Architectural Modelling
Overview

• Software Architecture is intended to give us control over Quality Attributes.
• Ideally we’d like to be able to use Software Architecture to predict Quality Attributes.
• We should be able to build a predictive model of the Software Architecture and use the model to predict Qas.
• The current situation is patchy...
Performance: Queueing Models
Data Needed for the Model

• We need the following information in order to model effectively:
  – The distribution for the arrival of service requests
  – The queuing discipline
  – The scheduling algorithm
  – The distribution of service times for service requests
  – Network characteristics

• The theory places restrictions on the distributions
  – Arrivals are usually expected to be Poisson Distributions specified by arrival rate
  – Service times are usually exponentially distributed on the service rate.
  – Some queuing behaviors are excluded such as reneging or jockying
Queueing Model of MVC
Data for MVC

1. Rate of service requests: the View component will service them at some rate.
2. Service requests to the Controller are generated by the View component.
3. Service requests from the Controller to the View component
4. Service requests from the Controller to the model
5. Service requests from the Model to the View Component
Modelling MVC

• We need estimates of:
  – Distribution of external service demands
  – Queuing Disciplines within the queues in front of each component.
  – Network latencies
  – Transfer characteristics:
    • View – Controller
    • Controller – View
    • Controller – Model
    • Model – View

• This is a well-studied area and can produce good predictions if the estimates are good.
• More sophisticated techniques are available.
• Scaling to large numbers of components is an issue
Availability

• Recall availability is defined as:
  – $\frac{mtbf}{mtbf+mttr}$

• Not as well developed as Performance

• Range of different architectural solutions result in different availabilities.

• One key issue is how long it takes to detect that a failure has taken place...

• Example is a Broker system.
Hot Spare

- Active and redundant both receive identical request stream.
- Synchronous maintenance of broker state.
- Fast failover in the event of failure of the active system.
Warm Spare

- Warm broker is maintained at the most recent checkpoint state.
- In the event of failure the system rolls back to the most recent checkpoint.
- This is slower than the hot spare approach
Cold Spare

- Here there is no attempt to synchronise.
- In the event of failure the cold spare is started.
- The system state is recovered via interaction with other systems (so they have to be resilient to failure in the broker)
Analytical Model Landscape

<table>
<thead>
<tr>
<th>Quality Attribute</th>
<th>Intellectual Basis</th>
<th>Maturity/Gaps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Availability</td>
<td>Markov models; statistical models</td>
<td>Moderate maturity; mature in the hardware reliability domain, less mature in the software domain. Requires models that speak to state recovery and for which failure percentages can be attributed to software.</td>
</tr>
<tr>
<td>Interoperability</td>
<td>Conceptual framework</td>
<td>Low maturity; models require substantial human interpretation and input.</td>
</tr>
<tr>
<td>Modifiability</td>
<td>Coupling and cohesion metrics; cost models</td>
<td>Substantial research in academia; still requires more empirical support in real-world environments.</td>
</tr>
<tr>
<td>Performance</td>
<td>Queuing theory; real-time scheduling theory</td>
<td>High maturity; requires considerable education and training to use properly.</td>
</tr>
<tr>
<td>Security</td>
<td>No architectural models</td>
<td></td>
</tr>
<tr>
<td>Testability</td>
<td>Component interaction metrics</td>
<td>Low maturity; little empirical validation.</td>
</tr>
<tr>
<td>Usability</td>
<td>No architectural models</td>
<td></td>
</tr>
</tbody>
</table>
## Analysis in 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>
Types of Analysis

- Thought experiment: just a sort of discussion using informed people.
- Back of the envelope: using very approximate techniques with unreliable assumptions.
- Checklist: collated experience.
- Analytic Model: based on sound abstractions – heavily dependent on estimates being correct.
- Simulation: higher level of detail – less analytic, more concrete.
- Prototype: approximate system in an experimental setup.
- Experiment: fielded system, simulated load
- Instrumentation: measuring the variable of interest.
Summary

• Architecture is the correct level to deal with Quality Attributes.

• Some Qas have good, well established, analysis techniques, others don’t.

• Analysis can be costly depending on how accurate you want it to be.

• Analysis through the lifecycle helps with decision taking.