Product Quality Engineering
Objectives

- Identify aspects of quality beyond functionality and few defects/failures
  - Performance, availability, usability, etc.
- For selected quality attributes
  - Define the concept
  - Identify engineering practices to provide the attribute
  - Identify testing and other measures to gather indicators of the attribute
# Q vs q

- Quality includes many more attributes than just absence of defects:

<table>
<thead>
<tr>
<th>Features</th>
<th>Evolvability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance</td>
<td>Extensibility</td>
</tr>
<tr>
<td>Availability</td>
<td>Modifiability</td>
</tr>
<tr>
<td>Safety</td>
<td>Portability</td>
</tr>
<tr>
<td>Security</td>
<td>Scalability</td>
</tr>
<tr>
<td>Reusability</td>
<td></td>
</tr>
<tr>
<td>Usability</td>
<td>Cycletime</td>
</tr>
<tr>
<td></td>
<td>Cost</td>
</tr>
</tbody>
</table>
Quality includes many more attributes than just absence of defects:

- Features
- Performance
- Availability
- Safety
- Security
- Reusability
- Usability
- Evolvability
- Extensibility
- Modifiability
- Portability
- Scalability
- Cycletime
- Cost

Addressed in lecture
ISO9126 Attribute Classification
ISO/IEC 9126 Software engineering -- Product quality

Reliability
Maturity
Fault-tolerance
Recoverability

Functionality
Suitability
Accurateness
Interoperability
Compliance
Security

Usability
Understandability
Learnability
Operability

Portability
Adaptability
Installability
Conformance
Replaceability

Efficiency
Time behavior
Resource behavior

Maintainability
Analyzeability
Changeability
Stability
Testability
Measurement Information Model
Product Quality Engineering

Objectives

• Attribute goals
• Criticality of goals
• Preferred tradeoffs

Design

Analysis

• Quantitative / Qualitative
• Fidelity varies with effort, available info

Development

Measurement

• Testing & Field Data
• Customer Feedback
Functionality (Features)

- Requirements process defines objectives
  - Includes decisions about release phasing
  - Also address interoperability, standards compliance, …
- Requirements quality engineering practices
  - Prototyping, customer interaction for early defect detection
  - Requirements checklists (and templates) for defect elimination
  - Domain modeling for completeness and streamlining
  - Feasibility checking is a preliminary analysis step
- Analysis at requirements and design time
  - Sequence/interaction diagrams for use cases
  - Exploring alternative scenarios
  - May use formal methods to analyze consistency & completeness
- Acceptance testing measures success in feature delivery
- Customer satisfaction is the ultimate measure
Performance Engineering Practices

- Specify performance objectives
  - Even where user does not have specific requirements, useful to set performance targets
- Analyze designs to determine performance
  - Use performance benchmarking to obtain design parameters
  - Performance modeling and simulation, possibly using queuing and scheduling theory, for higher fidelity results
- Performance testing
  - Benchmarking (individual operations), stress testing (loads), soak testing (continuous operation)
Performance Objectives: Examples

- **Response Time**
  - Call setup: < 250 ms
  - System startup: < 2 minutes
  - Resume service within 1.5 sec on channel switchover

- **Throughput**
  - 1000+ call requests /sec

- **Capacity**
  - 70+ simultaneous calls
  - 50+ concurrent users

- **Resource Utilization**
  - Max 50% CPU usage on full load
  - Max 16MB run time memory
  - Max bandwidth: 96 kb/sec
Performance Analysis

- Example: Spell checker
  - If you were building a spell checker that searched words in a document against a wordlist, what will be its performance?
- Gives very approximate results
- Useful to get an idea of whether the performance goals are:
  - Impossible to meet
  - A significant design concern
  - A “don’t care” (can be met easily)
- Helps to identify bottlenecks: which parts of the design need to worry most about performance?
Metrics for Performance

- **Within project:**
  - Performance targets (requirements)
  - Estimated performance (design)
  - Actual performance (testing)

- **Across projects:**
  - Metrics available for some domains
    - For example, polygons/sec for graphics, packets/sec for networks
    - Can measure performance on “standard” benchmarks
  - But overall, no general performance metrics
Measuring Performance

- Benchmarking operations:
  - Run operation 1000s of times, measure CPU time used, divide to get average time
  - Need to compensate for system effects: load variations, caches, elapsed vs. CPU time, etc.
- Performance testing:
  - Execute operations using applications – benchmark performance
- Performance is very sensitive to configuration
- Load testing: performance testing under typical and high-use operating conditions, where there may be multiple concurrent requests active simultaneously
Availability Engineering Practices

- Defining availability objectives similar to reliability
  - Based on cost impacts of downtime
- Design techniques for availability
  - Implement fault-tolerance at software and hardware levels
- Availability analysis:
  - Fault trees to determine possible causes of failures
    - FMEA: Failure modes and effects analysis
    - Sort of like fishbones!
  - Attach MTBF numbers to entries and propagate up the tree
  - Combine with recovery times to get estimated downtime
Availability Testing & Metrics

- **Availability testing:**
  - Fault injection: introduce faults, study recovery behavior
  - Fault injection capabilities built into code
  - Study failure behavior during system tests: reliability & availability

- **Availability metrics:**
  - % of time system needs to be up and running (or)
  - % of transactions that must go through to completion

- **Availability goals of 99.9% not unusual**
  - 8 hours of downtime per year

- **Availability goal of 99.999% (“5 NINES”) for telecom etc.**
  - Less than 5 minutes downtime per year, including upgrades
  - Requires upgrading the system while it is operational
Usability Engineering Practices

- Specify usability objectives
  - Often internal to development team
  - May be either quantitative or qualitative
- Workflow observation and modeling, user profiles
- Create interface prototype, analyze for usability
  - Interface concept has primary impact on usability
  - State machine models for navigation design and analysis
- Add usability “widgets” to improve usability properties
- Analysis and testing:
  - Assess usability based on operational profiles
    - Keystrokes/clicks/number of steps for frequent operations
  - Assess usability using surveys: SUMI standardized survey tool
  - User observation testing: watching actual users try to get work done
- Alpha/beta testing
Usability Objectives: Examples

- **Usability:**
  - User types: Administrators & Operators
  - Look and feel same as Windows packages (compliant with Windows Style Guide)
  - Server invocation in < 60 ms
  - Invocation command shall have < 5 Command line arguments
  - Expert user should be able to complete the task in < 5 sec
  - New users to start using the system in one hour without training
  - Context sensitive help for most of the common operations
  - SUMI rating of 48 or higher
SUMI: Software Usability Measurement Inventory

- SUMI is a survey-based approach for usability analysis
  - Standard user questionnaire – 50 questions
  - Pre-calibrated response analysis tool
    - Constantly calibrated against 100s of major software products
    - Score is relative to state-of-the-art
  - Score of 0-10 along five dimensions: efficiency, learnability, helpfulness, control, affect
- Inputs: Actual interface and software behavior, prototypes
- SUMI score is a metric for usability

Usability: Quality Engineering

- Various guidelines on what to do, not to do:
  - User Interface Hall of Shame, Hall of Fame
- Focus on eliminating various kinds of problems:
  - Widget choices to eliminate input errors
    - Such as a calendar to choose date instead of typing
  - Graying out to eliminate invalid choices
  - Input validation
  - Fault detection & handling model to eliminate crashes
  - Standardized libraries of UI widgets within applications, to eliminate inconsistencies
Quick Summary of Usability Engineering

- UI design needs to focus first on the basics, then on the cosmetics
- Focus on user characteristics, expectations and the operations they want to perform
- Consistent interface concept is the most critical part of UI design
- “Obvious” behavior is good!
- Need to figure out and use the right widgets for each UI task
- Cosmetic aspects are nice add-ons after the basics in place
- Usability is about users getting things done and feeling comfortable using the software, not about impressing them!
Evolvability Engineering

- Identifying evolvability objectives:
  - Likely types of future changes
- Designing with evolvability in mind:
  - Most design patterns focus on evolvability
  - Note tradeoffs: designs that increase evolvability along one dimension may reduce evolvability along others
    - For example, with OO, easier to add classes & behaviors, harder to make some types of changes to operations (affects multiple classes)
- Evolvability analysis with SAAM:
  - SAAM: Software Architecture Analysis Method
  - Review-based technique that analyzes the architecture to determine how hard it is to make certain types of changes
  - It is possible to analyze for subjective/qualitative attributes!
Evolvability Objectives: Examples

- **Portability**
  - Application should run on Windows 7 as well
  - Should be able to use different databases Oracle/SQL Server/...

- **Scalability**
  - Increase the number of state vectors in the space communications network from 66 to 110

- **Extensibility**
  - Should be easy to incorporate password protection
  - Medium effort to add context sensitive help feature to the GUI
  - Diagnostic monitoring tool should be extensible with respect to analysis capabilities for monitored data

- **Maintainability**
  - The tool should allow easy addition of new message formats
  - The tool should be customizable for new business processes
Evolvability Engineering Practices

- Addressing (only) those types of changes that are likely
  - Avoiding over-engineering
  - Refactoring approach from agile processes
- Generating multiple design options and comparing their quality attributes
- Matching concerns with solutions: design patterns thinking
- Design-by-contract, built-in self-tests, test suites
  - To provide early detection of failures due to changes
- Changes during development itself provide feedback on evolvability
### Product Quality Data Chart

#### Key Product-Quality Attributes (Performance, Usability...):

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Goal</th>
<th>Arch/Design based Projection</th>
<th>Test Results</th>
<th>Benchmark Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Product Evolution Goals:

<table>
<thead>
<tr>
<th>Evolution Req</th>
<th>Goal</th>
<th>Arch/Design based Projection</th>
<th>Action plan</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Availability Goal

- Nines goal
- Nines Estimated
- Nines Achieved

#### Usability score from SUMI (if used):

- Overall Score
- Efficiency
- Affect
- Helpfullness
- Control
- Learnability

© Motorola India Electronics Ltd, 2000
Summary

- Product Quality encompasses a number of attributes: “ilities”

- It is possible to systematically focus on each attribute
  - Specify objectives, analyze designs, measure results during testing
  - Specific engineering practices to achieve given quality attributes

- Objective metrics exist for some attributes but not others
  - But subjective data is also useful