Software Quality Measurement

Dr. Mark C. Paulk
Mark.Paulk@utdallas.edu, Mark.Paulk@ieee.org

Jonsson School of Engineering and Computer Science
What Is Quality?
Some Traditional Definitions

Quality is conformance to requirements
  - Phillip Crosby

Quality is defined by the customer
  - W. Edwards Deming

Quality is fitness for use.
  - Joseph Juran

Nine dimensions of quality: performance, features, functionality, safety, conformance, reliability, durability, service, and aesthetics
  - David Garvin (1987)
Is Quality “Failures” of the Software?
IEEE 1633 Failure Severity

A rating system for the impact of every recognized credible software failure mode.

NOTE—The following is an example of a rating system:
  • Severity #1: Loss of life or system
  • Severity #2: Affects ability to complete mission objectives
  • Severity #3: Workaround available, therefore minimal effects on procedures (mission objectives met)
  • Severity #4: Insignificant violation of requirements or recommended practices, not visible to user in operational use
  • Severity #5: Cosmetic issue that should be addressed or tracked for future action, but not necessarily a present problem.
Is Quality “Defects” in the Software?
IEEE 1044 Defect Attributes

Defect ID, Description, Status, Asset, Artifact, Version detected, Version corrected, Probability, Effect, Type, Mode, Insertion activity, Detection activity, Failure reference, Change reference, ...

**Priority** Ranking for processing assigned by the organization responsible for the evaluation, resolution, and closure of the defect relative to other reported defects.

**Severity** The highest failure impact that the defect could (or did) cause, as determined by (from the perspective of) the organization responsible for software engineering.
## Severity vs Priority

<table>
<thead>
<tr>
<th>Severity</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical</td>
<td>Resolve Immediately</td>
</tr>
<tr>
<td></td>
<td>- show stopper</td>
</tr>
<tr>
<td>Major</td>
<td>Give High Attention</td>
</tr>
<tr>
<td></td>
<td>- urgent</td>
</tr>
<tr>
<td>Average</td>
<td>Normal Queue</td>
</tr>
<tr>
<td></td>
<td>- high</td>
</tr>
<tr>
<td>Minor</td>
<td>Low Priority</td>
</tr>
<tr>
<td></td>
<td>- medium</td>
</tr>
<tr>
<td>Exception</td>
<td>Defer</td>
</tr>
<tr>
<td></td>
<td>- low, cosmetic</td>
</tr>
</tbody>
</table>
Traditional Definitions of Quality
(Glass 1998)

The traditional definitions are wrong.
• quality is not about user satisfaction, product conformance, or costs and schedules
• nor is it solely about defects

Instead, there is a well-defined, intuitive relationship between quality and those other product traits, one that clearly distinguishes among them.
Quality Is A Collection (Glass, 2004)

Quality is a collection of attributes.
- Reliability is about a software product that does what it’s supposed to do, dependably.
- Human engineering (also known as usability) is about a software product that is easy and comfortable to use.
- Understandability is about a software product that is easy for maintainers to comprehend.
- Modifiability is about a software product that is easy for a maintainer to change.
- Efficiency is about a software product that economizes on both running time and space consumption.
- Testability is about a software product that is easy to test.
- Portability is about creating a software product that is easily moved to another platform.
Quality equals the totality of features and characteristics of a product or service that bear on its ability to satisfy stated and implied need.
- ISO 8402: 1986

Quality = portability + reliability + efficiency + human engineering + understandability + modifiability + testability (Boehm 1975)
- different software products call for different “ilities” mixes
- no universal definition of quality at the detail level, that is independent of the product in question, exists
ISO/IEC 25010
Software Quality Requirement and Evaluation (SQuaRE)

Functional
suitability
- functional
completeness
- functional
correctness
- functional
appropriateness

Performance
efficiency
- time behavior
- resource utilization
- capacity

Compatibility
- coexistence
- interoperability

Usability
- appropriateness
- recognizability
- learnability
- operability
- user error prediction
- user interface
- aesthetics
- accessibility
Reliability
- maturity
- availability
- fault tolerance
- recoverability

Security
- confidentiality
- integrity
- nonrepudiation
- accountability
- authenticity

Maintainability
- modularity
- reusability
- analyzability
- modifiability
- testability

Portability
- adaptability
- installability
- replaceability
Seven Quality Attributes in Software Architecture

Availability
Interoperability
Modifiability
Performance
Security
Testability
Usability

Availability

A property of software that it is there and ready to carry out its task when you need it to be.

Builds upon the concept of reliability by adding the notion of recovery

“Availability refers to the ability of a system to mask or repair faults such that the cumulative service outage period does not exceed a required value over a specified time interval.”

Availability is about minimizing service outage time by mitigating faults.
Availability Tactics

- Detect Faults
  - Ping / Echo Monitor
  - Heartbeat
  - Timestamp
  - Sanity Checking
  - Condition Monitoring
  - Voting
  - Exception Detection
  - Self-Test

- Recover from Faults
  - Preparation and Repair
    - Active Redundancy
    - Passive Redundancy
    - Spare
    - Exception Handling
    - Rollback
    - Software Upgrade
    - Retry
    - Ignore Faulty Behavior
    - Degradation
    - Reconfiguration
  - Reintroduction
    - Shadow State Resynchronization
    - Escalating Restart
    - Non-Stop Forwarding

- Prevent Faults
  - Removal from Service
  - Transactions
    - Predictive Model
  - Exception Prevention
  - Increase Competence Set

Fault → Fault Masked or Repair Made
Interoperability

The degree to which two or more systems can usefully exchange meaningful information via interfaces in a particular context.

Syntactic interoperability – the ability to exchange data.

Semantic interoperability – the ability to correctly interpret the data being exchanged.
Interoperability Tactics

- Locate
  - Discover Service
- Manage Interfaces
  - Orchestrate
  - Tailor Interface

Information Exchange Request

Request Correctly Handled
Modifiability

Modifiability is about change, and our interest in it centers on the cost and risk of making changes.

What can change?

What is the likelihood of the change?

When is the change made and who makes it?

What is the cost of the change?
Performance

It’s about time and the software system’s ability to meet timing requirements.

When events occur, the system, or some element of the system, must respond to them in time.
• real-time
• hard real-time

Characterizing the events that can occur (and when they can occur) and the system or element’s time-based response to those events is the essence is discussing performance.
Performance Tactics

Control Resource Demand
- Manage Sampling Rate
- Limit Event Response
- Prioritize Events
- Reduce Overhead
- Bound Execution Times
- Increase Resource Efficiency

Manage Resources
- Increase Resources
- Introduce Concurrency
- Maintain Multiple Copies of Computations
- Maintain Multiple Copies of Data
- Bound Queue Sizes
- Schedule Resources

Event Arrives
Response
Generated within Time Constraints
Security

A measure of the system’s ability to protect data and information from unauthorized access while still providing access to people and systems that are authorized.

An action taken against a computer system with the intention of doing harm is called an attack.

- an unauthorized attempt to access data or services
- an unauthorized attempt to modify data
- intended to deny services to legitimate users
Security Tactics

Detect Attacks
- Detect Intrusion
- Detect Service Denial
- Verify Message Integrity
- Detect Message Delay

Resist Attacks
- Identify Actors
- Authenticate Actors
- Authorize Actors
- Limit Access
- Limit Exposure
- Encrypt Data
- Separate Entities
- Change Default Settings

React to Attacks
- Revoke Access
- Lock Computer
- Inform Actors

Recover from Attacks
- Maintain Audit Trail
- Restore
- See Availability

System Detects, Resists, Reacts, or Recovers
Testability

Refers to the ease with which software can be made to demonstrate its faults through (typically execution-based) testing.

Refers to the probability, assuming that the software has at least one fault, that it will fail on its next test execution.

Industry estimates indicate that between 30 and 50 percent (or in some cases, even more) of the cost of developing well-engineered systems is taken up by testing.
Testability Tactics

- Control and Observe System State
  - Specialized Interfaces
    - Record/Playback
    - Localize State Storage
    - Abstract Data Sources
    - Sandbox
    - Executable Assertions

- Limit Complexity
  - Limit Structural Complexity
    - Limit Nondeterminism

Tests Executed → Faults Detected
Usability

Concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides.

Usability comprises:
- Learning system features.
- Using a system efficiently.
- Minimizing the impact of errors.
- Adapting the system to user needs.
- Increasing confidence and satisfaction.
Usability Tactics

Support User Initiative
- Cancel
- Undo
- Pause/Resume
- Aggregate

Support System Initiative
- Maintain Task Model
- Maintain User Model
- Maintain System Model

User Given Appropriate Feedback and Assistance
Cost and Schedule

Should cost and schedule be considered quality attributes?

Are they included in customer expectations?
  • for a custom software development project?
  • for a commercial software product?
    - look among competing products “off the shelf” for features, cost, and availability
Key Measurement Questions

Are we measuring the right thing?
- Goal / Question / Metric (GQM)
- business objectives ⇔ data
  - cost (dollars, effort)
  - schedule (duration, effort)
  - functionality (size)
  - quality (defects)

Are we measuring it right?
- operational definitions

Popular measures focus on project management concerns… with a focus on “quality” as defects in meeting requirements.
Goal-Driven Measurement

Goal / Question / Metric (GQM) paradigm


SEI variant: goal-driven measurement


ISO 15939 and PSM variant: measurement information model

**Goal-Driven Measurement**

**Goals**

Business => Sub-goals => Measurement

**Questions**

- How large is our backlog of customer change requests?
- Is the response time for fixing bugs compatible with customer constraints?

**Indicators**

- SLOC
- Staff-Hours
- Trouble Reports
- Milestone dates

**Definition Checklist**

- 4
- 4
- 4
- 4

**Indicator Template**

- Objective
- Question

**Infrastructure Assessment**

**Analysis & Diagnosis**

**Action Plans**
Operational Definitions

The rules and procedures used to capture and record data

What the reported values include and exclude

Operational definitions should meet two criteria

- Communication – will others know what has been measured and what has been included and excluded?
- Repeatability – would others be able to repeat the measurements and get the same results?
SEI Core Measures

Dovetails with SEI’s adaptation of goal-driven software measurement

Checklist-based approach with strong emphasis on operational definitions

Measurement areas where checklists have already been developed include:
  • effort
  • size
  • schedule
  • quality

See http://www.sei.cmu.edu/measurement/index.cfm
SLOC Definition Considerations

Whether to **include** or **exclude**
- executable and/or non-executable code statements
- code produced by programming, copying without change, automatic generation, and/or translation
- newly developed code and/or previously existing code
- product-only statements or also include support code
- counts of delivered and/or non-delivered code
- counts of operative code or include dead code
- replicated code

When the code gets counted
- at estimation, at design, at coding, at unit testing, at integration, at test readiness review, at system test complete
Response Measures for Availability

Time or time interval when the system must be available

Availability percentage

Time to detect the fault

Time to repair the fault

Time or time interval in which system can be in degraded mode

Proportion or rate of a certain class of faults that the system prevents, or handles without failing
Response Measures for Interoperability

Percentage of information exchanges correctly processed

Percentage of information exchanges correctly rejected
Response Measures for Modifiability

Cost in terms of …

Number, size, complexity of affected artifacts

Effort

Calendar time

Money (direct outlay or opportunity cost)

Extend to which this modification affects other functions or quality attributes

New defects introduced
Response Measures for Performance

The time it takes to process the arriving events (latency or deadline)

The variation in this time (jitter)

The number of events that can be processed within a particular time interval (throughput)

A characterization of the events that cannot be processed (miss rate)
Performance Measures for Security

How much of a system is compromised when a particular component or data value is compromised

How much time passed before an attack was detected

How many attacks were resisted

How long it took to recover from a successful attack

How much data was vulnerable to a particular attack
Performance Measures for Testability

- Effort to find a fault or class of faults
- Effort to achieve a given percentage of state space coverage
- Probability of fault being revealed by the next test
- Time to perform tests
- Effort to detect faults
- Length of longest dependency chain in test
- Length of time to prepare test environment
- Reduction in risk exposure
  - size of loss * probability of loss
Performance Measures for Usability

Task time

Number of errors

Number of tasks accomplished

User satisfaction

Gain of user knowledge

Ratio of successful operations to total operations

Amount of time or data lost when an error occurs
Dysfunctional Behavior

Austin’s *Measuring and Managing Performance in Organizations*

- motivational versus information measurement

Deming strongly opposed performance measurement, merit ratings, management by objectives, etc.

Dysfunctional behavior resulting from organizational measurement is inevitable unless
- measures are made “perfect”
- motivational use impossible
Software Complexity

Complexity is everywhere in the software life cycle... usually an undesired property... makes software harder to read and understand... harder to change


Dependencies between seemingly unrelated parts of a system... (unplanned) couplings between otherwise independent system components

Complexity Is Complex


- nonlinear dynamics
- open and closed systems
- feedback loops
- fractal structures
- co-evolution
- natural elements of human group behavior
  - exchange energy (competition to collaboration)
  - share information (limited to open and fully)
  - align choices for interaction (shallow to deep)
  - co-evolve (from on-the-fly to with-coordination)
Types of Complexity

Nonlinear dynamics $\rightarrow$ small differences at the start may lead to vastly different results
  - the butterfly effect

Open systems $\rightarrow$ the boundaries permit interaction with the environment

Feedback loops $\rightarrow$ a series of actions, each of which builds on the results of prior action and loops back in a circle to affect the original state
  - amplifying and balancing feedback loops
Fractal structures → nested parts of a system are shaped into the same pattern as the whole
- self-similarity
- software design patterns may contain other patterns…

Co-evolution → continual interaction among complex systems; each system forms part of the environment for all other systems
- system of systems
- simultaneous and continual change
- species survive that are most capable of adapting to their environment as it changes over time
A Vague Concept

Not always clear what “complexity” is measuring...

Characteristics include difficulty of implementing, testing, understanding, modifying, or maintaining a program.

Potential Software Complexity Measures

Lines of code

Source lines of code

Number of functions

McCabe cyclomatic complexity
  • maximum of all functions
  • average over functions

Coupling and cohesion
Halstead’s software science
  • length
  • volume
  • level
  • mental discriminations

Oviedo’s data flow complexity

Chidamber and Kemerer’s object oriented measures

Knot measure
  - for a structured program, the knot measure is always 0
Fan-in, fan-out

Henry and Kafura’s measure depends on procedure size and the flow of information into procedures and out of procedures.

- length x (fan-in x fan-out)

And so forth…
(Source) Lines of Code

LOC – total number of lines in a source code file, including comments, blank lines, etc.
- countable using the Unix wc utility

SLOC – any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line
• includes program headers, declarations, executable and non-executable statements

McCabe Cyclomatic Complexity

In the control flow graph for a procedure reachable from the main procedure containing
• N nodes
• E edges
• p connected procedures
  - only procedures that are reachable from the main procedure

\[ V(G) = E - N + 2p \]

For structured programs, \( V(G) = \# \text{ of decisions} + 1 \)

Recommended Ranges for Cyclomatic Complexity

V(G) should be less than 10
  • commonly accepted range

Some recommend less than 5

Some suggest that 10-20 should be classified as “challenging”
Halstead’s Software Science


\( N_1 \) number of operators in a program
\( N_2 \) number of operands in a program
\( \eta_1 \) number of unique operators in a program
\( \eta_2 \) number of unique operands in a program
\( \eta \) program vocabulary = \( \eta_1 + \eta_2 \)
\( N \) program length = \( N_1 + N_2 \)

\( V \) program volume = \( N \times \log_2 \eta \)
\( D \) difficulty = \( \frac{\eta_1}{2} \times \frac{N_2}{\eta_2} \)
\( lv \) level = \( 1 \div D \)
\( E \) effort (number of mental discriminations) = \( D \times V \)

\( B \) number of delivered bugs = \( V \div 3000 \)
Halstead’s E

Halstead’s E is in terms of discriminations per second
• Stroud number is 18 discriminations / second

To convert Halstead’s E to person months, one correction factor is

\[
18 \text{ discriminations/sec} \times 60 \text{ sec/min} \times 60 \text{ min/hr} \times 8 \text{ hr/day} \times 17 \text{ day/mon}
\]

\[= 8,812,800 \text{ discriminations/month}\]

“Halstead Time” is typically measured in minutes.
Weyuker’s Properties of Complexity Measures


1) There exists programs $P$, $Q$, such that $|P| \neq |Q|$.
2) Let $c$ be a nonnegative number. There are only finitely many programs of complexity $c$.
3) There are distinct programs $P$ and $Q$ such that $|P| = |Q|$.
4) There exists $P$, $Q$, such that $P$ is equivalent to $Q$ and $|P| \neq |Q|$.
5) For every $P$, $Q$ then $|P| \leq |P;Q|$ and $|Q| \leq |P;Q|$.
6) a) There exists $P$, $Q$, $R$ such that $|P| = |Q|$ and $|P;R| \neq |Q;R|$.
   b) There exists $P$, $Q$, $R$ such that $|P| = |Q|$ and $|R;P| \neq |R;Q|$.
7) There are $P$ and $Q$ such that $Q$ is formed by permuting the order of the statements of $P$ and $|P| \neq |Q|$.
8) If $P$ is a renaming of $Q$, then $|P| = |Q|$.
9) There exists $P$, $Q$ such that $|P| + |Q| < |P;Q|$.
## Summary of Weyuker’s Findings

<table>
<thead>
<tr>
<th>Property</th>
<th>LOC (statement count)</th>
<th>Cyclomatic complexity</th>
<th>Halstead effort</th>
<th>Data flow complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Scales and Legal Operations
Defects and Reliability

Defect prediction models
  • predict the number of defects in a module or system
  • predict which modules are defect-prone

Reliability models
  • predict failures (usually mean-time-to-failure MTTF)

Be wary of attempts to equate defect densities with failure rates!
Who Uses?

Defect prediction models are used during development.
• by project management and the development team
• to focus effort on the parts of the system that need the most attention
• to understand the impact of selected processes, techniques, and tools on quality

Reliability models can be used during testing to determine where the software is ready to release.
• to understand the quality of the operational software
Causal Factors for Defects

Difficulty of the problem

Complexity of designed solution

Programmer/analyst skill

Design methods and procedures used

Explanatory Variables for Predicting Defects

Size measures (LOC)

Complexity measures
- McCabe cyclomatic complexity
- Halstead software science: effort
- count of procedures
- Henry and Kafura’s Information Flow Complexity
- Hall and Preisser’s Combined Network Complexity

OO structural measures (Chidamber and Kemerer)

Code churn measures
- amount of change between releases
Process change and fault measures
- experience
- number of developers making changes
- number of defects in previous releases
- number of LOC added/changed/deleted
Techniques Used

Regression models
  • multicollinearity is a problem

Factor analysis / principal component analysis

Bayesian belief networks

Artificial neural networks

Capture-recapture
Bayesian Belief Networks

Fenton and Neil (1999) concluded that Bayesian belief nets were the best solution.
- Explicit modeling of “ignorance” and uncertainty in estimates, as well as cause-effect relationships.
- Makes explicit those assumptions that were previously hidden - hence adds visibility and auditability to the decision-making process.
- Intuitive graphical format makes it easier to understand chains of complex and seemingly contradictory reasoning.
- Ability to forecast with missing data.
- Use of “what-if?” analysis and forecasting of effects of process changes.
- Use of subjectively or objectively derived probability distributions.
- Rigorous, mathematical semantics for the model.
Capture-Recapture

Uses the overlap between the sets of defects found by different reviewers to estimate residual defects.

Assumptions
- reviewers work independently of each other
- searching is performed before, and not during, an inspection meeting

If the overlap is large, few defects are left to be detected.

If the overlap is small, many faults are undetected.
History of Capture-Recapture

First known use of capture–recapture was by Laplace (1786), who used it to estimate the population size of France.

In biology, capture–recapture is used to estimate the population size of animals in an area.
**Lincoln-Petersen Method**

\[ N = \frac{M \times C}{R} \]

- **N** – estimate of total population size
- **M** – total number of animals captured and marked on the first visit
- **C** – total number of animals captured on the second visit
- **R** – number of animals captured on the first visit that were then recaptured on the second visit
Example

Capture 10 specimens on a first visit and mark them

Capture 15 specimens on a second visit
  • 5 are marked from the first visit

\[ N = \frac{M \times C}{R} = \frac{10 \times 15}{5} = 30 \]
Chapman Estimator

A less biased estimator for small samples

\[ N = \frac{(M + 1) (C + 1)}{R + 1} - 1 \]

\[ \text{var}(N) = \frac{(M + 1) (C + 1) (M - R) (C - R)}{(R + 1) (R + 1) (R + 2)} \]

Example
- \( N = \frac{(10 + 1) (15 + 1)}{(5 + 1)} - 1 = 29.3 \)
- \( \text{var}(N) = \frac{11 \times 16 \times 5 \times 10}{6 \times 6 \times 7} = 34.9 \)
- \( \text{std}(N) = \sqrt{34.9} = 5.9 \)
Capture-Recapture Models in Software Engineering

Basic model (M0) assumes that all faults are equally probable to be found and that all reviewers have equal abilities to find faults.

Mh model – the probabilities of fault detection vary

Mt model – abilities of reviewers vary

Mth model – both the probabilities of fault detection and the abilities of reviewers vary
Capture-Recapture Estimators

M0
• M0–ML – maximum likelihood (Otis, 1978)

Mt
• Mt–ML – maximum likelihood (Otis, 1978)
• Mt–Ch – Chao’s estimator (Chao, 1989)

Mh
• Mh–JK – Jackknife (Burnham, 1978)
• Mh–Ch – Chao’s estimator (Chao, 1987)

Mth
• Mth–Ch – Chao’s estimator (Chao, 1992)
Challenges in Using Defect Prediction Models (Fenton, 1999)

Difficult to determine in advance the seriousness of a defect

Great variability in the way systems are used by different users, resulting in wide variations of operational profiles

Difficult to predict which defects are likely to lead to failures (or to commonly occurring failures)
- 33% of defects led to failures with a MTTF greater than 5,000 years
- proportion of defects which led to a MTTF of less than 50 years was around 2%
Software Reliability


Probability of failure-free operation of a computer program for a specified time in a specified environment.

Reliability is defined with respect to time.
- execution time
- calendar time

Characterizing failure occurrences in time
- time of failure
- time interval between failures
- cumulative failures experienced up to a given time
- failures experienced in a time interval
The Random Nature of Failures

Mistakes by programmers, and hence the introduction of defects, is a complex, unpredictable process.

Conditions of execution of a program are generally unpredictable.

Failure behavior is affected by two principal factors:
- number of defects in the software being executed
- execution environment or operational profile of execution
Nonhomogenous Processes

A random process whose probability distribution varies with time is called nonhomogeneous.

Musa’s basic execution time model and logarithmic Poisson execution time model assume that failures occur as a (NHPP) nonhomogeneous Poisson process.

As the software is debugged, defects are removed

- changing failure profiles
- a nonhomogeneous process
Predicting Reliability

Stochastic reliability growth models can produce accurate predictions of the reliability of a software system providing that a reasonable amount of failure data can be collected for that system in representative operational use.

Unfortunately, this is of little help in those many circumstances when we need to make predictions before the software is operational.

What Is Quality Today?

A collection of quality attributes
• some are relevant to a given project, some are not

Budget and schedules are important “quality” factors for custom software development

Failures are a critical quality attribute from the customer’s perspective
• low failure rates are necessary but not sufficient for customer satisfaction

It is important for developers to measure and understand defects to achieve the other quality factors.
**Operationally Defining Software Quality**

What quality attributes of the application are important to the customer and users?

How will we measure these attributes?

How will we verify they have been achieved?
  - are they testable?
  - must we simulate the operational environment?

How can we predict what they will be in operation?
  - operational profiles vary at any time and change over time
Questions and Answers