CDT413: Advanced Software Engineering

Software Architecture Design

- Software Quality Attributes
- Software Architecture Tactics
- Architectural Patterns and Styles

Software Quality Attributes

- System Quality Attributes
  - Availability
  - Modifiability
  - Performance
  - Security
  - Testability
  - Usability
- Business Qualities
  - Time to market
  - Cost and benefit
  - Projected lifetime
  - ...
  - Architecture Qualities
    - Conceptual integrity
    - ...

Availability

- The probability that the system will be operational when needed
- Depends on the probability of failure as well as the time it takes to repair the system after failure
  - mean time to failure / (mean time to failure + mean time to repair)
- Failures are caused by faults, but faults need not lead to failures
  - E.g. the system can be designed to be fault tolerant

Modifiability

- The ease with which the system can be changed
- Modifiability with respect to change X
  - 1 / (The cost of making change X)

Performance

- The timing and efficiency of the system
- Latency
- Throughput
- Jitter (= variation in latency)
- Miss rate and data loss
Security

- The system’s ability to resist unauthorized usage while still providing services to legitimate users
- Depends on the ability to survive different types of attacks
  - Unauthorized attempt to access or change data
  - Denial of service attack

Security

- Nonrepudiation
  - A transaction cannot be denied by any of the parties to it after it has taken place
- Confidentiality
  - Data or services are protected against unauthorized access
- Integrity
  - Data or services are being delivered as intended

Assurance

- The parties to a transaction are who they claim to be

Availability

- The system will be available for legitimate use

Auditing

- The system tracks activities sufficient to reconstruct them

Testability

- The ease with which the system can be tested
- Assuming a system has one fault
  - Testability = Probability of failure at the next test execution
- For a system to be testable, it must be possible to
  - Control each component’s internal state and input
  - Observe the component’s output

Usability

- How easy it is for the user to accomplish a desired task
  - Learning system features
  - Using the system efficiently
- What kind of user support the system provides
  - Minimizing the impact of errors
  - Adopting to user needs
  - Increasing confidence and satisfaction

ISO 9126 Software Quality Model

- Functionality
  - Suitability
  - Accuracy
  - Interoperability
  - Compliance
  - Security
- Reliability
  - Maturity
  - Recoverability
  - Fault Tolerance
- Efficiency
  - Time Behavior
  - Resource Behavior
- Maintainability
  - Stability
  - Analyzability
  - Changeability
  - Testability
- Portability
  - Installability
  - Replaceability
  - Adaptability
- Usability
  - Learnability
  - Understandability
  - Operability

Quality Requirements

- A system’s quality requirements must be stated operationally
  - “The system must have high availability” is not a meaningful requirement
- “If the system receives an unanticipated message during normal operation it must inform the operator and continue to operate with no downtime”
Quality Attribute Scenarios

- Source of stimulus
- Stimulus
- Environment
- Artifact
- Response
- Response measure

Example Performance Scenario

- Web-based financial system
- “Users initiate 1000 transactions per minute stochastically under normal operations, and these transactions are processed with an average latency of two seconds.”

Example Performance Scenario

- Source of stimulus: Users
- Stimulus: Initiate transactions
- Environment: Normal operation
- Artifact: The system
- Response: Transactions are processed
- Response measure: Average latency of 2 s

Example Modifiability Scenario

- A system with a graphical user interface
- “A developer wishes to change the user interface to make a screen’s background color blue. This change will be made to the code at design time. It will take less than three hours to make and test the change and no side effect changes will occur in the behavior.”

Example Modifiability Scenario

- Source of stimulus: Developer
- Stimulus: Wishes to change the UI
- Environment: At design time
- Artifact: Source code
- Response: Modification is made with no side effects
- Response measure: In less than three hours

Modifiability Scenarios

- Source: End user, developer, system administrator
- Stimulus: Wishes to add/delete/modify/vary functionality, quality attribute, capacity
- Artifact: System user interface, platform, environment; system that interoperates with target system
Modifiability Scenarios

- Environment: At runtime, compile time, build time, design time
- Response: Locate places in architecture to be modified; makes modification; tests modification; deploys modification
- Response measure: Cost in terms of number of elements affected, effort, money; extent to which this affects other functions or quality attributes

Performance Scenarios

- Source: One of a number of independent sources, possibly from within the system
- Stimulus: Periodic events arrive; sporadic events arrive; stochastic events arrive
- Artifact: System

Performance Scenarios

- Environment: Normal mode; overload mode
- Response: Processes stimuli; changes level of service
- Response measure: Latency, deadline, throughput, jitter, miss rate, data loss

Software Architecture Tactics

- Design decisions with known ramifications for a system's quality attributes
- A collection of tactics is called an architectural strategy
- Architectural patterns and styles incorporate several tactics

Modifiability Tactics

- Aims to control the time and cost to implement, test, and deploy changes
- Organized in three sets
  - Localize modifications
  - Prevent ripple effects
  - Defer binding time

Modifiability Tactics: Localize Modifications

- Maintain semantic coherence
- Anticipate expected changes
- Generalize the module
- Limit possible options
Modifiability Tactics: Defer Binding Time

- Defer binding between components to loadtime or runtime
- Reduces the time required for deployment of changes
- Allows changes to be made by non-developers

Modifiability Tactics: Prevent Ripple Effects

- Hide information
- Maintain existing interfaces
  - Add new interfaces instead of changing the existing
  - Add adapter that wraps a changed module
  - Provide a stub for a deleted module
- Restrict communication paths
- Use an intermediary

Performance Tactics

- Aims to ensure that the system can respond to events within certain time constraints.
- Organized in three sets
  - Resource demand
  - Resource management
  - Resource arbitration (scheduling policies)

Performance Tactics: Resource Demand

- Increase computational efficiency
- Reduce computational overhead
- Manage event rate
- Control frequency of sampling
- Bound execution times
- Bound queue sizes

Architectural Styles and Patterns

- According to Bass et al. (2003), an architectural pattern is determined by
  - A set of element types
  - A topological layout of the elements indicating their interrelationships
  - A set of semantic constraints
  - A set of interaction mechanisms that determine how the elements coordinate through the allowed topology
- This is really the same as architectural styles according to Garlan and Shaw (1994)
- Incorporate several architectural tactics