Final Exam Review

• Rob Pettit
• SWE 443 - Spring 2015
Topics Covered

• Material since midterm
  – Architectural Requirements and Quality Attributes
Requirements and Quality Attributes

• Requirements can be categorized as:
  – Functional
  – Quality attribute
  – Constraint

• Many different architectures can satisfy functional requirements
  – Architecture is chosen based on the combination of functional requirements, constraints, and quality attributes
Architectural Design Decisions

- Architectural design decisions can be categorized as:
  1. Allocation of responsibilities
  2. Coordination model
  3. Data model
  4. Management of resources
  5. Mapping among architectural elements
  6. Binding time decisions
  7. Choice of technology
Patterns and Tactics

- Patterns are large scale strategies for structuring software architecture designs
  - Augmented by tactics
  - Lower level design decisions that can be applied to address specific quality attributes
Categories of quality attributes (and corresponding tactics) include:
- Availability
  - Detect, recover, and prevent faults
- Interoperability
  - Manage interfaces and locate services
- Modifiability
  - Planning for change
- Performance
  - Predictability, not just speed
- Security
  - Detect, resist, react, and recover from attacks
- Testability
  - Ease with which we can find faults
- Usability
  - Helping user accomplish the desired tasks
Availability Tactics

- Detect Faults
  - Ping / Echo Monitor
  - Heartbeat
  - Timestamp
  - Sanity Checking
  - Condition Monitoring
  - Voting
  - Exception Detection
  - Self-Test

- Recover from Faults
  - Preparation and Repair
    - Active Redundancy
    - Passive Redundancy
    - Spare
    - Exception Handling
    - Rollback
    - Software Upgrade
    - Retry
    - Ignore Faulty Behavior
    - Degradation
    - Reconfiguration

- Prevent Faults
  - Reintroduction
    - Shadow
    - State Resynchronization
    - Escalating Restart
    - Non-Stop Forwarding
  - Removal from Service
    - Transactions
    - Predictive Model
    - Exception Prevention
    - Increase Competence Set

Fault Masked or Repair Made
Interoperability Tactics

Interoperability Tactics

- Locate
- Manage Interfaces
  - Discover Service
  - Orchestrate
  - Tailor Interface

Information

Exchange Request

Request Correctly Handled
Modifiability Tactics

- Reduce Size of a Module
  - Split Module

- Increase Cohesion
  - Increase Semantic Coherence

- Reduce Coupling
  - Encapsulate
    - Use an Intermediary
    - Restrict Dependencies
    - Refactor
    - Abstract Common Services

- Defer Binding

Changes Made and Deployed

Change Requests
Performance Tactics

Control Resource Demand
- Manage sampling rate
- Limit event response
- Prioritize events
- Reduce overhead
- Bound execution times
- Increase resource efficiency

Manage Resources
- Increase resources
- Introduce concurrency
- Maintain multiple copies of computations
- Maintain multiple copies of data
- Bound queue sizes
- Schedule resources

Events arrive → Response generated within time constraints
Security Tactics

Detect Attacks
- Detect Intrusion
- Detect Service Denial
- Verify Message Integrity
- Detect Message Delay

Resist Attacks
- Identify Actors
- Authenticate Actors
- Authorize Actors
- Limit Access
- Limit Exposure
- Encrypt Data
- Separate Entities
- Change Default Settings

React to Attacks
- Revoke Access
- Lock Computer
- Inform Actors

Recover from Attacks
- Maintain Audit Trail
- Restore
- See Availability

System detects, resists, reacts, or recovers
Testability Tactics

- Control and Observe System State
  - Specialized Interfaces
  - Record/Playback
  - Localize State Storage
  - Abstract Data Sources
  - Sandbox
  - Executable Assertions

- Limit Complexity
  - Limit Structural Complexity
  - Limit Non-determinism

Tests Executed

Faults Detected
Usability Tactics

User Request

Support User Initiative
- Cancel
- Undo
- Pause/Resume
- Aggregate

Support System Initiative
- Maintain Task Model
- Maintain User Model
- Maintain System Model

User Given Appropriate Feedback and Assistance
Usability Aspects

- Usability comprises the following areas:
  - Time to learn
  - Speed of performance
  - Rate of user errors
  - Retention of skills
  - Subjective satisfaction
An architectural pattern establishes a relationship between:

- **A context.** A recurring, common situation in the world that gives rise to a problem.
- **A problem.** The problem, appropriately generalized, that arises in the given context.
- **A solution.** A successful architectural resolution to the problem, appropriately abstracted. The solution for a pattern is determined and described by:
  - A set of **element types** (for example, data repositories, processes, and objects)
  - A set of interaction mechanisms or **connectors** (for example, method calls, events, or message bus)
  - A topological **layout of the components**
  - A set of semantic **constraints** covering topology, element behavior, and interaction mechanisms
Examples of Patterns

- Layered / Tiered
  - Supports separation of concerns / portability
- Broker
  - Promotes dynamic binding/discovery of services
- Model-View-Controller
  - Separates user views from back-end algorithms and data
- Pipe and Filter
  - Concurrent processing of data streams
- Client/Server
  - Central control with many distributed clients
- Peer-to-Peer (P2P)
  - Distributed, but equal components
- Publish-Subscribe
  - Producers and consumers unaware of each other, but send and receive data
Relationships Between Tactics and Patterns

• Patterns are built from tactics; if a pattern is a molecule, a tactic is an atom.
• Model View Controller, for example utilizes the tactics:
  – Increase semantic coherence
  – Encapsulation
  – Use an intermediary
  – Use run time binding
• Tactics help to fine tune patterns
  – Address specific quality attributes and tradeoff decisions
Tactics Augment Patterns

- Patterns solve a specific problem but are neutral or have weaknesses with respect to other qualities.
- Consider the broker pattern
  - May have performance bottlenecks
  - May have a single point of failure
- Using tactics such as
  - Increase resources will help performance
  - Maintain multiple copies will help availability
Architecturally Significant Requirements (ASRs)

An ASR must have the following characteristics:

• *A profound impact on the architecture*
  – Including this requirement will very likely result in a different architecture than if it were not included.

• *A high business or mission value*
  – If the architecture is going to satisfy this requirement it must be of high value to important stakeholders.
# Looking for ASRs

<table>
<thead>
<tr>
<th>Design Decision Category</th>
<th>Look for Requirements Addressing . . .</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation of Responsibilities</td>
<td>Planned evolution of responsibilities, user roles, system modes, major processing steps, commercial packages</td>
</tr>
<tr>
<td>Coordination Model</td>
<td>Properties of the coordination (timeliness, currency, completeness, correctness, and consistency)</td>
</tr>
<tr>
<td></td>
<td>Names of external elements, protocols, sensors or actuators (devices), middleware, network configurations (including their security properties)</td>
</tr>
<tr>
<td></td>
<td>Evolution requirements on the list above</td>
</tr>
<tr>
<td>Data Model</td>
<td>Processing steps, information flows, major domain entities, access rights, persistence, evolution requirements</td>
</tr>
<tr>
<td>Management of Resources</td>
<td>Time, concurrency, memory footprint, scheduling, multiple users, multiple activities, devices, energy usage, soft resources (buffers, queues, etc.)</td>
</tr>
<tr>
<td></td>
<td>Scalability requirements on the list above</td>
</tr>
<tr>
<td>Mapping among Architectural Elements</td>
<td>Plans for teaming, processors, families of processors, evolution of processors, network configurations</td>
</tr>
<tr>
<td>Binding Time Decisions</td>
<td>Extension of or flexibility of functionality, regional distinctions, language distinctions, portability, calibrations, configurations</td>
</tr>
<tr>
<td>Choice of Technology</td>
<td>Named technologies, changes to technologies (planned and unplanned)</td>
</tr>
</tbody>
</table>

© Len Bass, Paul Clements, Rick Kazman, distributed under Creative Commons Attribution License
## Analysis at Different Stages of the Lifecycle

<table>
<thead>
<tr>
<th>Life-Cycle Stage</th>
<th>Form of Analysis</th>
<th>Cost</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements</td>
<td>Experience-based analogy</td>
<td>Low</td>
<td>Low-High</td>
</tr>
<tr>
<td>Requirements</td>
<td>Back-of-the-envelope</td>
<td>Low</td>
<td>Low-Medium</td>
</tr>
<tr>
<td>Architecture</td>
<td>Thought experiment</td>
<td>Low</td>
<td>Low-Medium</td>
</tr>
<tr>
<td>Architecture</td>
<td>Checklist</td>
<td>Low</td>
<td>Medium</td>
</tr>
<tr>
<td>Architecture</td>
<td>Analytic Model</td>
<td>Low-Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Architecture</td>
<td>Simulation</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td>Architecture</td>
<td>Prototype</td>
<td>Medium</td>
<td>Medium-High</td>
</tr>
<tr>
<td>Implementation</td>
<td>Experiment</td>
<td>Medium-High</td>
<td>Medium-High</td>
</tr>
<tr>
<td>Fielded System</td>
<td>Instrumentation</td>
<td>Medium-High</td>
<td>High</td>
</tr>
</tbody>
</table>