Quality Attribute Scenarios and Architectural Tactics

15-313: Foundations of Software Engineering

Jonathan Aldrich
Quality Attribute Scenarios [BCK03]

- **Stimulus**: A condition that affects a system
- **Source of stimulus**: The entity that generates the stimulus
- **Environment**: The condition under which the stimulus occurred
- **Artifact**: The artifact that was stimulated
- **Response**: The activity that must result from the stimulus
- **Response measure**: The measure by which the response is evaluated
Availability

• Is the system able to provide services to users?
  • Often measured as a probability

• Issues
  • faults → failures
    • can intervene to avoid this
  • fault/failure detection
  • failure notification
  • failure recovery
    • how long to repair?
## Availability Scenarios

<table>
<thead>
<tr>
<th>Scenario Portion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>Internal vs. external to system</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Crash, omission, timing, incorrect response</td>
</tr>
<tr>
<td>Artifact</td>
<td>System’s processors, communication channels, persistent storage, processes</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal operation; degraded mode</td>
</tr>
<tr>
<td>Response</td>
<td>Log the failure, notify users/operators, disable source of failure, be unavailable, continue (normal or degraded mode)</td>
</tr>
<tr>
<td>Measure</td>
<td>Time interval available, availability %, repair time, unavailability time interval</td>
</tr>
</tbody>
</table>
How to Increase Availability?

- **Architectural Tactic**
  - strategy for promoting quality attribute
  - independent of implementation technology
  - independent of exact architectural structure

- **Three kinds of Availability Tactic**
  - Fault detection
  - Fault recovery
  - Fault prevention
Availability Tactics: Fault Detection

- ping/echo
  - ping another component
  - expect an echo before a timeout

- heartbeat
  - expect periodic message

- exceptions
  - detect generated exception
Availability Tactics: Fault Recovery

- **voting**
  - multiple components produce answer
  - give client the answer with the most votes
  - most useful for hardware failures
    - buggy software will fail in the same way
      - occurs even if built by different teams!

- **active replicas**
  - all replicas respond to all messages

- **passive replicas**
  - passive replicas periodically updated with current state
  - requires limited replay

- **spare**
  - must boot, load checkpoint, and replay recent messages

- **checkpoint/rollback**
  - allows undoing operations after a failure
Availability Tactics: Fault Prevention

• remove from service
  • e.g. reboot a component that’s getting low on memory
    • surprisingly effective for OS drivers

• transactions
  • avoids failures/inconsistencies when part of an operation fails

• process monitor
  • detect fault in running process, then restart and reinitialize before errors propagate
What is the cost of these tactics?
Modifiability

- What is the cost of change?

- Issues
  - What is changing?
    - functions, platforms, hardware, protocols…
    - quality attributes
  - Who changes it?
  - When is it changing?
# Modifiability Scenarios

<table>
<thead>
<tr>
<th>Scenario Portion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>End-user, developer, system-administrator</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Add/delete/modify functionality or quality attributes</td>
</tr>
<tr>
<td>Artifact</td>
<td>System user interface, platform, environment</td>
</tr>
<tr>
<td>Environment</td>
<td>At runtime, compile time, build time, design-time</td>
</tr>
<tr>
<td>Response</td>
<td>Locate places in architecture for modifying, modify, test modification, deploys modification</td>
</tr>
<tr>
<td>Measure</td>
<td>Cost in effort, money, time, extent affects other system functions or qualities</td>
</tr>
</tbody>
</table>
Modifiability Tactics

• Localize modifications
  • Modifications affect the requirements of as few modules as possible

• Prevent ripple effects
  • Limit effect of changing one module on other modules

• Defer binding time
  • Allow modifications late in process at low cost
Modifiability Tactics: Localization

- Maintain semantic coherence
  - group responsibilities that are likely to change together in the same module

- Hide information based on anticipated changes
  - organize architecture to minimize number of modules affected by *specific* changes

- Generalize the modules
  - the more general the module’s interface, the more likely changes in requirements won’t require changing the interface or the implementation
    - BUT – generality creates complexity and cost, so only use it if you need it

- Limit possible options
  - Restrict the set of possible changes so that you can plan better for the supported changes
Modifiability Tactics: Prevent Ripples

- **Key issue: Notion of interfaces**
  - types and signatures, semantics, sequences
  - identity, location, existence
  - quality of service, resource use

- **Tactics**
  - Hide information (like the above) within a module
  - Maintain interfaces
    - Extend in backwards compatible way, add an adapter
  - Restrict communication paths
    - Fewer connections along which ripples can propagate
  - Add an intermediary
    - Convert data types, replace interfaces with a proxy
    - Hide identity using broker, location using name server
    - Resource manager hides resource behavior
Modifiability Tactics: Deferring Binding

• Event registration
  • Plug-and-play connection between components
  • Cost in understandability

• Polymorphism
  • Late binding of method calls
  • Cost in performance

• Configuration files
  • Bind at deployment
  • Cost in complexity
Performance

• How long does it take the system to respond to an event?
  • Generalizes to throughput, etc.

• Issues
  • Sources of events
    • end users, interrupts, timers, messages
  • Arrival patterns
    • periodic, sporadic, stochastic
  • Response criteria
## Performance Scenarios

<table>
<thead>
<tr>
<th>Scenario Portion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>A number of sources both external and internal</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Periodic events, sporadic events, stochastic events</td>
</tr>
<tr>
<td>Artifact</td>
<td>System or component</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal mode; overload mode</td>
</tr>
<tr>
<td>Response</td>
<td>Process stimuli; change level of service</td>
</tr>
<tr>
<td>Measure</td>
<td>Latency, deadline, throughput, jitter, miss rate, data loss</td>
</tr>
</tbody>
</table>
Performance Tactics

• Fundamental issues
  • Resource use
    • computation, memory, bandwidth, system-specific resources
  • Blocking for resources
    • contention, availability, dependencies

• Strategies
  • Reduce resource demand
    • Increase efficiency, lower performance expectations
  • Increase resources
    • Throw money at the problem
  • Coordinate resources
    • Share more efficiently
Performance Tactics: Reduce Demand

- Increase efficiency
  - Better algorithms
  - Trade space for time
- Reduce overhead
  - Avoid costly operations, e.g. indirection
    - Typically conflicts with other quality attributes!
- Reduce event rate
  - Sample environmental data less frequently
    - May reduce precision
- Discard events
  - Sample from input stream
    - Cost: some requests are lost
- Bound execution time
  - e.g. number of iterations in problem
    - Cost: solution may be less precise
Performance Tactics: Increase Resources

• Introduce concurrency
  • Adds complexity, but gets things done faster if there is inherent parallelism

• Duplicate data or computation
  • Avoid contention for shared resources
  • Cache data from a remote server

• Buy more resources
  • Faster CPU, more CPUs, more memory, faster network
  • Only barrier is $$$
    • So, does the value of the additional performance exceed its cost?
Performance Tactics: Coordinate Resources

- Typically scheduling strategies
  - FIFO
    - fair when all requests are equivalent
  - fixed priorities
    - most important
    - shortest deadline
  - dynamic priorities
    - earliest deadline first
  - static scheduling
    - allocate everything up front
    - good for real-time / embedded systems
Security

- Ability to resist attack while remaining functional

Issues

- Confidentiality – can’t see private data
- Integrity – can’t modify data without permission
- Nonrepudiation – can’t deny malicious action
- Availability – no denial of service
## Security Scenarios

<table>
<thead>
<tr>
<th>Scenario Portion</th>
<th>Possible Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
<td>User/system that is identified correctly/incorrectly or unknown, who is internal/external and authorized or not, with full/limited access</td>
</tr>
<tr>
<td>Stimulus</td>
<td>Attempt to display/modify data; access services; reduce availability</td>
</tr>
<tr>
<td>Artifact</td>
<td>System services, data</td>
</tr>
<tr>
<td>Environment</td>
<td>Online/offline, connected/disconnected, firewalled/open</td>
</tr>
<tr>
<td>Response</td>
<td>Grant/block access; audit requests; encrypt data; detect attack; enter degraded mode</td>
</tr>
<tr>
<td>Measure</td>
<td>Cost/benefit of attack to attacker, cost of attack to defender and users; probability of identifying attacker; availability during DOS attack; …</td>
</tr>
</tbody>
</table>
Security Tactics: Resisting Attacks

- Authentication
  - User identification – often username/password
- Authorization
  - Which users can perform which operations?
- Encryption
  - Protect confidential data
- Checksums / signatures
  - Ensure integrity of data
- Principle of least privilege
  - Reduce damage attacker can do
- Limit access
  - Firewalls, etc.
Security Tactics: Detection/Recovery

- Intrusion detection systems
- Audit trails
- Availability tactics
More Quality Attributes

- Testability
- Usability
- Interoperability
- Scalability (Performance? Modifiability?)
- Portability (Modifiability)