LiteRM quality model

This generic quality model contains a mixture of quality attributes (over 60), quality attribute groups and subgroups (9), and attribute properties (over 30). Its purpose is to help developers:

1. identify relevant quality attributes and their properties including dependencies
2. specify feasible quality requirements with measures and indicators
3. define an achievement strategy including attribute-specific support tactics
4. define a verification strategy including attribute-specific verification tactics
5. define a monitoring strategy for critical qualities
6. save usable attribute-specific information from lessons learned or other sources
7. access attribute-specific resources

You can change any aspect of the model. In fact, you are expected to. While this model can be directly tailored into project quality specifications, we recommend the following. An organization tailors this generic model to its preferences. This means deleting irrelevant quality attributes (by moving them to the irrelevant quality attributes group), adding others, deleting and adding attribute properties, and changing terminology. Also adding any other information that is likely to help a project dealing with quality goals. This tailoring creates the organizational quality model. This organizational model is then tailored by projects to their specific quality goals to create the project quality specifications.

INTERNAL quality attributes

This model organizes quality attributes into 3 main groups.

Internal qualities are those that can only be directly experienced by developers. This means that testing plays little to no role in the verification of these attributes.

Internal qualities can be indirectly experienced by users, for example when failures take a long time to fix.

Internal qualities support some or all other qualities.

Basic quality group

These qualities support all qualities or all external qualities.

They are NOT explicitly shown as supports in the outline.

Understandability

Definition Ease of both static and dynamic understanding of functions, functional organization, interfaces, architecture, detailed design, and implementation. Understandable design and code implies a clear statement of what, how, when, and why. Understandability entails understanding of individual components and interactions within and between groups of components that share a specified goal.

Assumptions/Rationale

Task-adequate understanding of a coded solution depends on:

- Understanding of application domain and its language
- Essential complexity of the goals
- Feasibility and essential complexity of any solution (necessary complexity)
- Understanding of applicable solution subpatterns
- Presentation of solution (conceptual integrity and organization)

- Predictability of the code (predictability)

Richness of reader's solution experience supports the first 4 items

Richness of reader's coding language experience supports the last item

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average time to understand functions, functional organization, interfaces, architecture, detailed design, and implementation
- Design and code quality measures

Operational Measures

- Number of times misunderstanding is identified as a failure factor by root cause analysis
- Maximum severity of misunderstanding-related failures

Aspect of every other quality

Supported by – always or sometimes code readability, conceptual integrity

Associated with verifiability, compliance

Conflicts with performance qualities

Threats unnecessary complexity, redundancy, inconsistency, poor organization

Mitigations use of coding and interface design standards that promote understandability
Standards should: (1) contain naming conventions, (2) require single responsibility classes and components, (3) require prolog describing intent and achievement strategy and comments describing complex algorithms.

Other achievement tactics

- Encourage reuse of high-quality code and design patterns, especially for crosscutting concerns
- Coding and interface design standards that promote understandability

Achievement Questions

- Which aspects of the design or code are the most difficult to understand?
- Which tactics of expression clarify these aspects?

Verification tactics

- Review standards compliance
- Measure complexity and speed of learning
- Verify all supporting qualities

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Source Code Maintainability Analysis OMG/CISQ

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Conceptual Integrity

Definition Degree to which components are few, simple, and clearly organized

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Measures of design quality
- Amount of unusable or redundant design elements

Operational Measures

- Number of confusing design issues identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of understandability

Supported by – always or sometimes necessity, consistency, domain alignment, modularity qualities

Associated with code readability

Conflicts with

Threats unnecessary complexity, redundancy, inconsistency

Mitigations simple implementations, components, organizations, interfaces, and protocols

Other achievement tactics

- Measure and reduce unnecessary complexity
- Coding and design standards that minimize complexity

Achievement Questions

- Which components are the most complex?
- How can these complex components be simplified?

Verification tactics

- review standards compliance
- measure complexity
- verify all supporting qualities

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Necessity

Definition Degree to which components and component content support adequacy

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

Number of necessity defects e.g. unusable or redundant design elements, identified in a technical review

Operational Indicators

Number of unused design elements

Aspect of conceptual integrity

Supported by – always or sometimes

Associated with consistency, domain alignment, modularity

Conflicts with

Threats

- gold plating
- overly simplistic

Mitigations

Other achievement tactics

Achievement Questions

Which components or component contents are unnecessary?

Verification tactics

Random reviewer can explain need and describe impact of removal for 10 randomly selected system elements

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Consistency

Definition Degree to which components are similar or predictable

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of consistency defects identified in a technical review

Operational Measures

- Number of consistency defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of conceptual integrity

Supported by – always or sometimes

Associated with necessity, domain alignment, modularity

Conflicts with

Threats Unnecessary differences

Mitigations

Other achievement tactics

Coding and design standards that promote consistency

Achievement Questions

- Which components or component contents are surprising?
- Are the surprises necessary?

Verification tactics review standards compliance

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Domain alignment

Definition Degree to which components match domain objects and functions

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of domain alignment defects identified in a technical review

Operational Measures

- Number of domain alignment defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of conceptual integrity

Supported by – always or sometimes

Associated with necessity, consistency, modularity

Conflicts with

Threats

Mitigations

Other achievement tactics
Use of domain vocabulary

Verification tactics
Review domain model with domain experts

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

MODULARITY subgroup

Definition The degree to which a system is composed of discrete components.

Aspect of Conceptual integrity

Associated with necessity, consistency, domain alignment

Other properties

b. Type = composite basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

Cohesion

Definition Degree to which components are semantically and logically organized and tightly coupled

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of inadequate cohesion defects identified in a technical review

Operational Measures

- Number of inadequate cohesion defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of modularity

Supported by – always or sometimes

Associated with coupling

Conflicts with

Threats

Mitigations

Other achievement tactics
Coding and design standards that promote cohesion

Verification tactics
Review standards compliance

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Coupling

Definition Degree to which components are loosely coupled

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of excessive coupling defects identified in a technical review

Operational Measures

- Number of excessive coupling defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of modularity

Supported by – always or sometimes

Associated with cohesion

Conflicts with performance qualities

Threats

Mitigations

Additional achievement tactics
Coding and design standards that promote weak coupling

Verification tactics
Review standards compliance

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Code Readability

Definition Ease of understanding what each code element and module is and what it does.

Assumptions/Rationale

Readability depends on:

- programming language-level (the higher the better) - modularization can be used to raise the language level
- understanding and predictability of elements in the programming language subset and design modules

- program organization and statement formatting
- code complexity e.g. length and cyclomatic complexity
- reader knowledge and experience

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Developers can accurately describe behavior after minimal period of technical review
- Number of readability defects identified in a technical review

Operational Measures

- Number of readability defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of understandability

Supported by – always or sometimes conceptual integrity, essential complexity, organization, and predictability

Associated with

Conflicts with

Threats

- unnecessary complexity
- redundancy
- unnecessary differences

Mitigations

Other achievement tactics

- consistent, appropriate, and unambiguous vocabulary and style and precision of expression
- appropriate level and amount of documentation and contextual help
- coding and design standards about labeling and annotation

Verification tactics

- review standards compliance
- review documentation and contextual help
- measure complexity and developer ability to describe behavior
- verify supporting qualities

Associated Tools

- Measurement
- Achievement
- Verification

Resources

"Code Evasion" IEEE Software

"How Not to Write FORTRAN in any Language" Don Seeley ACM Queue

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Essential complexity

Definition Degree to which code complexity is needed for adequacy

Assumptions/Rationale

Depends on the complexity inherent in the requirements and the complexity of the tactics available to satisfy them

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Cyclomatic complexity of the code
- Lines of unreachable code
- Number of unnecessary complexity defects in algorithm or data identified in a technical review

Operational Measures

- Number of unnecessary complexity defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of code readability

Supported by – always or sometimes

Associated with organization, predictability

Conflicts with performance qualities

Threats unnecessary complexity

Mitigations

- refactor the code
- decompose when useful to reduce complexity

Other achievement tactics

Verification tactics

- set standards for complexity
- measure complexity

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Organization

Definition Degree to which information is clearly organized, formatted into sections, subsections, paragraphs, and sentences, and informatively labeled to facilitate reader understanding.

Assumptions/Rationale

> Organizations that facilitate the clean chunking of information into domain and solution concepts facilitate reader understanding

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Ability of developers to describe organization of the code and data after a quick review
- Ability of code analyzers to accurately display structure of the code
- Number of blocks and lines that are "too long"
- Number of confusing organization defects identified in a technical review

Operational Measures

- Number of confusing organization defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of code readability

Supported by – always or sometimes

Associated with essential complexity, predictability

Conflicts with performance qualities

Threats failure to comply with coding standards

Mitigations coding standards compliance verification

Other achievement tactics

Verification tactics analyze and review standards compliance

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Predictability

Definition Ease of accurately predicting the behavior of code elements and modules.

Assumptions/Rationale

Some programming language elements are “unpredictable”. This can be partially controlled by coding standards.

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Ability of developers to accurately predict behavior of the code after a quick review

- Number of coding standard violations related to unpredictable constructs identified by an analyzer or technical review

Operational Measures

- Number of confusing construct defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of code readability

Supported by – always or sometimes

Associated with essential complexity, organization

Conflicts with

Threats poorly-defined language properties variably implemented by a compiler

Mitigations don't use poorly-defined properties

Other achievement tactics

Identify functions to monitor, measure, and trace

Verification tactics

- review standards compliance
- monitor, measure, and trace behavior

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

VERIFIABILITY subgroup

Definition Ease of assessing whether a software configuration satisfies its requirements

Software subfield verification, validation, and test

Aspect of every other quality

Resources

The Quest for Software Requirements -- section 6.4

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

g. Visibility group = internals

Notes

5.1.2 Capers Jones Software Quality Survey

Capers Jones has analyzed the effects that general practices, such as inspections and static analysis, have on improving software quality [Jones 2012].

Table 4 lists the practices associated with poor quality software—less than a 25% success rate for improving quality and good results—greater than a 95% success rate for improving quality. The best results are associated with a combination of these techniques (i.e., the results of a combination of techniques increases the assurance associated with the design and implementation).

Table 4: Quality Practices Study

Poor Results		Good Results	
Testing as only form of defect removal	LOC Metrics for quality (omits non-code defects)	Formal inspections (requirements, design, and code)	Code static analysis
Informal testing and uncertified test personnel	Failure to estimate quality or risks early	Requirements modeling	Testing specialists (certified)
Testing only by developers; no test specialists	Passive quality assurance (< 3% QA staff)	Defect detection efficiency (DDE) measurements	Root-cause analysis
LOC metrics for quality (omits non-code defects)		Defect removal efficiency (DRE) measurements	Active quality assurance (> 3% SQA staff)
		Automated defect tracking tools	

Testability (externals and mixed)

Definition Ease of assessing whether a software configuration (i.e., programs, data, and procedures) satisfies its external and mixed requirements by executing the configuration.

A deep neural net trained by a comprehensive set of positive and negative examples has limited testability because its precise recognition boundaries are unpredictable.

Software subfield verification, validation, and test

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Complexity of logical conditions in the code
- Cyclomatic complexity of the code
- Number of use cases needed to cover the code
- Number of test cases needed for required test coverage
- Mutation score

Operational Measures

- Number of "missing test" defects identified as failure factors by root cause analysis
- Maximum severity of these failures

Aspect of verifiability

Supported by – always or sometimes observability, controllability, repairability subgroup, recoverability, reliability

Associated with reviewability, analyzability, measurability

Conflicts with robustness, internal performance qualities

Threats Complexity

Mitigations

Other achievement tactics

- If-then table specs
- Minimize component complexity e.g., cyclomatic complexity
- Localize state storage
- Abstract data sources
- Test frameworks
- Record and Playback
- Isolate system for experimentation
- Assess code reachability by measuring the complexity of the simplest logical expression to reach

Verification tactics

- Review standards compliance
- Measure code complexity
- Measure code and data coverage
- Assess test results

Review Questions

- How will exception handlers and condition checkers be tested?
- Which test coverage criterion is appropriate?
- Which tests should be automated and how?
- Are there test data security or privacy issues?
- Are scarce resources needed for testing?
- Are there liability issues to be addressed by testing?
- How will acceptance and limited release testing be managed?
- Who will determine if each form of testing is adequate?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Architecture in Practice -- chapter 10

Exploratory Testing Explained -- Bach

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Observability

Definition Ease of observing the executions and outcomes of a system

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Ease of determining test paths and test outcomes

Operational Measures
Ease of detecting failures

Aspect of testability, analyzability, measurability, ease of auditing, ease of operating

Supported by – always or sometimes monitorability

Associated with controllability, repairability, recoverability

Conflicts with privacy, resource security, performance qualities

Threats

Mitigations

Other achievement tactics

- Use specialized interfaces
- Use test monitors
- use coverage analyzers
- use automated oracles

Achievement Questions

- Which aspects of executions and outcomes are the most difficult to observe?
- Which design tactics can clarify these aspects?

Verification tactics

- Assess ease of determining results
- Assess ease of evaluating test suite

Review Questions

- How to create a snapshot of the system's state to use for troubleshooting?
- How to create custom instrumentation to provide detailed operational reports?
- How to discover details of the requests sent to the system?
- How to monitor system activity and performance?
- How to implement tracing?
- How to provide troubleshooting support?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Monitorability

Definition Degree of exception handling, logging, and self-checking done in a system to monitor partial and final results during execution

Software subfield verification, validation, and test

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Density of self-checks in the code
- Number of independent monitors
- Logging of execution flow, data access and changes, intermediate results, and timing

Operational Measures

Number and severity of unmonitored failures

Aspect of observability

Supported by – always or sometimes reliability

Associated with

Conflicts with performance qualities

Threats

Mitigations

Other achievement tactics

- Monitor system condition, states, and state sequences
- Use component self-checking as well as independent monitors
- Conditional logging of execution flow, data access and changes, intermediate results, and timing

Achievement Questions

- Which states, behaviors, and results are the most difficult to monitor?
- Which design tactics can reduce these difficulties?

Verification tactics

- Review monitor placements
- Test monitor operations
- Review logs
- Analyze undetected failures

Review Questions

- How will monitors be tested?
- Is any code under-monitored or over-monitored?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Controllability

Definition Ease of influencing the flow of control and data availability during test execution.

Assumptions/Rationale

Controllability and observability may determine the effectiveness of testing

Leading Indicators -- provide preoperational evidence of quality goal achievement

Ease of controlling test path execution and stored data usage

Operational Measures

Number and severity of failures involving statements or stored data that was unused during test

Aspect of testability

Supported by – always or sometimes monitorability

Associated with observability, repairability subgroup, recoverability

Conflicts with safety, resource security

Threats dead code, any quality function that does not have an external trigger or is difficult to reach e.g. exception handlers

Mitigations use of testpoint processor to create specific conditions

Other achievement tactics

- Specialized interfaces
- Built-in test controls
- Coding standards that limit nondeterminism

Verification tactics

- Test interfaces and test controls
- Detect unreachable code

Review Questions

- Which code will be hard to reach?
- How will exception handlers be reached?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

REPAIRABILITY subgroup

Definition The ease of fixing system components and restoring the system to operational status

Aspect of testability, internal durability group

Supported by – always or sometimes reliability

Associated with observability, controllability, recoverability

Conflicts with performance qualities, privacy, resource security

Resources

The Quest for Software Requirements -- section 6.2 (corrective maintenance)

Source Code Maintainability Analysis OMG/CISQ

Other properties

b. Type = composite basic quality

c. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

Diagnosability

Definition Ease of locating and identifying system faults

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average time to locate and identify defects causing test failures
- Ratio of accurate diagnoses to all diagnoses during test

Operational Measures

- Average time to locate and identify defects causing production failures
- Ratio of accurate diagnoses to all diagnoses during production

Aspect of repairability

Supported by – always or sometimes observability, reliability

Associated with modifiability, restorability

Conflicts with (see repairability subgroup)

Threats complexity

Mitigations

Other achievement tactics

- Maximize fault isolation via extensive self-checking of component inputs and results
- Log component entry and exit conditions and data values

Achievement Questions

- Which faults will be difficult to isolate and identify?
- Which design tactics will help?

Verification tactics

- Review standards compliance
- Analyze and test self-checks
- Test logging
- Measure and track average time to fault isolate

Review Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Debugging by David J Agans

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = repair quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Modifiability

Definition Degree to which a system minimizes the cost and risk of making changes. We can distinguish functional and data modifiability.

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average number of locations changed during a test change
- Average time to make a change during test
- Ratio of successful changes to all changes during test

Operational Measures

- Average number of locations changed during an operational change
- Average time to make changes during production
- Ratio of successful changes to all changes during production

Aspect of repairability

Supported by – always or sometimes (see repairability subgroup)

Associated with diagnosability, restorability

Conflicts with (see repairability subgroup)

Threats complexity

Mitigations

Additional achievement tactics

- Reduce module size
  a. Split modules
- Reduce coupling with cohesive design
  a. Encapsulate
  b. Use an intermediary
  c. Restrict communication paths
  d. Promote weak coupling
  e. Abstract common services
- Increase cohesion
  f. Separate unrelated functions
- Defer binding
- Design and coding standards focused on modifiability

Achievement Questions

- Which changes are the riskiest?
- Which design tactics can mitigate this risk?

Verification tactics

- Review compliance with modifiability-focused guidelines
- Measure coupling and cohesion

Review Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Architecture in Practice -- chapter 7

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Restorability

Definition Ease of restoring the system to a previous or base state

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average time to restore system during test
- Ratio of successful restores to all restores during test

Operational Measures

- Average time to restore system during production
- Ratio of successful restores to all restores during production

Aspect of repairability

Supported by – always or sometimes (see repairability subgroup)

Associated with diagnosability, modifiability

Conflicts with (see repairability subgroup)

Threats

Mitigations

Other achievement tactics

- Log sequence of base states
- Encapsulate initialization to designated base state
- Design and coding standards focused on restorability

Achievement Questions

- Which aspects of restoration are the riskiest?
- Which design tactics can mitigate this risk?

Verification tactics

- Test base state logging
- Test initialization process

Review Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Recoverability

Definition Ease of reconstructing or replacing damaged elements of a system or of restarting a system after a failure with minimal loss of work results.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Downtime and data loss following each initial acceptance test failure

Operational Measures

- Average time to recover from an operational failure
- Average time to rerun lost production

Aspect of testability, availability

Supported by – always or sometimes reliability

Associated with

Conflicts with

Threats

Mitigations

Other achievement tactics

- Identify backup scope, frequency, and configuration management
- Define and implement restart points and procedures

Achievement Questions

- Where are the most efficient backup points?
- Where are the safest restart points and what is the maximum work loss for each?

Verification tactics

- Review of backup and restart points
- Testing of recovery mechanism

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Reviewability (all)

Definition Ease of using technical reviews to detect system faults.

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of reviewability improvement suggestions during development

Operational Measures
Ratio of reviewability improvement suggestions to production failures

Aspect of verifiability

Supported by – always or sometimes understandability

Associated with testability, analyzability, measurability

Conflicts with

Threats complexity

Mitigations

Additional achievement tactics

Design and coding standards that limit complexity

Achievement Questions

- Which technical workproducts or their contents are the most difficult to review?
- Which tactics can reduce this difficulty?

Verification tactics

- Review compliance with design and coding standards
- Measure code complexity

Review Questions

- Which workproducts should be reviewed and which form of review should be used on each?
- Which stakeholders should be included in each review?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Peer Reviews in Software

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Analyzability (all)

Definition Ease of using various forms of analysis to detect system faults.

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of analyzability improvement suggestions during development

Operational Measures
Ratio of analyzability improvement suggestions to production failures

Aspect of verifiability

Supported by – always or sometimes observability, controllability, understandability

Associated with testability, reviewability, measurability

Conflicts with

Threats complexity

Mitigations

Other achievement tactics
- Conditional logging of execution flow, data access, intermediate results, and timing
- Design and coding standards that limit complexity

Achievement Questions
- Which technical workproducts or their contents are the most difficult to analyze?
- Which tactics can reduce this difficulty?

Verification tactics
Perform a range of verification-related analysis techniques

Review Questions

- Which workproducts and outcomes should be analyzed?
- How will test data trustworthiness (input) and result data trustworthiness (output) be assessed?
- Which forms of analysis should be used and when?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Measurability (all)

Definition Ease of using measurement to detect system faults.

Assumptions/Rationale

Some qualities, such as understandability, are only partially measurable because they depend on things other than the code, such as the knowledge and experience of the "understander".

Leading Indicators -- provide preoperational evidence of quality goal achievement

Number of measurability improvement suggestions during development

Operational Measures

Ratio of measurability improvement suggestions to production failures

Aspect of verifiability

Supported by – always or sometimes observability, controllability, understandability

Associated with testability, reviewability, analyzability

Conflicts with

Threats

Mitigations

Additional achievement tactics

Achievement Questions

- Which technical workproducts or their contents are the most difficult to measure?
- Which tactics can reduce this difficulty?

Verification tactics

attempt to define, evaluate, and interpret measures across a range of crosscutting goals

Review Questions

Is the achievement of each quality goal clearly defined and measurable?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Gilb, Tom "Advanced Requirements Specification: Quantifying the Qualitative"

Simmons, Eric "Quantifying Quality Requirements"

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

b. Type = basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

COMPLIANCE subgroup

Definition Degree to which system complies with laws, regulations, standards, requirements, and guidelines

Aspect of every other quality

Other properties

b. Type = composite basic quality

d. Design scope = universal cc [hom cc, het cc, universal cc, local, none]

Notes

Laws and regulations

Assumptions/Rationale

Implications/Expectations

Leading Indicators -- provide preoperational evidence of compliance
Number of non-compliances detected by analyzers and technical reviews during development

Operational Measures
Number of non-compliances detected during operation

Aspect of compliance

Associated with design and coding standards, verification guidelines

Achievement Questions
- Which laws or regulations are the most difficult to comply with?
- Which tactics can reduce this difficulty?

Verification tactics
Compliance reviews

Review Questions

Associated Tools
- Measurement
- Achievement
- Verification

Resources

Software Systems Architecture - chapter 28

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = basic quality

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Requirement states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Design and coding standards

Assumptions/Rationale

Enforced coding standards are required to support code understandability

Implications/Expectations

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of non-compliances detected by analyzers and technical reviews during development

Operational Measures
Number of non-compliances detected during operation

Aspect of compliance

Associated with laws and regulations, verification guidelines

Achievement Questions

- Which design or coding standards are the most difficult to comply with?
- Which tactics can reduce this difficulty?

Verification tactics
Compliance reviews

Review Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

James Shore The Art of Agile Development
http://www.jamesshore.com/Agile-Book/coding_standards.html

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = basic quality

e. **Consensus Priority** = important [critical, important, desirable]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Requirement states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement and Verification Strategies

Current Quality Goal Spec

Current Achievement and Verification Strategy

Verification guidelines

Assumptions/Rationale

Each quality goal must have an adequate verification strategy outlined early to demonstrate goal understanding.

Testing of each domain and system function must satisfy a code coverage requirement.

Implications/Expectations

The project must have a code coverage requirement and a coverage analyzer.

Leading Indicators -- provide preoperational evidence of quality goal achievement

Number of non-compliances detected by technical reviews during development

Operational Measures

Number of non-compliances detected during operation

Aspect of compliance

Associated with laws and regulations, design and coding standards

Achievement Questions

- Which verification guidelines are the most difficult to comply with?
- Which tactics can reduce this difficulty?

Verification tactics

- Verification strategies for quality goals must be technically reviewed for adequacy.
- Test suites and coverage analyzer reports must be technically reviewed for adequacy.

Review Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Tutorial on MC/DC code coverage
http://shemesh.larc.nasa.gov/fm/papers/Hayhurst-2001-tm210876-MCDC.pdf

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = basic quality

f. Architecture-relevant = no [yes, maybe, no]

States

a. Requirement states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement and Verification Strategies

Current Quality Goal Spec

Current Achievement and Verification Strategy

Internal Behavior group

These goals refer to qualities of internal system behavior.

Internal PERFORMANCE subgroup

Definition The amount of useful work accomplished by a system compared to the time and resources used and the responsiveness of the system

Software subfield performance engineering

Associated with external performance subgroup

Resources

Software Systems Architecture - chapter 25

Software Architecture in Practice -- chapter 8

"Quality Attributes" Technical Report CMU/SEI-95-TR-021 Chapter 3

The Practical Performance Analyst

Automated Source Code Performance Efficiency Measure OMG/CISQ

Other properties

b. Type = composite internal behavior quality

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

Notes

List of performance analysis tools:
http://en.wikipedia.org/wiki/List_of_performance_analysis_tools

Capacity

Definition Number of concurrent processes that a system can handle or permanent storage that a system can provide.

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Maximum number of concurrent processes successfully tested
- Maximum amount of stored data successfully tested

Operational Measures

Number of capacity-based failures during operation

Aspect of Performance

Supported by – always or sometimes

Associated with throughput, efficiency, response time

Conflicts with

Threats Unnecessary locking of common resources, inadequate resources

Mitigations Analyze locking protocols

Other achievement tactics

Identify likely growth areas

Control resource demand

- Monitor concurrent users
- Limit and eject users
- Prioritize events
- Limit event responses
- Minimize data transmission
- Reduce overhead
- Bound execution times

Manage resources

- Schedule resources
- Increase resources
- Increase resource efficiency
- Introduce concurrency
- Asynchronous locking
- Optimize locking protocols
- Maintain multiple copies of computations
- Maintain multiple copies of data e.g. caching stable, non-sensitive data
- Bound queue sizes

Verification tactics

- Comprehensive testing of capacity support mechanisms and of capacity under a variety of loads
- Measure and track number of concurrent processes

Review Questions

- How many concurrent users are estimated for a normal load and peak load?
- How much data storage will be needed to handle a peak load for its estimated duration?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Requirement Patterns -- chapters 9.3 and 9.4

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal behavior quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Efficiency

Definition Production per unit of consumption

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Resource usage and execution time for a reference set of tasks during testing

Operational Measures
Resource usage and execution time for a reference set of tasks during operation

Aspect of performance

Supported by – always or sometimes

Associated with capacity, response time, throughput

Conflicts with - always or sometimes understandability, verifiability, interoperability, data trustworthiness, security, ease of use, adaptability qualities, repairability qualities

Threats

Mitigations

Additional achievement tactics

- Faster algorithms
- Caching stable, non-sensitive data

Architectural patterns to avoid layered groupings of modules, blackboard, presentation abstraction control, microkernal, reflection

Verification tactics

- Analyze algorithms
- Measure and track resource and time utilization

Review Questions

- When does the system use the most resources?
- Where does the system spend most of its time?
- How will efficiency measures be logged?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

The Quest for Software Requirements -- section 5.3

Source Code Efficiency Analysis OMG/CISQ

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal behavior quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Representative Operational Profile: An operational profile that is likely to occur during use of the system after deployment. Specifically not a profile designed to stress the application in ways not possible or rarely encountered in actual use.

Stress Profile: An operational profile designed to cause extreme resource consumption or challenge the system's performance, regardless of whether the profile is likely or even possible to occur in actual use.

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Internal Durability group

These goals extend the useful life of a system in the face of inevitable change.

Interversion compatibility

Definition (1) degree to which newer versions of software work with data from older versions or (2) degree to which newer versions of a component can replace older versions without other changes.

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of interversion-related failures during testing

Operational Measures
Number of interversion-based failures during operation

Aspect of Durability

Supported by – always or sometimes

Associated with modifiability

Conflicts with

Threats

Mitigations

Other achievement tactics

- Map old data to new and new data to old
- Encapsulate new functionality and new data in new version

Verification tactics

- Review data management strategy
- Test working with old and new data and old and new versions

Review Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal durability quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Configuration reusability

Definition The potential for existing component configurations to be reused in a new version or application.

Assumptions/Rationale

Component configurations may be reusable, if **quality levels have been aggressively managed throughout a configuration's evolution**. Code quality can degrade during a series of versions.

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Measures of basic qualities
- Profile of types, severity, and frequency of detected defects in prior usage

Operational Measures

Profile of types, severity, and frequency of detected production defects

Supported by – always or sometimes component reusability, functional integrity, data trustworthiness, modifiability, portability, safety or dependability

Conflicts with performance qualities

Threats

Mitigations

Other achievement tactics

- Document design rationale and assumptions
- Document environmental assumptions
- Design for reuse by minimizing environmental assumptions and dependencies

Verification tactics

analyze design and code to detect design and environmental assumptions and dependences

Review Questions

What are the major risks in this reuse?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal durability quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Component reusability

Definition Potential for and ease of reusing a component in different systems.

Assumptions/Rationale

Components in a library are assumed to be reusable and been designed for reuse. Component reusability requires that: (1) the intent and quality levels e.g., level of reliability, of each component are clearly understood and (2) the component interfaces are published and all interactions are through these interfaces.

Leading Indicators -- provide preoperational evidence of quality goal achievement

Number of reusability-related failures during testing

Operational Measures

- Number of reusability-related failures during operation
- Number of desirable, but infeasible, reuse opportunities

Aspect of **configuration reusability**

Supported by – always or sometimes functional integrity, data trustworthiness, modifiability, portability, safety or dependability

Associated with

Conflicts with performance qualities

Threats

Mitigations

Other achievement tactics

- Prefer object-oriented components
- Prefer stateless components
- Design and coding standards to support component reuse

Verification tactics

Analyze design and code to detect environmental assumptions and dependences

Review Questions

Which standards impact reusability?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

The Quest for Software Requirements -- section 7.3

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal durability quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

ADAPTABILITY subgroup

Definition The ability to change to fit different environments, needs, and preferences

Parametric Adaptation
The variation of algorithm parameters over time. The change in an artificial neural net synapse weights as learning progresses would be an example of Parametric Adaptation.

Algorithmic Adaptation
The switching from one algorithm to another to solve a given problem. The alternation between various vision algorithms as lighting or other conditions change would be an example of Algorithmic Adaptation.

Resource Adaptation
The differing utilization of resources by a running distributed system over time. The relocation of executing code from one processor to another, presumably less encumbered, processor would be an example of Resource Adaptation.

Resources

Software Systems Architecture - chapter 27

The Quest for Software Requirements -- section 6.1

Other properties

b. Type = composite internal adaptability quality

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

Portability

Definition Ease of moving a system to different hardware or software platforms

Assumptions/Rationale A minimum level of dependability is assumed because porting undependable applications may be unwise.

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of portability-related failures during testing

Operational Measures
Number of portability-related failures during operation
Number of desirable, but infeasible, portability targets

Aspect of adaptability

Supported by – always or sometimes component reusability, **dependability**

Associated with extensibility, scalability, internationalizability

Conflicts with precision, performance qualities, safety, security

Threats

Mitigations

Additional achievement tactics

- Isolate platform dependencies in a small set of components that can be replaced during a port
- Design standards that support portability

Architectural patterns to avoid model view controller

Verification tactics

- Review standards compliance
- Comprehensively test system on target platforms

Review Questions

- How many platforms must be supported?
- What additional platform ports are likely?
- What data conversion procedures are needed?
- What unusual devices must be supported on each platform?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

The Quest for Software Requirements -- section 7.2

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal adaptation quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Enhanceability

Definition Ease of adding new: functions, data types, communication channels, and user types

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of enhanceability-related failures during testing

Operational Measures

- Number of enhanceability-related failures during operation
- Number of desirable, but infeasible, enhancement opportunities

Aspect of adaptability

Supported by – always or sometimes modifiability

Associated with portability, scalability, internationalizability

Conflicts with

Threats inflexible communication

Mitigations

Additional achievement tactics

- Prefer object-oriented components
- Prefer stateless components
- Encapsulate communication functions
- Encapsulate authorization
- Design and coding standards that support enhanceability

Verification tactics

Review design and code for standards compliance

Review Questions

What enhancements are likely?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Requirement Patterns -- chapter 10.2

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = internal adaptation quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

EXTERNAL quality attributes

This model organizes quality attributes into 3 main groups.

External qualities are those that are directly and fully experienced by users, both human and automated.

This means that these attributes are primarily verified by testing.

Facilitation quality group

These goals refer to system qualities that make specific activities easier.

Ease of configuring, updating, and operating

Definition Ease of configuring, updating, and operating a system

> Configuring includes ability to adjust: component deployment, displays, behavior, and outcomes manually via parameter values or automatically based on defined conditions

Assumptions/Rationale

> Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average number of steps to accomplish tasks
- Maximum complexity of the steps

Operational Measures

- Average speed to accomplish tasks
- Missteps while performing tasks

Aspect of dependability

Supported by – always or sometimes **visibility**, **internationalizability**

Associated with ease of learning

Conflicts with

Threats hard-coded configuration choices

Mitigations parameterize configuration choices that are not clear cut

Other achievement tactics

- Parameterize configuration choices
- Support normal operation
    a. System status checking
- Support exception handling
    a. backup
    b. recovery and restart
- Support mistaken action recovery
    a. Provide cancel
    b. Provide undo
- Design guidelines focused on ease of configuring and operation

Verification tactics

- Review design and code for missed parameterization opportunities
- Review compliance with design guidelines
- Code review and test each parameterized configuration mechanism
- Test configuring scenarios and operational usage
- Measure and track speed, steps, and missteps

Elicitation Questions

- How to enable the system behavior to change based on operational environment requirements, such as infrastructure or deployment changes?
- How to enable the system behavior to change at run time based on system load; for example, by queuing requests and processing them when the system is available?
- What configuring or operational challenges may occur?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = important [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Ease of installing and uninstalling

Definition Ease of installing and uninstalling a system

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average number of steps to accomplish tasks
- Maximum complexity of the steps

Operational Measures

- Average speed to accomplish tasks
- Missteps while performing tasks

Aspect of

Supported by – always or sometimes reliability, data trustworthiness

Associated with

Conflicts with

Threats

Mitigations

Additional achievement tactics

Develop guide with install/update/uninstall procedures

Verification tactics

- Review install/update/uninstall guide
- Test using guide
- Measure and track speed and missteps

Elicitation Questions

What install, update, or uninstall challenges may occur?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Requirement Patterns -- chapter 10.6

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = important [critical, important, desirable]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Ease of learning

Definition Ease of learning domain functionality

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Maximum number of steps to accomplish tasks
- Maximum complexity of the steps

Operational Measures

- Average time to make proficient use of functions
- Missteps while performing tasks

Aspect of dependability, safety

Supported by – always or sometimes ease of use

Associated with Ease of managing, ease of auditing, ease of configuring, updating and operating

Conflicts with

Threats complexity, inconsistency, unnecessary

Mitigations use of coding and interface design standards that promote understandability

Other achievement tactics

- Organize functions by user type, object, or task
- Include clear, easily accessible help for navigation and functionality
- Coding and interface design standards that promote understanding by limiting complexity, inconsistency, and unnecessary functionality and design elements

Verification tactics

- Review standards compliance
- Measure time to task-adequate understanding

Elicitation Questions

- Which aspects of the system are the most complex?
- Can any aspect be simplified?
- What training is required before use?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

The Quest for Software Requirements -- section 5.7

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = important [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Ease of use

Definition Ease of using a system to accomplish tasks within its domain and scope

Software subfield usability engineering

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average number of steps to accomplish frequent tasks
- Maximum complexity of the steps in frequent tasks

Operational Measures

- Average speed to accomplish tasks by appropriate users
- Missteps while performing tasks

Aspect of ease of learning

Supported by – always or sometimes Functional integrity, error resistance, reliability, ease of access subgroup, compatibility subgroup, **dependability**, **internationalizability**

Associated with Ease of managing, configuring and operating

Conflicts with security and efficiency

Threats too many functions, too complex functions, too much interaction, poor navigation, poor user messages

- careless, indecisive, or tired behavior resulting in mistakes
  mitigate with error resistance and cancel and undo functions
- paranoid behavior resulting in privacy concerns
  mitigate by only requiring necessary information
  and clearly explaining privacy policy and tactics
- forgetful or impatient behavior resulting in forgotten codes or duplicate actions
  mitigate by securely providing codes or opportunity to change codes
  and by detecting duplicate requests and data
- unreliable behavior resulting in unmet obligations
  mitigate with status monitoring and reporting of additional needs

Mitigations

See above

Additional achievement tactics

- Develop behavioral profiles
- Show most recent and upcoming functions
- Provide "do all"
- Provide pause and resume, cancel, undo, undo all
- Organize functions by object or task
- Provide informative exception messages
- Design standards focused on ease of use e.g. hiding functional complexity

Verification tactics

- Review standards compliance
- Test frequent and infrequent tasks
- Measure speed, steps, and missteps for tasks

Elicitation Questions

- What are the different types of users
- Which tasks are frequent and infrequent for each user type (i.e. what is the assumed usage profile)?
- How much effort (e.g. mouse clicks, page scrolls, or keystrokes) and average time does each task take?
- How clear, succinct, and accessible are user instructions and messages?
- How will progress be guided and confirmed?
- Are restart points clear and adequate to minimize reentry?
- How does the interface compensate for user foibles e.g. forgetting, incorrect input, incorrect function selection, etc.?
- Are there sufficient shortcuts for common scenarios?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Systems Architecture - chapter 28

Software Architecture in Practice -- chapter 11

The Quest for Software Requirements -- section 5.7

Software Architecture: Analysis of Usability

Visual Usability: Principles and Practices for Designing Digital Applications

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

EASE OF ACCESS subgroup

Ease of access is partitioned by user type.

Other properties

b. Type = composite facilitation quality

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

Ease of access in different situations

Definition Ease of access by users in different security classes, accessing different functions, using different hardware and software (e.g. operating systems and browsers)

Assumptions/Rationale

Use quality attribute scenarios

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percent of successful accesses in valid scenarios

Operational Measures
Percent of successful accesses in operation

Aspect of Ease of use

Supported by – always or sometimes

Associated with Error resistance, ease of access with specific disabilities

Conflicts with privacy, resource security

Threats
New devices and new software

Mitigations

Other achievement tactics

- Develop user group profiles
- Clear security guidelines with access privileges
- Array of hardware and software handlers
- Include ease of access tactics in interface design guidelines

Verification tactics

- Review compliance with design and security guidelines
- Test each support tactic and device handler

Elicitation Questions

- What are the different user security classes and their accessible functions?
- What different hardware and software configurations are supported?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Just Ask: Integrating Accessibility Throughout Design
http://www.uiaccess.com/accessucd/users.html

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement and Verification Strategies

Current Quality Goal Spec

Current Achievement and Verification Strategy

Ease of access with different disabilities

Definition Ease of access by users with different disabilities including auditory issues, color blindness, visual acuity issues, blindness, physical issues, and mild cognitive issues. It also includes access using common screen readers. This may involve section 508 compliance.

Assumptions/Rationale

Use quality attribute scenarios

Leading Indicators -- provide preoperational evidence of quality goal achievement
percent of successful accesses by covered user types during test

Operational Measures
percent of successful accesses by covered user types during operation

Aspect of Ease of use

Supported by – always or sometimes

Associated with Error resistance, Ease of access in different situations

Conflicts with

Threats

- only readable directions and outputs
- navigation and functions requiring small muscle control
- sensitive to incorrect input

Mitigations

Additional achievement tactics

Perceivability

- text alternatives for non-text content
- audio alternatives
- sign language alternatives

Operability

- all functionality available from the keyboard
- adjustable presentation timing
- help avoiding and correcting mistakes e.g. with audio verification of intent

Understandability

- enable language selection (localization)
- provide definitions for domain-specific words or phrases
- minimize reading level
- consistent navigation

Verification tactics

- Review compliance with design guidelines
- Test each support tactic

Elicitation Questions

- Which specific disabilities are to be supported?
- Is 508 compliance required?

Associated Tools

- Measurement

- Achievement
  Interface guidelines at http://www.w3.org/TR/WCAG/ and https://www.section508.gov/

- Verification
  Tools at http://www.w3.org/WAI/ER/tools/complete

Resources

Software Systems Architecture - chapter 28

Software Requirement Patterns - subsection 8.3

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement and Verification Strategies

Current Quality Goal Spec

Current Achievement and Verification Strategy

Error resistance

Definition System's ability to eliminate, prevent, detect, and possibly correct user input errors

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Percent of correctly handled input errors in test
- Severity of missed input errors in test

Operational Measures

- Percent of correctly handled input errors in operation
- Severity of missed input errors in operation

Aspect of ease of use, dependability

Supported by – always or sometimes

Associated with Ease of access subgroup

Conflicts with

Threats Poor interface design

Mitigations

Other achievement tactics

- Clear indication of system state
- Clear notification of problems and suggested actions
- Only provide sensible alternatives with consequences
- Extensive input verification
- Provide opportunity to review input before commitment

Verification tactics

- Review of user interfaces
- Exploratory testing of erroneous inputs
- Measure and track average time to undetected error and undetected error rate

Elicitation Questions

- What is the range of errors that users can make along with their severity?
- Which erroneous information or actions can't be corrected and are they to be logged?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

User Interface Dependability through Goal-Error Prevention

Human Interface/Human Error

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

g. Visibility group = externals

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Functional Integrity

Definition Degree to which system provides enough functionality to support each of its domain tasks

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of uncompletable tasks discovered during use case testing

Operational Measures
Number of different complaints about missing functionality

Aspect of reliability

Supported by – always or sometimes

Associated with correctness

Conflicts with

Threats

Mitigations

Other achievement tactics

Organize functions by object or task

Verification tactics

- From a domain task perspective, analyze completeness of each set of functions
- Develop and analyze a range of use cases to detect insufficiency

Elicitation Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

COMPATIBILITY subgroup

Organized by compatibility with different objects.

Platform compatibility

Definition Degree to which an application can perform its required functions on multiple platforms (browsers, OSs, hardware configurations)

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percent of successful performances on multiple platforms during test

Operational Measures
Percent of successful performances on multiple platforms during operation

Aspect of Ease of use

Supported by – always or sometimes

Associated with Ease of access, Error resistance, Application compatibility

Conflicts with performance qualities, ease of use by barring browser, OS, or hardware specific capabilities

Threats Use of platform-specific protocols

Mitigations Use common protocols

Additional achievement tactics
Identify common OS and browser protocols and hardware alternatives

Verification tactics
Compatibility test on different platforms

Elicitation Questions

- Which platforms are required?
- Which platforms have the greatest market share?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Cross Browser Compatibility

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = ease of use quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Application compatibility

Definition Degree to which an application can perform its required functions in the context of other applications (associated (e.g. word processor and style checker), similar (e.g. different word processors), and co-resident)

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percent of successful performances in multiple contexts in test

Operational Measures
Percent of successful performances in multiple contexts in operation

Aspect of Ease of use

Supported by – always or sometimes interoperability

Associated with Ease of access, Error resistance, Platform compatibility

Conflicts with

Threats

Mitigations

Additional achievement tactics
Follow OS-specific protocols

Verification tactics
Test multi-application compatibility

Elicitation Questions

- Which associated applications would be useful and do they have an API?
- Which similar applications lead the market?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Interoperability

Definition Degree to which a system effectively exchanges data via interfaces with other systems and shares the interpretation of that data.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percent of successful data exchanges in test

Operational Measures
Percent of successful data exchanges in operation

Aspect of Application compatibility

Supported by – always or sometimes data trustworthiness, (other system) availability,

Associated with

Conflicts with dependability, efficiency, security, ease of installing

Threats vague or ambiguous interface or behavior descriptions

Mitigations

Additional achievement tactics

Manage exchange

- Orchestrate complex exchanges
- Tailor interfaces by adding or removing capabilities

Verification tactics

- Analyze system interfaces
- Comprehensively test data exchanges

Elicitation Questions

- How to handle different data formats from external or legacy systems?
- How to isolate systems through the use of service interfaces or mapping layers?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Architecture in Practice -- chapter 6

The Quest for Software Requirements -- section 7.1

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Ease of managing

Definition Ease of managing a system

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average number of steps to accomplish tasks
- Maximum complexity of the steps

Operational Measures

- Average speed to accomplish tasks
- Missteps while performing tasks

Aspect of dependability

Supported by – always or sometimes ease of auditing

Associated with ease of learning and use

Conflicts with privacy, resource security

Threats

Mitigations

Other achievement tactics

- Status checking
- Performance tuning
- Design guidelines focused on ease of management

Verification tactics

- Review compliance with design guidelines
- Test management usage
- Measure and track speed, steps, and missteps

Elicitation Questions

- Are there several types of managers?
- What tasks should managers perform?
- Are frequent management tasks easy?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Ease of auditing

Definition Ease of auditing a system for required behavior and compliant attributes

Assumptions/Rationale

Quality attribute scenarios [described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Average number of steps to accomplish tasks
- Maximum complexity of the steps

Operational Measures

- Average speed to accomplish tasks
- Missteps while performing tasks

Aspect of ease of managing

Supported by – always or sometimes ease of detecting dishonesty, visibility

Associated with Ease of learning and use

Conflicts with privacy, resource security

Threats

Mitigations

Additional achievement tactics

- Identify aspects to be audited
- Monitor and control system state
  a. Specialized interfaces
  b. Conditional logging of execution flow, data access, intermediate results, and timing
- Coding standards related to ease of auditing issues

Verification tactics

- Identify objects and behaviors to be audited
- Develop audit procedures
- Review code compliance
- Analyze logs

Elicitation Questions

What should be logged and when?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Ease of detecting dishonesty

Definition Effectiveness of software in detecting theft and employee fraud

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Percent of undetected fraud attempts in test
- Severity of undetected fraud attempts in test

Operational Measures

- Percent of detected thefts and frauds
- Severity of undetected thefts and frauds

Aspect of ease of auditing

Supported by – always or sometimes visibility

Associated with

Conflicts with

Threats - Dishonest employees, dishonest customers or browsers, hackers and thieves

Mitigations -

Implement continuous auditing that monitors:

- customer complaints of undelivered items
- voided sales, credit memos, refunds, adjustments to receivables, payables, or inventory, unusual bills, and duplicate payments

Other achievement tactics

- Identify fraud scenarios undetectable by software
- Monitor missing bank reconciliation items, ghost employees, and expense account items
- Perform regular and surprise audits

Verification tactics

- Develop fraud scenarios
- Analyze transactional data for possible indicators of fraud
- Develop and run misuse cases

Elicitation Questions

- What forms of dishonesty are the most expensive?
- How can these forms be detected?
- What forms of dishonesty can the system not detect?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

"Red Flags for Fraud"

"Fraud Prevention and Detection in an Automated World"

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = facilitation quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

External Behavior group

These goals refer to qualities of visible system behavior.

Reliability

Definition Ability of a software product to maintain a specified level of performance when used under specified conditions.

Software subfield reliability engineering

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Relative code churn
- Percentage of initial use case test failures

Operational Measures

- Frequency of patches
- Average time to failure
- Average time between failures
- Failure rate
- Severity of failures

Aspect of availability, modifiability

Supported by – always or sometimes correctness, data trustworthiness

Associated with external performance subgroup, recoverability

Conflicts with efficiency, adaptability qualities

Threats

- Defective code or data
- Hardware failure
- Excessive customization

Mitigations

- Formal review
- Data analysis
- Thorough testing
- Monitoring reliability measures

Additional achievement tactics

- Monitor and control system states and data trustworthiness (e.g. output) with specialized interfaces
- Comprehensive exception handling
- Maximize reuse of reliable components
- Design and coding standards that limit complexity
- Verification guidelines requiring technical reviews, measurement, analysis, and comprehensive usage, code, and data test coverage as well as exploratory testing

Architectural patterns to avoid pipes and filters, reflection

Verification tactics

- Review standards compliance
- Review and test code including exception handlers
- Analyze data
- Measure and track average time to failure
- Verify all supporting qualities

Elicitation Questions

- Which system aspects threaten reliability?
- What is acceptable reliability?
- Which functions require ultra-high reliability?
- How will unreliability of in-house and third-party software be detected and communicated?
- How to handle unreliable external systems?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

The Quest for Software Requirements -- section 5.5

Software Reliability Engineering

Source Code Reliability Analysis OMG/CISQ

Use of Relative Code Churn Measures to Predict System Defect Density

"Code Evasion" IEEE Software

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Correctness

Definition Degree to which a system performs as specified or expected.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of initial use case test failures that are out-of-spec

Operational Measures
Percentage and severity of out-of-spec production failures

Aspect of Reliability

Supported by – always or sometimes accuracy, precision

Associated with functional integrity

Conflicts with

Threats Defective code or data

Mitigations Formal review, data analysis, thorough testing

Other achievement tactics
Extensive input validation and self-checking of component results

Verification tactics

- Verify all supporting qualities
- Formal review
- Data analysis
- Thorough testing

Elicitation Questions
Does the test suite cover the range of functions and conditions?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Accuracy

Definition The degree to which a result is sufficiently close to its necessary value

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of initial use case test failures where results are inaccurate

Operational Measures
Percentage and severity of production failures with inaccurate results

Aspect of Correctness

Supported by – always or sometimes

Associated with Precision

Conflicts with

Threats Incomplete behavioral specification

Mitigations

Additional achievement tactics

Verification tactics

- Formal review
- Comprehensive testing
- Measurement of modes and medians between results and actual values

Elicitation Questions

Does the test suite cover the range of functions and conditions?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Precision

Definition The degree to which results are insufficiently exact.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of initial use case test failures where results are insufficiently exact

Operational Measures
Percentage and severity of production failures with imprecise results

Aspect of Correctness

Supported by – always or sometimes

Associated with Accuracy

Conflicts with Portability

Threats redundant calculations

Mitigations

Other achievement tactics
Design and coding standards that limit redundancy

Verification tactics
- Formal review,
- Comprehensive testing
- Measurement of modes and medians between instances of a result

Elicitation Questions
Does the test suite cover the range of functions and conditions?

Associated Tools
- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = no [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

External PERFORMANCE subgroup

Definition The amount of useful work accomplished by a system compared to the time and resources used and the responsiveness of the system

Software subfield performance engineering

Aspect of Availability

Associated with reliability, recoverability, internal performance subgroup

Conflicts with understandability, verifiability, interoperability, security, adaptability qualities, repairability qualities

Architectural patterns to avoid layered groupings of modules

Resources

Software Systems Architecture - chapter 25

Software Architecture in Practice -- chapter 8

"Quality Attributes" Technical Report CMU/SEI-95-TR-021 Chapter 3

The Practical Performance Analyst

Automated Source Code Performance Efficiency Measure OMG/CISQ

Other properties

b. Type = composite external behavior quality

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = yes [yes, maybe, no]

Notes

List of performance analysis tools:
http://en.wikipedia.org/wiki/List_of_performance_analysis_tools

Responsiveness

Definition The total time it takes to respond to a request for service.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Response times during final testing on production platform e.g. beta test

Operational Measures

- Initial and final response times under heavy load and light load
  Initial response time is the time to receive the first response to a service request.
  Final response time is the time to receive the final response to a service request.
- Average response time

Aspect of external performance subgroup

Supported by – always or sometimes

Associated with throughput, capacity, efficiency

Conflicts with (see performance subgroup)

Threats supplementary processing e.g. decryption

Mitigations

Other achievement tactics

- Increase capacity
- Concurrent processing
- Reduced latency
- Optimal data organization and management
- Caching stable, non-sensitive data
- Compile-time linking
- Faster algorithms
- Faster communication

Verification tactics

- Comprehensive testing
- Measurement of response times under heavy and light loads

Elicitation Questions

- Where does the system spend most of its time?
- What options exist to decrease response time?
- How will response time measures be logged?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Requirement Patterns -- chapter 9.1

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Throughput

Definition Number of requests appropriately handled in a time interval

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Transaction rate during final testing on production platform e.g. beta test

Operational Measures

- Transaction rate under heavy load and light load
- Average transaction rate

Aspect of external performance subgroup

Supported by – always or sometimes

Associated with responsiveness, capacity, efficiency

Conflicts with (see performance subgroup)

Threats Unnecessary serialization of "long" processes, inadequate resources

Mitigations Analyze design for opportunities to overlap "long" processes

Additional achievement tactics

- Increase capacity
- Asynchronous processing
- Reduced latency
- Optimal data organization and management
- Caching stable, non-sensitive data
- Faster algorithms
- Faster communication
- More bandwidth
- Replicate processing
- Load balancing

Verification tactics

- Comprehensive testing of throughput under a variety of loads
- Measure and track transactions processed per unit of time

Elicitation Questions

- What is the estimated transaction profile (frequency and duration) for the system?
- How will the actual profile be logged?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Requirement Patterns -- chapter 9.2

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

External Durability group

These goals extend the useful life of a system in the face of inevitable change.

Internationalizability

Definition Degree to which a system adapts itself to a specific region or language without engineering changes

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of failures during system adaptation during initial acceptance testing

Operational Measures
Percentage of successful system adaptations

Aspect of adaptability, ease of use, ease of configuring and operating

Supported by – always or sometimes

Associated with adaptability subgroup, scalability

Conflicts with

Threats system user messages e.g. error message, in one language

Mitigations centralize system messages

Other achievement tactics

- centralize message handling
- make sure all user messages are instantiated to locale-specific messages
- determine date and time formats as well as currency by locale
- use character sets that cover all languages for which data will be gathered and stored
- design and coding standards that support internationalization

Verification tactics

- Review compliance with supporting standards
- Review locale selection and instantiation code
- Run a thorough test suite through each selectable locale

Elicitation Questions

- Which languages and character sets must be supported?
- Which date, time, and currency formats must be supported?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

"Internationalization and Localization"
en.wikipedia.org/wiki/Internationalization_and_localization

Software Systems Architecture - chapter 28

Software Requirement Patterns -- chapter 10.5

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external durability quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

MIXED quality attributes

This model organizes quality attributes into 3 main groups.

Mixed qualities are those that can be partially experienced by users. Some aspects can only be directly experienced by developers.

This means these attributes can only be partially verified by testing.

Mixed Behavior group

These goals refer to qualities of system behavior that require verification tactics beyond testing

Software trustworthiness

Definition Software trustworthiness means:

1. software does what it should when it should most of the time (dependability)
2. software doesn't do what it shouldn't when it shouldn't most of the time (safety)
3. all software failures are at most annoying (safety)

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Number of design or code defects detected during review and analysis
- Number of all test failures
- Number of dangerous test failures

Operational Measures

- Mean time to dependability failure
- Mean time to safety failure

Aspect of

Supported by – always or sometimes

Safety, Dependability

Associated with

Conflicts with

Threats Human error

Mitigations

- Disciplined development process
- Extensive verification

Additional achievement tactics

Verification tactics

Comprehensive and continual verification

Elicitation Questions

Associated Tools

- Measurement
- Achievement
- Verification

Resources

DO-178C - Software Considerations in Airborne Systems and Equipment Certification

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Safety

Definition The ability of a system to do little or no harm to valuable assets

Software subfield safety engineering

Assumptions/Rationale

1. Safety is a fragile quality because it depends on many other qualities and the accurate identification of safety hazards.
2. Ultra-safety requirements (e.g. MISRA SIL 4) should be defined by a set of required achievement constraints.

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Ratio of hazards added during HA technical review to hazard count after HA technical review
- Ratio of safeguard defects identified during a technical review to number of safeguards
- Ratio of safeguard defects found during testing to number of safeguards

Operational Measures

- Time since last "dangerous" failure or defect
- Number of "dangerous" failures or defects detected per time interval
- Greatest harm from a harmful event
- Shortest harmful event free duration
- Longest harmful event free duration
- Expected length of harmful event free duration
- Expected rate of harmful events
- Ratio of actual loss to acceptable loss in a duration
- Estimated residual risk

Aspect of software trustworthiness

Supported by – always or sometimes dependability, **ease of learning**

Note: While safety-critical functionality may be supported by these qualities, at the same time they may conflict with non-safety-critical functionality. For example, availability supports dependability, but it may cause non-safety-critical functionality to be sacrificed so the system can continue to operate in a safe, but degraded mode.

Conflicts with efficiency, interoperability, adaptability qualities

Threats [identify using hazard analysis - this is an aggregate quality that includes one quality for each specific hazard e.g., safe *from death by electrocution* (the hazard)]

Mitigations [identify after identifying hazards]

Other achievement tactics

- identify valuable assets and hazards
- identify safety-critical and safety-related functions and constraints needed for safety
  e.g. "The Fire Detection System shall detect smoke above X ppm within 5 seconds."
- isolate and protect safety-critical functions
- guard safety-critical functions with explicit conditions i.e. never with defaults such as "otherwise"
- identify safety-critical users
- eliminate or mitigate hazards i.e. identify appropriate control actions
- effectively execute control actions and receive accurate and sufficient feedback
- alert users to dangerous actions with rotating warning messages

- precede each dangerous action with a delay so user can change their mind and cancel
- limit complexity
- design interfaces that prevent and detect user errors
- use warning labels and messages when appropriate
- use specified tactics for higher-level SILs

Verification tactics

- review hazards and mitigations for completeness and effectiveness during a safety audit
- thoroughly test each safeguard
- measure and track time since last "dangerous" failure or defect and number of "dangerous" failures or defects,
- verify all supporting qualities
- use self-checks
- monitor for misbehavior e.g., unanticipated acceleration
- monitor system state to make sure safety-critical and safety-related functions are active

Elicitation Questions

- What valuable assets are at risk?
- Which functions are safety-critical or safety-related?
- Who/What can perform these safety-critical or safety-related functions and under what conditions?
- What harm can the system or its actors possibly do?
- What can mitigate these hazards?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

"Quality Attributes" Technical Report CMU/SEI-95-TR-021 Chapter 6

Engineering a Safer World

Software Safety Primer

MISRA Report 2 on Integrity

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Dependability

Definition Likelihood that a system will deliver services with appropriate performance when necessary only to authorized recipients

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Number of undependable events during acceptance testing

Operational Measures

- Average time between undependable events, e.g. security breeches, failures, or poor performances
- Rate of occurrence of undependable events

Aspect of safety, portability, component reusability

Supported by – always or sometimes data trustworthiness, availability, **ease of managing**, **error resistance**

Associated with

Conflicts with interoperability, efficiency

Threats difficulty of problem detectability and effective responses

Mitigations

Verification tactics

- Verify all supporting qualities and standards compliance
- Monitor availability, data trustworthiness, performance qualities, and security

Elicitation Questions

- What problems can interfere with dependability?
- What is acceptable dependability?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Fundamentals of Dependable Computing for Software Engineers

"Quality Attributes" Technical Report CMU/SEI-95-TR-021 Chapter 4

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Data trustworthiness

Definition The degree to which data has the following characteristics:

Data quality

- Accuracy
- Precision
- Timeliness
- Consistency
- Legality

Data task-adequacy

- Granularity
- Sufficiency
- Necessity

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Number of data quality or task-adequacy defects detected during data review and analysis
- Number of test failures due to data defects

Operational Measures

Percentage and severity of production failures due to data defects

Aspect of dependability

Supported by – always or sometimes

Trustworthiness of sources and processes, **security**

Associated with availability

Conflicts with

Threats Untrustworthy sources or processes

Mitigations Verify all sources and processes

Additional achievement tactics

Continual detection of unauthorized value changes

Verification tactics

- use a comprehensive data analyzer
- technically review data files
- reverify periodically

Elicitation Questions

- What are the necessary relationships between data elements?
- How will data correctness and consistency be checked and how often?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

The Quest for Software Requirements -- section 5.4

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Sources trustworthiness

Definition The degree to which data sources are trustworthy

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Number of corrupt source data defects detected during review and analysis that are uncorrectable
- Number of missing values detected by analyzer
- Number of data standard compliance defects
- Number of data transformation failures
- Number of test failures due to corrupt source data

Operational Measures

Percentage and severity of production failures due to source-corrupt data defects

Aspect of data trustworthiness

Supported by – always or sometimes

Associated with trustworthiness of data processes

Conflicts with

Threats

Mitigations

Additional achievement tactics

Verification tactics

- Comprehensive verification of each data process
- Comprehensive reverification of any changes to data processes or data structures

Elicitation Questions

How will source trustworthiness be verified and how often?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = none [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Processes trustworthiness

Definition The degree to which data create, retrieve, update, and delete processes are trustworthy

Assumptions/Rationale

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Number of data process defects detected during process review and analysis
- Number of data process test failures

Operational Measures

Percentage and severity of production failures due to data process defects

Aspect of data trustworthiness

Supported by – always or sometimes

Associated with trustworthiness of data sources

Conflicts with

Threats

Mitigations

Additional achievement tactics

Verification tactics

- Comprehensive verification of each data process
- Comprehensive reverification of any changes to data processes or data structures

Elicitation Questions

How will data quality and task-adequacy be verified and how often?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = local [hom cc, het cc, universal cc, local, none]

f. Architecture-relevant = maybe [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Availability

Definition System's ability to carry out its tasks, when needed even under abnormal conditions

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Number and duration of unavailability events during acceptance testing
- Average and longest time to recover during acceptance testing

Operational Measures

- Average and longest duration of outages
- Average and shortest duration between outages
- Outage rate
- Ratio of availability to scheduled availability in a time period

Aspect of dependability

Supported by – always or sometimes security, robustness, **external behavior group**, **scalability** and **repairability subgroup**

Associated with data trustworthiness

Conflicts with

Threats

- Denial of service attacks
- Inadequate resources
- Network failures
- Unnecessary dependencies
- Misuse

Mitigations

Other achievement tactics

Understand

- Sources of unavailability
- Locations of access
- Usage profile
- Misuse cases
- Failure impact
- Availability of interfacing apps

Defect and failure prevention and mitigation

- Redundancy (multiple platforms, multiple storage arrays)
- Voting
- Transaction protocols
- Exception prevention
- Efficient resource usage (load balancing)
- Increase competence set
- Temporary removal from service

Failure detection

- ping/echo
- health monitor
- heartbeat
- time stamping
- sanity checking

Failure recovery

- Exception handling
- Rollback
- Dynamic upgrade
- Retry
- Degraded operation
- Reassigning
- Reintroduction

Verification tactics

- verify all supporting qualities
- review sufficiency and necessity of support tactics
- test all prevention, mitigation, detection, and recovery tactics
- measure and track average time to unavailability and average time to recovery

Elicitation Questions

- Which system aspects threaten availability?
- Are their useful states of semi-availability?
- What is acceptable availability?
- How will availability status be communicated?
- How can the duration and frequency of maintenance be minimized?
- How can the duration and frequency of other causes of unavailability be minimized?
- How can the duration of failure detection and recovery be minimized?
- When can advanced warning, estimated duration, and alternative actions be provided?
- When should data integrity be checked following an unavailability event?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Systems Architecture - chapter 26

Software Architecture in Practice -- chapter 5

The Quest for Software Requirements -- section 5.2

Software Requirement Patterns - subsection 9.5

Blueprints for High Availability: Designing Resilient Distributed Systems

Realizing and Refining Architectural Tactics: Availability

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Security

Definition System's ability to prevent, withstand, and recover from denial of service, privacy, and resource access attacks.

Software subfield security engineering

Operational measures

Frequency of security patches

Aspect of availability

Supported by - sometimes or always privacy, resource security, **reliability**

Associated with robustness, **external behavior group**

Conflicts with ease of use, access in different situations, management, and auditing, along with interoperability, performance qualities, extensibility, repairability qualities

Threats [identify using hazard analysis - this is an aggregate quality that includes one quality for each specified hazard e.g., safe *from death by electrocution* (the hazard)]

Mitigations [identify after identifying hazards]

Other achievement tactics

- Develop or follow security policies
- Design security into the system architecture
  - distrustful decomposition
  - privilege separation
  - secure access layer
- Security mechanisms should be pervasive, simple, scalable, and easy to manage
- Security solutions should include mechanisms to configure and monitor systems for regulatory compliance
- Identify threat sources, including unintentional insider threats
- Identify system vulnerabilities
- Develop attack trees and misuse cases
- Require secure coding standards

Architectural patterns to avoid pipes and filters, blackboard

Verification tactics

- Audit security
- Use self-checks

Resources

Software Systems Architecture - chapter 24

Software Architecture in Practice -- chapter 9

The Quest for Software Requirements -- section 5.1

Software Requirement Patterns -- chapter 11

"Quality Attributes" Technical Report CMU/SEI-95-TR-021 Chapter 5

The Art of Software Security Assessment: Identifying and Preventing Software Vulnerabilities

The Hacker Playbook: Practical Guide To Penetration Testing

OWASP Application Security Verification Standard (2014)

OWASP Secure Coding Practices: Quick Reference Guide

2011 CWE/SANS Top 25 Most Dangerous Software Errors --
http://cwe.mitre.org/top25/

Source Code Security Analysis OMG/CISQ

SAFEcode Fundamental Practices for Secure Software Development

Other properties

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = yes [yes, maybe, no]

Notes

Lists of vulnerability scanners and other security tools can be found at:
www.owasp.org/index.php/Category:Vulnerability_Scanning_Tools
www.sectools.org

Security by the numbers:

In 2014 the average time it took to detect a cyber-crime by outsiders was 170 days (i.e. almost 6 months). Attacks involving insiders with access to the network took 259 days (i.e. almost 9 months)! [Bimodal distribution] 69% of the breaches involved an insider

The average time it took a company to recover from a data breach was 45 days.

In 10 percent of the data breaches, company wasn't able to determine the threat actor.

False positives in detection systems is a BIG problem.

Privacy

Definition Systems ability to prevent unauthorized access to data and activity information

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of successful attacks during final acceptance by motivated expert hackers

Operational Measures
Time since last data or activity focused breach

Aspect of security

Supported by – always or sometimes

Associated with resource security

Conflicts with (see security)

Threats

Mitigations

Other achievement tactics

Identify privacy-related functions and constraints
Classify users by role (i.e. privacy-relevant properties)
Define user roles needed for authorized access to each privacy-related function
Develop a threat profile with consequences and risks
Data

- develop an information governance strategy
- encrypt data including backups
- consider implications of legal data privacy requirements
- consider need for document, email and fax privacy
- train staff about privacy policies
- monitor data access patterns
- monitor data trustworthiness

Activity

- Scrub traces of activity

Define privacy policies

Verification tactics

- Verify supporting qualities
- Review privacy policies
- Monitor compliance with privacy policies
- Review code to verify encryption of all confidential data and no activity tracing
- Test privacy supports
- Penetration test
- Measure and track time since last breach and number of breaches

Elicitation Questions

- Which data must be protected and how?
- How is data export controlled?
- How can activity be traced?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Privacy Engineering

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Resource security

Definition System's ability to control and audit useful access to capabilities and resources. Read-only access to strongly encrypted data would not be “useful”.

Software subfield security engineering

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of successful attacks during final acceptance by motivated expert hackers

Operational Measures

- Frequency of breaches
- Severity of breaches
- Frequency of virus infections

Aspect of security

Supported by – always or sometimes reliability

Associated with privacy

Conflicts with (see security)

Threats hackers, malicious employees, malware, phishing, denial of service, botnets, man-in-the-middle attacks

Mitigations

Control access

- Authentication
  a. Consider password strength
  b. Consider segmented sign-ons e.g. 2 people each having a part of the sign-on
  c. Consider bio authentication
- Authorization levels
- Network security systems i.e. firewalls
- Secure communication e.g. encryption and digital signatures
- Secure administration

Resist attacks

- Identify trust boundaries
- Identify users
- Authenticate users
- Authorize users, course and fine-grained
- Limit access
- Restrict input and response data
- Encrypt output
- Limit exposure
- Separate entities
- Change defaults

Detect attacks

- Identify security-critical and security-related functions and constraints on those functions needed for security
- Monitor application access patterns
- Detect intrusion
- Detect message delay
- Verify message integrity

React to attacks

- Revoke access
- Lock computer
- Inform users
- Maintain audit trail

Recover from successful attacks

- Verify data trustworthiness
- Restore with revoked accesses

Additional achievement tactics

- Identify essential assets
- Segregate data and functionality at different security levels (e.g. public, confidential, secret)
- Trust assurance must be interoperable to support collaboration
- Access to data should be controlled by security attributes of the data itself
- Data must be appropriately secured when stored, in transit, and in use and implement privacy policies
- Security-relevant actions should be logged, audited, and perhaps monitored
- Resources should be protected at levels appropriate to their value,privacy, and integrity
- Access should have only the privileges required to carry out specified tasks

Verification tactics

- Analyze and verify possible security breaches
- Review and test code for each security mechanism
- Penetration test

Elicitation Questions

- Which security breaches are possible?
- How should authentication and authorization be handled?
- What information must be protected?
- What are the security user classes and their access frequency>
- How should access be controlled?
- Which actions must be controlled, both always and during time periods?
- Are there session time limits?
- What session information is monitored and logged?
- How should malicious input be detected and handled?
- How to protect against SQL injection?
- How to protect against cross-site scripting?
- How are unauthorized access attempts and breaches reported?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Information Security

Fundamental Practices for Secure Software Development (**2nd edition)**

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Robustness

Definition Degree to which a system continues to function properly under a normal and heavy load for an extended period or following a system modification, abnormal condition, detected exception, component failure, or denial of service attack

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

Operational Measures

- Average time between avoidable failures
- Rate of avoidable failures
- Severity of avoidable failures

Aspect of availability

Supported by – always or sometimes survivability, fault tolerance, **reliability**

Associated with security, **external behavior group**

Conflicts with safety, testability, performance qualities

Threats

- Failure to handle exceptions
- Overly narrow input handling
- Inadequate verification of system modifications

Mitigations

- Comprehensive exception handling
- Deal with a broad range of inputs
- Identify invalid inputs
- Comprehensively verify system modifications

Other achievement tactics

- Identification of likely exceptions
- Comprehensive exception handling and recovery
- Thoroughly validate input
- Load balancing
- Logging of abnormal conditions, corrupt data, heavy loads, handler invocations, and response times
- Verification standards requiring thorough verification of modifications

Verification tactics

- Verify supporting qualities
- Review and test exception handlers

- Verify input
- Perform load, stress, endurance, and spike testing
- Measure and track average time between failures
- Use software fault injection during test

Elicitation Questions

- How to redirect load under extreme circumstances?
- How to take the system offline, but still queue pending requests?
- How will the robustness of third party components be verified?
- Who should be notified of handler invocation?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Systems Architecture - chapter 26

The Quest for Software Requirements -- section 5.6

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Fault tolerance

Definition The degree to which a system continues to operate properly in the event of the failure of one or more of its components.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of failure during initial component failure testing

Operational Measures

- Ratio of system failures caused by component failures to all component failure events
- Severity of system failures caused by component failures
- Average downtime

Aspect of robustness

Supported by – always or sometimes

Associated with survivability

Conflicts with safety

Threats Inadequate detection of component failures, abnormal behavior, or corrupt data

Mitigations Self-checking of component results, data trustworthiness, behavior, and timing.

Additional achievement tactics

- Encapsulating external modules or services
- Software redundancy of data and process
- Hardware redundancy
- Exception handling
- Recovery mechanisms
    a. checkpoint/restart
    b. error correcting code
    c. reexecution e.g. resend dropped or corrupted messages
    d. dynamic reconfigurability

Verification tactics

- Technical review and thorough testing of failure detection and handling
- Measure and track average time between failures
- Self-checks of component results and data trustworthiness to restrict failure propagation

Elicitation Questions

- How to design failover support related to different tiers in the system?
- How to decide if there is a need for a geographically separate redundant site to failover to in case of natural disasters such as earthquakes or tornados?
- How to handle unreliable network connections?
- How to detect failures and automatically initiate a failover?
- How to handle failed communications?
- How to handle failed transactions?

Associated Tools

- Measurement
- Achievement
- Verification
  Chaos Monkey

Resources

"An Introduction to Fault-Tolerant Systems**"** Kjetil Nørvag

Patterns for Fault Tolerant Software by Robert Hanmer

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed behavior quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

f. Architecture-relevant = yes [yes, maybe, no]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

Survivability

Definition System's ability to continue providing critical services while preventing and withstanding denial of service attacks

Software subfield security engineering

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement
Percentage of successful attacks during final acceptance by motivated expert hackers

Operational Measures

- Average time between avoidable failures caused by denial of service attacks
- Avoidable failure rate
- Severity of avoidable failures
- Average downtime

Aspect of robustness

Supported by – always or sometimes

Associated with fault tolerance

Conflicts with

Threats denial of service attacks by hackers or malicious employees

Mitigations

Prevent attacks
- Mitigate system platform vulnerabilities

Detect attacks
- Monitor platform configuration
- Monitor and control access request volume and content e.g. requests by same user or same application
- Monitor and control requests for resource-intense functions
- Detect message delay
- Verify message integrity
- Monitor application and data access patterns

React to attacks
- Temporarily revoke access of some users and/or applications
- Temporarily block some resource-intense functions and non-essential services
- Inform users
- Maintain audit trail

Recover from successful attacks

- Restore platform configuration
- Restore system with revoked access and blocked functions

Additional achievement tactics

Verification tactics

- Analyze and verify possible modes of service denial attacks
- Review and test code for each security mechanism

Elicitation Questions

- What are the vulnerabilities of the system platform e.g. loss of power?
- How is the system vulnerable to denial of service attacks?
- What are the worst things that a malicious employee can do?
- Which users are non-essential?
- Which functions are non-essential and resource-intense?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = external behavior quality

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies

[External Behavior group]

Mixed Durability group

These goals extend the useful life of a system in the face of inevitable change.

Scalability

Definition Ease with which a system handles a growing workload.

Assumptions/Rationale

Quality attribute scenarios **[**described in **Software Architecture in Practice**]

Leading Indicators -- provide preoperational evidence of quality goal achievement

- Percentage of failures during initial scalability testing
- Average system effort during each successful scalability test

Operational Measures

Number of failures due to faulty scalability

Aspect of durability

Supported by – always or sometimes **reliability, data trustworthiness**

Associated with

Conflicts with performance qualities

Threats hard-coded constraints, inflexible communication

Mitigations parameterize constraints when useful

Other achievement tactics

- Project growth of users, transactions, applications
- Identify scaling mechanisms on one or several machines
- Design and coding standards that support scalability

Verification tactics

- Review standards compliance
- Audit scalability by analyzing scaling strategy and mechanisms
- Test scaling with large work loads

Elicitation Questions

- What is the likely growth in users, transactions, and applications?
- What are the major choke points and how will they be scaled?

- How will spikes in traffic and load be handled?
- How will supplemental devices, communication channels, and machines be managed?
- How will workload growth and system responses be monitored?

Associated Tools

- Measurement
- Achievement
- Verification

Resources

Software Systems Architecture - chapter 25
The Quest for Software Requirements -- section 6.3
Software Requirement Patterns -- chapter 10.1
On System Scalability
20 Obstacles to Scalability ACM Queue 2013

Risk Factors

a. Developer understanding = [superficial, limited, deep]

b. Cost (implementation, verification, maintenance) = [high, medium, low]

c. Feasibility (technical, cost, understanding) = [low, medium, high]

Other properties

a. Sources/Enterprise goals:

b. Type = mixed durability quality

c. Associated scope = [system, <specific partitions>]

d. Design scope = het cc [hom cc, het cc, universal cc, local, none]

e. **Consensus Priority** = [critical, important, desirable]

States

a. Goal states are < @Incomplete, Complete, Validated, Implemented, Inactive>

Notes

Past Quality Goal Specs

Past Achievement, Monitoring, and Verification Strategies

Current Quality Goal Spec

Current Achievement, Monitoring, and Verification Strategies