: reducing the number of cases: 3 -> 1
  - How about an empty deck of cards?
    - Will result in 2 empty decks

How do you approach the problem?

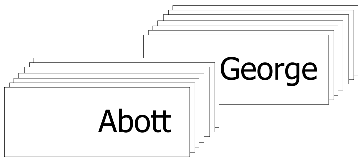
Roland Backhouse, *Program Construction: Calculating Implementations from Specifications,* John Wiley & Sons, 2003

A searching problem (cont.)



- Problem solution
  - At all times, maintain 3 decks of cards:
    - the *left deck*: contains cards that are all known to precede X;
    - the *right deck*: contains cards that are all known to not precede X;
    - the *middle deck*: contains cards that may or may not precede X.
  - All 3 decks are ordered and are such that left + middle + right decks = original deck

A searching problem (cont.)

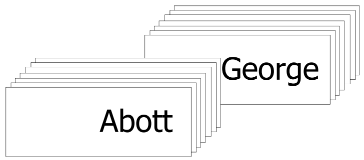


- Problem solution
  - At all times, maintain 3 decks of cards:
    - the *left deck*: contains cards that are all known to precede X;

Become loop invariants

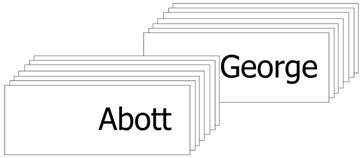
    - the *middle deck*: contains cards that may or may not precede X.
  - All 3 decks are ordered and are such that left + middle + right decks = original deck

A searching problem (cont.)



- Problem solution (cont.)
  - Initially, the left and right decks are __________

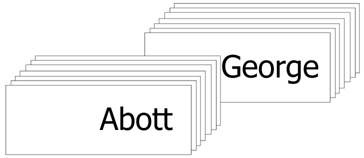
A searching problem (cont.)



- Problem solution (cont.)
  - Initially, the left and right decks are __both empty__

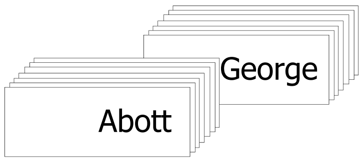
Become Initialization of variables

A searching problem (cont.)



- Problem solution (cont.)
  - The task is complete when____________________ ___

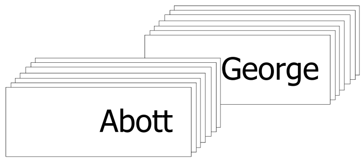
A searching problem (cont.)



- Problem solution (cont.)
  - The task is complete when __the middle deck is empty__

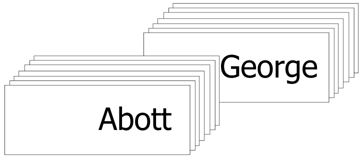
Become termination condition

A searching problem (cont.)



- Problem solution (cont.)
  - We make progress to this state by repeatedly ____ ________

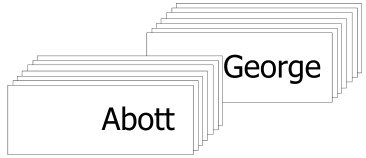
A searching problem (cont.)



- Problem solution (cont.)
  - We make progress to this state by repeatedly moving cards from the middle deck to the left or right deck.

Become loop body

A searching problem (cont.)



- Problem solution (cont.)
  - Can you now write the program?

Towards correctness

- Use proven methodologies and processes
- Use standard proven algorithms or libraries (rather than writing new ones)
- Rigorous functional specification
- Scientific construction of programs
- Verification
  - formal methods, inspection, testing
- Static and dynamic analysis tools (eg JPF)

Towards correctness

•Structured programming
* Total correctness and partial correctness
* Fight against software faults:
    Fault prevention
    Fault detection
    Fault tolerance
    Fault correction (run-time correction)

E.g. N-version programming
E.g. Data diversity
v.s. Metamorphic testing

Example

- Data diversity
  - Think beyond integers?

Example

- Data diversity
  - Sin(a) ?

Example

- Data diversity (cont.)

sin (a + b) = sin (a) cos (b) + cos (a) sin (b)

Example

- Data diversity (cont.)

sin (a + b) = sin (a) cos (b) + cos (a) sin (b)

cos (a) = sin (pi/2 - a)

Example

- Data diversity (cont.)

sin (a + b) = sin (a) cos (b) + cos (a) sin (b)

cos (a) = sin (pi/2 - a)

sin (x) = sin (a) sin (pi/2 - b) + sin (pi/2 - a) sin (b)
where x=a+b

More about correctness

- It is an absolute (yes/no) quality
- What if the specification is wrong?
  - (e.g., it derives from incorrect requirements or errors in domain knowledge) .

Software qualities
Part II

Discussion

- Is "correct software" all what we want?

Discussion

- Not all incorrect behaviors signify equally serious problems, that is, some incorrect behaviors may be tolerable
  - Examples?

Discussion

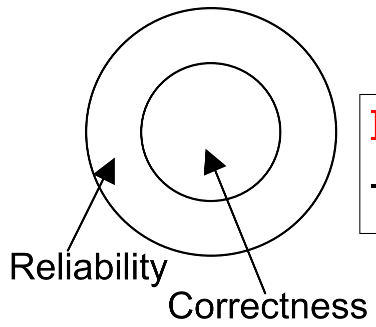
- Not all incorrect behaviors signify equally serious problems, that is, some incorrect behaviors may be tolerable
  - This is about "reliability".

Reliability

- informally, user can rely on it
  - "dependable" is often used as a synonym for "reliable"
- "correctness" is an absolute quality; "reliability" is relative.
  - E.g. "probability of absence of failures for a certain time period"
  - If the consequence of an error is not serious, incorrect software may still be reliable.

Reliability

- if specs are correct, all correct software is reliable, but not vice-versa
  - **Why**?

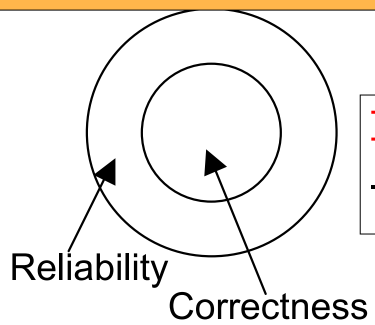


Idealized situation

--- If specification is correct

Reliability

We sometimes have correct applications designed for **incorrect specifications**, so that correctness of the software may not be sufficient to guarantee the user that the software behaves "as expected".



Idealized situation

--- If specification is correct

Reliability

- Software aging
  - progressive performance degradation or sudden crash (after longtime execution).
  - cause: resources?
  - software rejuvenation (a countermeasure)

Reliability

- Acceptance testing
- Alpha testing, beta testing

(vs. System testing, integration testing)

Question

- How about unspecified situations?

Robustness

- software behaves "reasonably" even in unforeseen circumstances
  - **Examples**?

Robustness

- software behaves "reasonably" even in unforeseen circumstances
  - e.g., incorrect input, hardware failure
  - Even in circumstances not anticipated in the requirements specificaiton
  - E.g. Database management systems
    ATM machine
    Public telephone
    Hackers using unexpected http requests

Robustness

- Discussion: could a correct program be not robust ?
  - When should you answer yes?
  - When should you answer no?

Robustness

- Question: could a correct program be not robust ?
  - Yes if the **specification does not state** what the program should do in the face of special situations (e.g. illegal inputs)
  - No if we could **specify** precisely what the application should do to make it robust
  - **Examples**?

Robustness

- The amount of code devoted to robustness depends on the application area
  - Compare:
    - An application written for novice computer users
    - An embedded system
      - Question: Need more or less code on robustness?

Robustness

- The amount of code devoted to robustness depends on the application area
  - Compare:
    - An application written for novice computer users
    - An embedded system

      - **But it often controls critical devices**

Correctness, robustness, and reliability for software production process

- In many engineering disciplines, much research is done to discover reliable processes
  - A process is reliable if it consistently leads to the production of high quality products (**different from "productivity".**)

Correctness, robustness, and reliability for software production process

- A process is robust, for example, if ______ ____________________________

Correctness, robustness, and reliability for software production process

- A process is robust, for example, if it can accommodate unanticipated changes in the enironment
  - Example?

Correctness, robustness, and reliability for software production process

- A process is robust, for example, if it can accommodate unanticipated changes in the enironment
  - e.g. sudden transfer of half the empoloyees .

Performance

- Efficient use of resources

What are they?

Performance

- Efficient use of resources
  - memory, processing time
  - Less traditionally: message exchanges in the case of distributed systems

How to evaluate performance?

Performance

- Evaluate/verify the performance
  - **Analyzing** complexity of algorithms

  Advantages and disadvantages?

Performance

- Evaluate/verify the performance
  - **Analyzing** complexity of algorithms
    - Only provide average or worst case info.
    - Rather than specific info about a particular implementation
    - May be **difficult** to apply in some situations

Performance

- Evaluate/verify the performance (cont.)
  - **Measurement**
    - Measure the actual performance
    - By means of run-time monitoring

    e.g. http://www.spec.org
    e.g. "debug version" vs. "release version"

    Advantages and disadvantages?

Performance

- Evaluate/verify the performance (cont.)
  - **Measurement**
    - It is crucial to select representative input data

Performance

- Evaluate/verify the performance (cont.)
  - Simulation
    - E.g. ART vs. RT

Performance

- Performance can affect scalability
  - a solution that works on a **small** local network may not work on a **large** intranet
    - Both **product** and **process**
    - **E.g. performance testing**

Performance

- Performance also applies to a development process: we call it productivity**.**

Usability

- Usable, or user friendly if
  - expected users find the system easy to use
- Easy to evaluate??

Usability

- Usable, or user friendly if
  - expected users find the system easy to use
- Rather subjective, difficult to evaluate
- E.g. usability testing

Usability

- Affected by ________________

Usability

- Affected by *user interface*
  - e.g., visual vs. textual
    - e.g., McAfee "failed to update"
- More than user interface
  - E.g. embedded system
    - Can the system be configured and adapted to the hardware environment easily?

Usability

- In many engineering disciplines
  - human factors, usability engineering
    - Extensive study of user needs and attitudes by specialists in fields such as industrial design or psychology

How to improve usability?
Can you drive another car? Why?

Usability

- In many engineering disciplines
  - human factors, usability engineering
    - E.g. Car manufacturers: position of various control knobs on the dashboard
    - E.g. TV manufacturers, microwave oven makers: try to make their products easy to use

Usability

- In many engineering disciplines (cont.)
  - Achieved through standardization of human interface
    - Once you know how to use one TV set, you can operate almost any other TV set
    - How about driving cars?
  - A clear trend in software applications
    - to more uniform and standard user interfaces, e.g. in Web browsers, GUI .

Verifiability

- How easy it is to verify the software
  - E.g. can every branch/statement be tested?
  - E.g. Path conditions?
  - E.g. int a[5]; a[7]=0;
- Use disciplined coding practices .

Maintainability

- Maintainability: ease of maintenance
- What is Maintenance in other engineering products?
  - upkeep of the product in response to the gradual deterioration of parts
    - E.g. transmissions are oiled
    - E.g. air filters are dusted and periodically changed

Maintainability

- Software: does not wear out
  - Unfortunately, the term is used so widely that we are practically obliged to continue using it.
- Software maintenance?

changes after release

How many kinds of changes?

Maintainability

- Maintenance costs exceed 60% of total cost of software
- Three main categories of maintenance
  - ***corrective***: removing residual errors (20%)
  - ***adaptive***: adjusting to environment changes (20%)
  - ***perfective***: quality improvements (>50%)

Maintainability

- Can be decomposed as
  - Repairability
    - ability to **correct** defects in reasonable time
  - Evolvability
    - ability to **adapt** sw to environment changes and to **improve** it in reasonable time

Question 1: Why changes are unavoidable?
Question 2: How would you add a second story of a house?

Maintainability

- Evolvability (cont.)
  - Software is **malleable**
    - Hence modifications are extremely "easy" to an implementation
  - Modification of **other** engineering products: starts at the **design** level
    - e.g. to add a second story of a house – first study "can the addition be done safely?"; then design; then the design must be approved; then construction.

Maintainability

- Evolvability (cont.)
  - Poor modification of software
    - People often ____ the feasibility and design analysis phases and proceed immediately to ______

Maintainability

- Evolvability (cont.)
  - Poor modification of software
    - People often skip the feasibility and design analysis phases and proceed immediately to modify the implementation

Maintainability

- Evolvability (cont.)
  - Poor modification of software
    - What is worse: after the change is accomplished, the modification is not even documented -- the specifications are not updated to reflect the change
    - This makes ______________________________ difficult

Maintainability

- Evolvability (cont.)
  - Poor modification of software
    - What is worse: after the change is accomplished, the modification is not even documented -- the specifications are not updated to reflect the change
    - This makes future changes more and more difficult

e.g. change impact analysis, dynamic analysis for program understanding.

Maintainability

- Evolvability (cont.)
  - Successful modification of software
    - If the software is
      - **designed** with evolution in mind, and
      - each **modification** is designed & applied carefully

Maintainability

- Evolvability (cont.)
  - Successful modification of software (cont.)
    - Successful software products are quite long lived
      - Their first release is the first of many releases
      - Each successive release being a step in the evolution of the system

Maintainability

- Evolvability (cont.)
  - Evolvability of sw is more and more important
    - **Why?**

Maintainability

- Evolvability (cont.)
  - Evolvability of sw is more and more important
    - **Economic** impact: sw cost and complexity
    - Leverage investment in sw as hardware advances
    - E.g. the American Airlines SABRE reservation system
      - Initially developed in the 1960s
      - Evolving for decades

Maintainability

- Evolvability (cont.)
  - Studies of large sw systems show
    - Evolvability <u>decreases/increases?</u> with each release
      - Why?

Maintainability

- Evolvability (cont.)
  - Studies of large sw systems show
    - Evolvability decreases with each release
      - Each release complicates sw structure
      - Hence future modification is more difficult to apply
  - Several SE principles help achieve evolvability .

Reusability

- Existing product (or components) used (with minor modifications) to build another product
  - Compare with evolvability
- Examples
  - ?

Reusability

- Examples
  - Numeric libraries
  - Unix shell
    - designed to be used both interactively and in "batch"
  - One of the goals of **OO**: achieve both reusability and evolvability

Reusability (cont.)

- Reusability of standard parts shows maturity of the field
  - Examples?

Reusability (cont.)

- Reusability of standard parts shows maturity of the field (cont.)
  - E.g. Cars: Standardized components used across many models

Reusability (cont.)

- lib, dll, file.a, file.o, …?

Portability

- Software is "portable" if it can run in different environments
  - Different hw or sw environment
  - e.g. a Web browser on Unix workstation, MS Windows PC, palmtop, mobile phone
- Economically important
- E.g. Unix and Linux have been ported to many different hw systems.

Portability

- Java?

Understandability

- Ease of understanding software
- Program modification requires program understanding
  - Software maintenance is dominated by the sub-activity of program understanding
  - Maintenance engineers spend most of their time trying to uncover the logic of the application and a smaller portion of their time applying changes to the application.

Interoperability

- Ability of a system to coexist and cooperate with other systems
  - Examples?

Interoperability

- Ability of a system to coexist and cooperate with other systems (cont.)
  - e.g., word processor and spreadsheet
  - e.g., software processing images from a scanner

Interoperability

- In **other engineering** products
  - E.g. Stereo systems, TV sets, video recorders from different manufacturers work together

How to achieve interoperability?

Interoperability

- Can be achieved through ________

Interoperability

- Can be achieved through standardization of interfaces
  - Examples?

Interoperability

- Can be achieved through standardization of interfaces (cont.)

Question:

Replacing a broken part of a gun with a similar part from another gun, in a battle field (relationship with **reusability**?)

Interoperability

- Open systems allow different apps to interoperate.
  - An extensible collection of independently written applications that function as an integrated system
  - Open interfaces
    - E.g. The Internet, TCP/IP
    - E.g. Web Services, XML .

Typical process qualities

- **Productivity** Number of lines of code produced??
  - denotes its efficiency and performance
- **Timeliness**
  - ability to deliver a product on time
- **Visibility**
  - all of its steps and current status are documented clearly

Productivity

- An *efficient* process results in faster delivery of the product
- Difficult to measure
  - Number of lines of code produced?
    - May discourage re-use
- Affected strongly by automation.

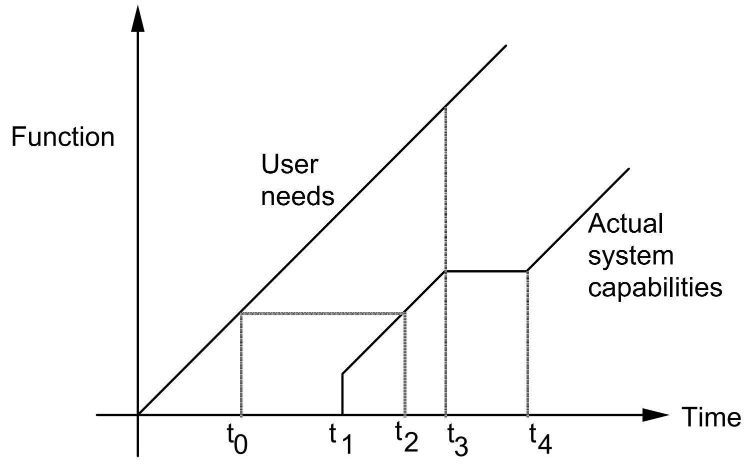
Timeliness: Problems

- **Timeliness** by itself is not a useful quality
  - Delivering on time a product that is defective is pointless
  - But early delivery of a preliminary version may favor the later acceptance of the final product

Timeliness: Problems

- How to measure the amount of work required for producing a piece of software?
- How to define precise and verifiable milestones?
- A mismatch occurs between user requirements and status of the product
  - Because of continuously changing user requirements

Timeliness shortfall



Case study (company A)

- In the 1980's, company A had promised the first release of its Ada compiler for a certain date. When the date arrived, the customers who had ordered the product received a letter stating that, since the product still contained many defects, the manufacturer had decided that it would be better to delay delivery rather than deliver a product containing defects. The product was promised for 3 months later.
- After 3 months, the product arrived, along with a letter stating that many, but not all, of the defects had been corrected. But this time, the manufacturer had decided that it was better to let customers receive the Ada compiler, even though it contained several serious defects, so that the customers could start their own product development using Ada. The value of early delivery at this new time outweighed the risk of delivering a defective product, in the opinion of the manufacturer. So, in the end, what was delivered was **late and defective**.

Case study (company B)

- Company B delivered, very early on, a compiler that supported a very small subset of the Ada language. Novel features of Ada, such as tasking and exception handling, were not supported. The result was the early delivery of a reliable product. As a consequence, the users started experimenting with the new language, and the company took more time to understand the subtleties of the new features of Ada. Over several releases, which took a period of two years, a full Ada compiler was delivered. Incremental delivery allows the product to become available earlier, and the use of the product helps in refining the requirements.

Timeliness: a Treatment

- Incremental delivery of the product

Visibility

- All of its **steps** and current status (of the intermediate **products**) are documented clearly

Case study

In many projects, most engineers and even managers are unaware of the exact status of the project: some may be **designing**, others **coding**, and still others **testing**, all at **the same time**. This, by itself, is not bad. However,…

tension arises: if the integration group has been testing a version assuming that the next version will involve fixing defects, while the engineering group starts to redesign a major part of the code to add functionality--This tension between one group trying to stabilize the software while another group is destabilizing it is common.

Visibility allows engineers to

____

____

Case study

In many projects, most engineers and even managers are unaware of the exact status of the project: some may be **designing**, others **coding**, and still others **testing**, all at **the same time**. This, by itself, is not bad. However,…

tension arises: if the integration group has been testing a version assuming that the next version will involve fixing defects, while the engineering group starts to redesign a major part of the code to add functionality--This tension between one group trying to stabilize the software while another group is destabilizing it is common.

Visibility allows engineers to <u>weigh the impact</u> of their actions and thus guides them in <u>making decisions</u>. It allows the members of the team to work in the **same direction**, rather than, as is often the case currently, in opposing directions ▪

Application-specific qualities

- Information systems (data oriented)
  - Data integrity
    - Under what circumstances will the data be corrupted when the system malfunctions?
  - Security
    - To what extent does the system protect the data?
  - Data availability
    - Under what conditions will the data become unavailable and for how long?
  - Transaction performance
    - Number of transactions carried out per unit time ▪

Application-specific qualities

- Real-time systems
  - Must respond to events within a predefined and strict period of time
    - E.g. Factory-monitoring: sudden increase in temperature
    - E.g. Controlling flight path
  - Different from "fast response times"
    - E.g. Mouse click: single/double click ▪

Application-specific qualities

- Distributed systems
  - The amount of distribution supported, e.g., are the **data** distributed, or is the **processing**, or both?
  - Whether the system can **tolerate** the partitioning of the network, e.g., when the network link makes it impossible for two subsets of computers to communicate
  - Whether system **tolerates** the failure of individual computers ▪

Application-specific qualities

- Many systems exhibit characteristics that are common to several areas
  - E.g. an **information** system that may also have some **real-time** requirements, and this system may also be **distributed**.

Quality measurement

- Qualities: goals in practice
  - These are what we want!
- Then we need principles and techniques
  - to help achieve the goals
- We also need to be able to *measure* a given quality
  - Many qualities are subjective
  - No standard metrics defined for most qualities
  - Much research is under way into objective metrics