Carnegie Mellon
Software Engineering Institute

Software Quality Attributes and Software Architecture Tradeoffs

Mario R. Barbacci
Charles B. Weinstock

Software Engineering Institute
Carnegie Mellon University
Pittsburgh PA 15213

Sponsored by the U.S. Department of Defense

Quality Attribute Taxonomies

Attribute taxonomies are developed and maintained by different communities of experts.

Methods used to achieve quality are attribute specific.

Stakeholders have different quality attribute requirements and some requirements might not be explicit.

Methods for different attributes can conflict or reinforce each other: win-win :) , win-lose :| , lose-lose :(

Quality Attribute Methods

We have a process for exposing stakeholders conflicts.

Experts can do analysis and find risks, sensitivities, and tradeoffs after conflict is identified.

Need cross references for methods to achieve different quality attributes:

	Performance	Dependability	Security
Security Method a	⇩	⇦⇨	⇧⇧⇧
Security Method b	⇧	⇩	⇧
Dependability Method c	⇧	⇧⇧	⇩

Approaches to Quality Attributes

- performance — from the tradition of hard real-time systems and capacity planning
- dependability — from the tradition of ultra-reliable, fault-tolerant systems
- security — from the traditions of the government, banking and academic communities
- usability — from the tradition of human-computer interaction and human factors
- safety — from the tradition of hazard analysis and system safety engineering
- integrability and modifiability — common across communities

Methods in Performance: Analysis

- Methods
  - Synthesis
  - Analysis
    - Queuing theory
      - Average case
      - Aggregate behavior
    - Scheduling theory
      - Utilization bounds
      - Response times
      - Worst case behavior
    - Formal methods
      - Formal specification
      - Formal verification

Methods in Dependability: Fault Tolerance

- Fault tolerance
  - Error processing
    - Detection
    - Diagnosis
    - Recovery
      - Backward recovery
      - Forward recovery
      - Compensation
  - Fault treatment
    - Diagnosis
    - Passivation
    - Reconfiguration

Tradeoffs in Usability: Intentional Deficiency

Efficiency might be sacrificed to avoid errors:

- asking extra questions to make sure the user is certain about a particular action

Learnability might be sacrificed for security:

- not providing help for certain functions e.g., not helping with useful hints for incorrect user IDs or passwords

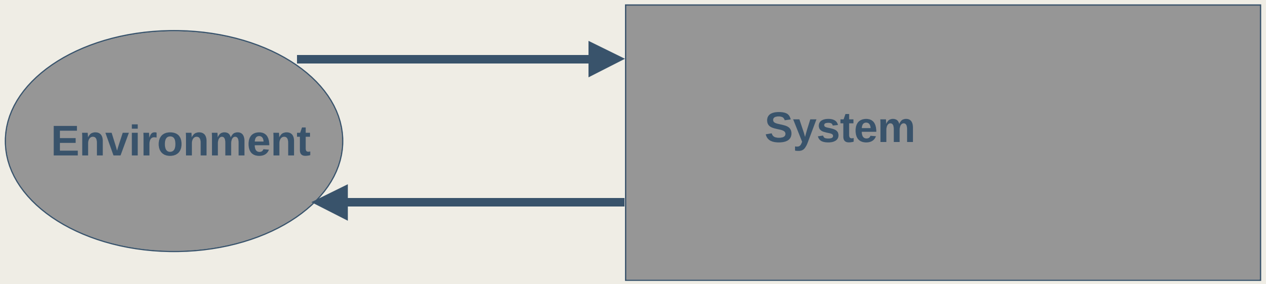
Learnability might be sacrificed by hiding functions from regular users:

- hiding reboot buttons/commands in a museum information system

Example Problem Description

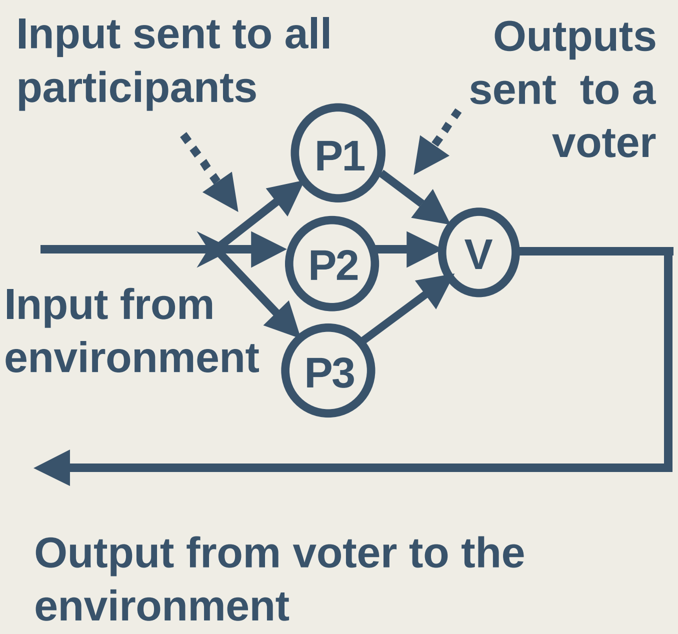
A system processes input data from the environment and in turn sends results back to the environment.

An important requirement could be that system failure rate be less than some minimum reliability requirement.

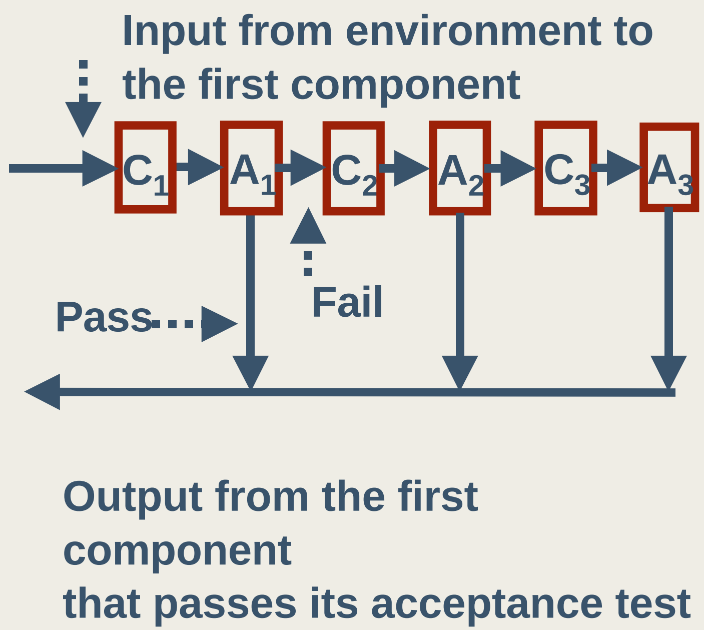


Approaches to Dependability

Triple-Modular Redundancy (TMR)



Recovery Blocks (RB)



Tradeoffs Between Dependability and Performance in TMR

If the components share a processor the latency depends on how many components are working:

- performance calculations should be based on worst-case i.e., all components are working
- voter can decide when to send output to constrain latency variability

Tradeoffs Between Dependability and Performance in RB

Latency variability is greater:

- components perform different algorithms (execution time varies)
- acceptance tests are component-dependent (execution time varies)
- when a component fails, there is a roll-back to a safe state before the next alternative is tried (previous execution time is wasted + time to restore state)

Additional Tradeoffs Between Dependability and Performance

TMR and RB repair operations also affect performance:

- running diagnostics
- restarting a process
- rebooting a processor

Carnegie Mellon
Software Engineering Institute

TMR Dependability Analysis

The reliability of a TMR system is:

$$R_{TMR}(t) = 3e^{-2\lambda t} - 2e^{-3\lambda t}$$

The Mean-Time-To-Failure of a TMR system without repairs is:

$$MTTF_{TMR} = \left(\int_0^{\infty} 3e^{-2\lambda t} dt - \int_0^{\infty} 2e^{-3\lambda t} dt\right) = \frac{3}{2\lambda} - \frac{2}{3\lambda} = \frac{5}{6\lambda}$$

The MTTF of a TMR system with repairs is:

$$MTTF_{TMR} = \frac{5}{6\lambda} + \frac{\mu}{6\lambda^2}$$

λ and μ are the failure and repair rates, respectively.

Carnegie Mellon
Software Engineering Institute

RB Dependability Analysis

For a 3-component recovery block system :

$$R_{RB}(t) = e^{-\lambda t} \sum_{i=0}^{2} C^i (1 - e^{-\lambda t})^i \qquad MTTF_{RB} = \frac{1}{\lambda}\left(1 + \frac{c}{2} + \frac{c^2}{3}\right)$$

Where c is the acceptance test coverage.

- If c=1 (acceptance test never fails to detect errors):

$$MTTF_{RB} = \frac{11}{6\lambda}$$

- If c=0.5 (acceptance test fail half the time):

$$MTTF_{RB} = \frac{4}{3\lambda}$$

Dependability Sensitivity Points

If a component has a failure rate of one per 1000 hrs. and a repair rate of one per 10 hours (λ=0.001, μ=0.1):

The Mean Time To Failure for the alternatives are:

- TMR without repair = $5/(6\lambda)$ = 833 hours
- Non-redundant component = $1/\lambda$ = 1,000 hours
- RB with 50% coverage = $4/(3\mu)$ = 1,333 hours
- RB with 100% coverage = $11/(6\mu)$ = 1,833 hours
- TMR with repair = $5/(6\lambda) + \mu/(6\lambda^2)$ = 17,500 hours

The choice of "voting" technique (i.e., TMR or RB) constitute a sensitivity point for dependability.

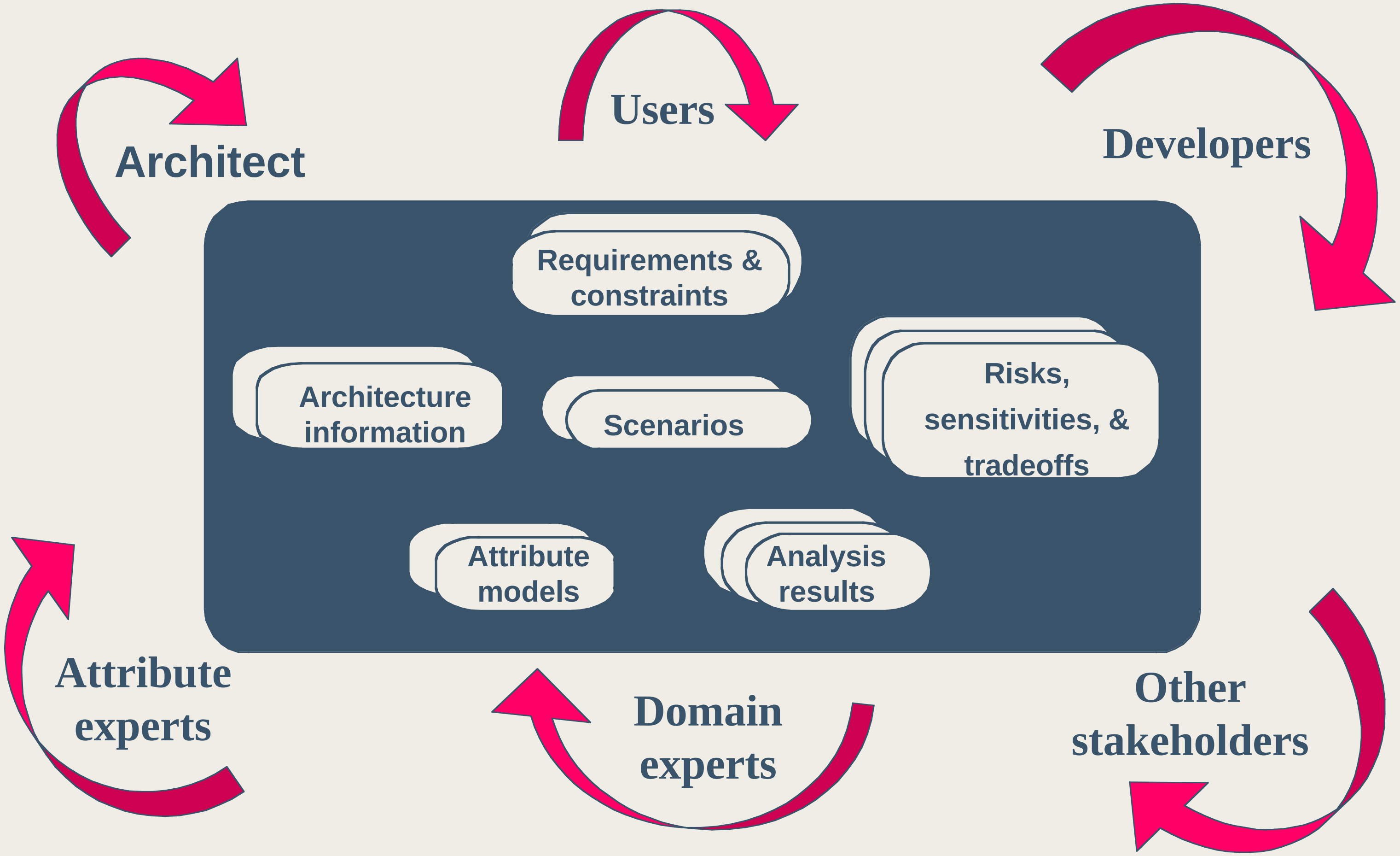
Risks in TMR and RB

Depending on the TMR approach to repairs, different risks emerge:

- a TMR system without repair is less dependable that just a single component!
- a TMR system with very lengthy repairs could be just as undependable

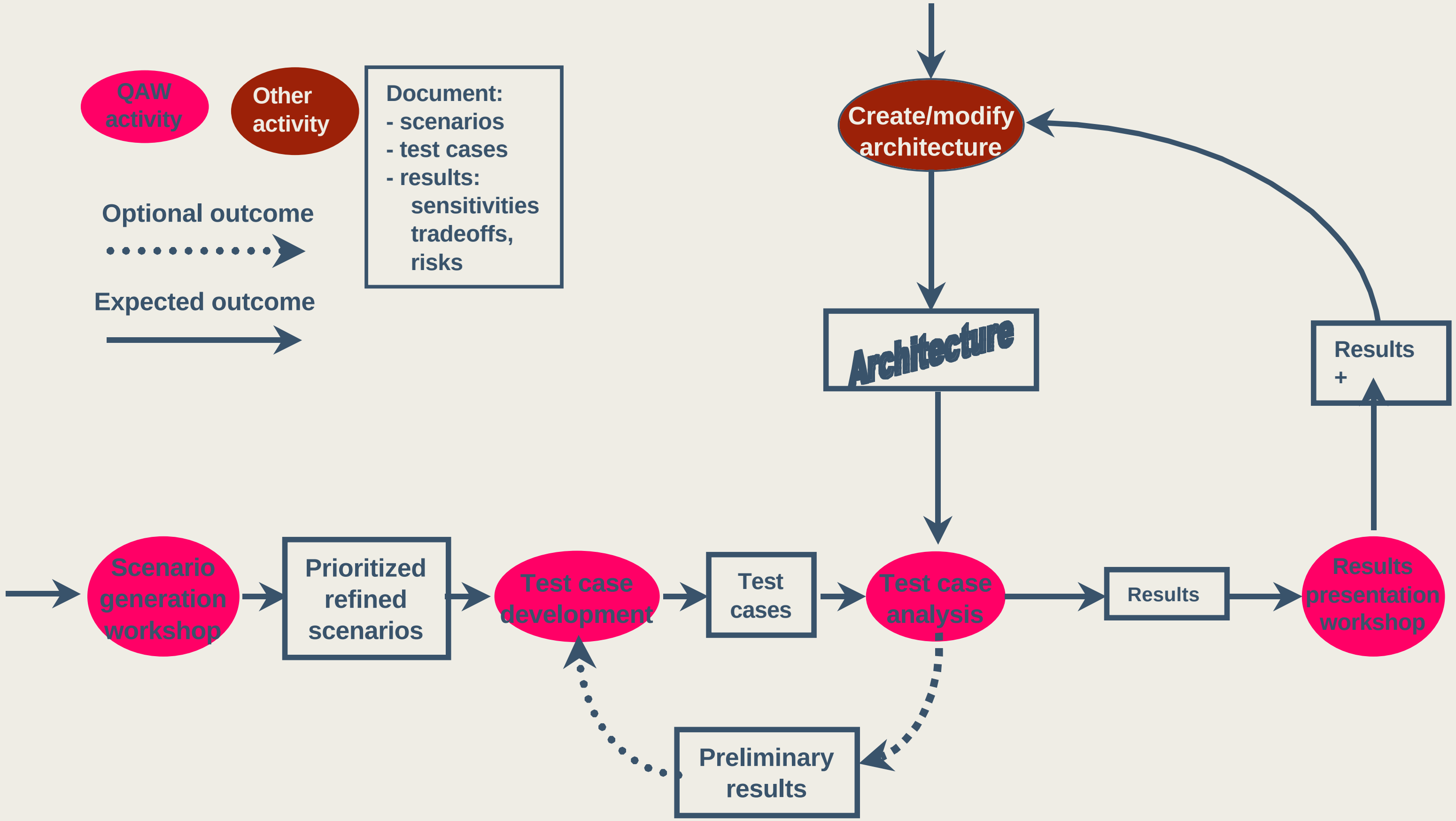
The RB time to execute components, tests, and recoveries varies and could present a performance risk if the deadlines are tight.

Interactions Between Stakeholders



Carnegie Mellon
Software Engineering Institute

The QAW Process



Need for Pre-identified Tradeoffs and Validation Experiments

Conducting “analysis” from first principles (QAW or otherwise) is inefficient.

Collections of pre-identified tradeoffs and sensitivities would help to guide analysis:

- **requires cooperation between domain experts**
- **need experiments to validate tradeoffs hypotheses**

	Performance	Dependability	Security
Security Method a	⇩	⇦⇨	⇧⇧⇧
Security Method b	⇧	⇩	⇧
Dependability Method c	⇧	⇧⇧	⇩

Software Quality Attributes

There are alternative (and somewhat equivalent) lists of quality attributes. For example:

IEEE Std. 1061	ISO Std. 9126	MITRE Guide to Total Software Quality Control	
Efficiency	Functionality	Efficiency	Integrity
Functionality	Reliability	Reliability	Survivability
Maintainability	Usability	Usability	Correctness
Portability	Efficiency	Maintainability	Verifiability
Reliability	Maintainability	Expandability	Flexibility
Usability	Portability	Interoperability	Portability
		Reusability	

Quality Factors and Sub-factors

IEEE Std. 1061 subfactors:

Efficiency
- Time economy
- Resource economy

Functionality
- Completeness
- Correctness
- Security
- Compatibility
- Interoperability

Maintainability
- Correctability
- Expandability
- Testability

Portability
- Hardware independence
- Software independence
- Installability
- Reusability

Reliability
- Non-deficiency
- Error tolerance
- Availability

Usability
- Understandability
- Ease of learning
- Operability
- Comunicativeness

Quality Factors and Sub-factors

IEEE Std. 9126 subcharacteristics:

Functionality
- Suitability
- Accurateness
- Interoperability
- Compliance
- Security

Efficiency
- Time behavior
- Resource behavior

Maintainability
- Analyzability
- Changeability
- Stability
- Testability

Reliability
- Maturity
- Fault tolerance
- Recoverability

Usability
- Understandability
- Learnability
- Operability

Portability
- Adaptability
- Installability
- Conformance
- Replaceability

A Typical Attribute Taxonomy

<Attribute name>

- **Concerns**
  - *<concern>*
  - *<concern>*
- **Factors**
  - *Internal/External*
  - *Behavior/Structure*
  - *Policies/Mechanisms*
  - *Causes/Effects*
- **Methods**
  - *Analysis/Synthesis*
  - *Procedures/Training*
  - *Development/Execution*

Performance Taxonomy

- Performance
  - Concerns
    - Latency
    - Throughput
    - Capacity
    - Modes
  - Factors
    - Environment
    - System
  - Methods
    - Synthesis
    - Analysis

Carnegie Mellon
Software Engineering Institute

Methods in Performance

- **Methods**
  - **Synthesis** — *normal software development steps with explicit attention to performance*
  - **Analysis** — *techniques used to evaluate system performance*

Methods in Dependability: Fault Removal

- Fault removal
  - Verification
    - Static
      - Formal verification
      - Code inspection
    - Dynamic
      - Symbolic execution
      - Testing
        - Conformance vs. Fault-findings
        - Functional vs. Structural
        - Fault-based vs. Criteria based
  - Diagnosis
  - Correction

Methods in Dependability: Fault Forecasting

- Fault forecasting
  - Qualitative
    - Identify failure modes
    - Classify failure modes
    - Order failure modes
    - Identify undesirable event combinations
  - Quantitative
    - Testing
    - Modeling
      - Stable reliability
      - Reliability growth

Dependability Taxonomy

- **Dependability**
  - **Concerns (attributes)**
    - **Availability**
    - **Reliability**
    - ***Safety***
    - ***Confidentiality***
    - ***Integrity***
    - ***Maintainability***
  - **Factors (impairments)**
    - **Faults**
    - **Errors**
    - **Failures**
  - **Methods (means)**
    - **Fault prevention**
    - **Fault removal**
    - **Fault forecasting**
    - **Fault tolerance**

Security Taxonomy

- Security
  - Concerns
    - Confidentiality
    - Integrity
    - Availability
  - Factors
    - Interface
    - Internal
  - Methods
    - Synthesis
    - Analysis

Methods in Security

- Methods
  - Synthesis
    - Process models
    - Security models
    - Secure protocols
  - Analysis
    - Formal methods
    - Penetration analysis
    - Covert channel analysis