CMPS 115 Winter 04

Class #7 (2004/01/27)

- Changes/Review
- Paper Prototyping Video
<break>
- Software Architecture
Changes & Lecture 6 Takeaway

- Changes/Notices
  - New submission policy (hard and soft copy)
  - ECP policy/template
  - No 2/12 inspections video, no 2/26 guest; revised syllabus on Thursday; suggestions?

- UI Design
  - Try to walk in the user’s shoes
  - You can try, but you’ll almost certainly fail! So test with users.

- Paper Prototyping:
  - Technique to encourage early experimentation and testing
  - Explicitly low tech to encourage comments at correct level and fluidity
Brownie Points Competition

105 points may be awarded (15/week: 9 by TA, 6 by instructor)
If class total > 80 points, teacher buys at Mission Pizza
* team with max pts chooses time, pizza types *

<table>
<thead>
<tr>
<th></th>
<th>27-Jan</th>
<th>3-Feb</th>
<th>10-Feb</th>
<th>17-Feb</th>
<th>24-Feb</th>
<th>2-Mar</th>
<th>9-Mar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Astrix</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BattleSnake</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Caesar's</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Paper Prototyping Video

Nielsen Norman Group
http://www.nngroup.com/reports/prototyping/
BREAK
Software Architecture

• It’s not a crisp term …

• It’s more important as scale and future risk increases (doghouse to skyscraper; product-line thinking)

• It’s a field of rapidly growing importance and specialization
  – mindset is “construction” (planned, oversee to completion)
  – not “development” (unending evolution, unfolding, growing)
  – http://www.wwisa.org; SEI/CMU certificate programs

• It’s a modeling exercise
  – blackbox components, connectors and interactions
  – art in choosing level of abstraction (details to support reasoning, simplification to be able to grasp)
Software Architecture Purposes

- It’s the quality driver (performance, maintainability, reliability, safety, security)
- It’s the cost driver (different architecture allocates costs differently over lifecycle)
- It’s a point of view (logical, process, physical, developmental) which helps focus requirements, design, development, test, delivery, …
- Architecture can be analyzed (roughly) via change-scenarios early on, for risk reduction
- Architectures can be transformed to explore approach variations
- Architecture as a “thought process” is widely accepted at mature/large organizations
- Architecture languages and tools are emerging; not yet a significant industrial force
Model Component Selection Approaches

- Interfaces (generally functional or messaging)
- Domains
  - application (problem space: in dialysis domain, maybe blood component has \{sampling, packaging, movement, typing\})
  - computer science (file systems, schedulers, etc)
- Domain Entities
  - what do the problem experts talk about? (blood samples)
- Abstraction Layers
  - simplify specification of problem-domain at higher levels
  - iterate, refactor and extract horizontal layers
- Archetype instantiations (recurring patterns)
4+1 View Approach
(Kruchten, 1994)

- Logical View
- Development View
- Process View
- Physical View

Scenarios

End-user
Functionality

Programmers
Software management

Integrators
Performance
Scalability

System engineers
Topology
Communications
4+1 Views

- **Logical**
  - object or domain model (static structure)

- **Process**
  - concurrency and synchronization (dynamic)

- **Development**
  - static organization of SW in dev environment (modularization in repository, builds, tools)

- **Physical**
  - mapping to hardware deployment, distributed aspects

- **Scenarios** (redundant, hence “+1”) but drives discovery and validation of architecture
RM-OpenDistProc Approach

Enterprise Viewpoint
business model

Information Viewpoint
‘universe of discourse’
logical model of data and
data-centric processes

distribution transparent

Computational Viewpoint
software component boundaries
capable of distribution

Information Viewpoint
software component boundaries
capable of distribution

Technology Viewpoint
standards model
map architected objects to
specific std/products

Engineering Viewpoint
distributed elements model
protocols, topologies

Information System
implementation independent
Reference Models

- A fairly complete “pattern of styles” or collection of models to solve a domain’s problems maximizing reuse, quality, etc.
Styles vs Patterns

- Often used loosely
  - “style” being informal, “pattern” being buzzword

- Styles are a gestalt decision over a view
  - a style imposes a rule over an entire aspect
  - risky and ugly to violate chosen style
    - object hooked up to a pipe? Pipe hooked up to DB?
    - Art deco birdbath as Gothic cathedral baptismal font?

- Patterns are local “tactical” solutions to particular forces
  - Façade pattern vs pipes & filters style
Common Styles

- Dataflow Systems -- Pipes and filters, Batch sequential.
- Call-and-return systems -- Main program and subroutines, OO systems, Hierarchical layers.
- Independent components -- Communicating processes, Event Systems. Sometimes called *implicit invocation*
- Virtual Machines -- Interpreters, Rule-based systems.
- Data-centered systems (repositories) -- Databases, Hypertext system, Blackboards.
Pipes and Filters Example

- Each component has a set of inputs and outputs.
- A component reads a stream of data on its input and produces a stream of data on its outputs.
- Input is transformed both locally and incrementally so that output begins before input is consumed (a parallel system).
- Components are called filters; independent, don’t know other filters.
  - If every filter process all data at one go, is *batch sequential*
- Connectors serve as conduits for the information streams and are termed pipes
  - specializations: pipeline, bounded
- Common style: Unix, compilers, signal processing, parallel systems, distributed systems, and functional programming implementations
Pipe and Filter

Pro

- easy understanding of the system's behavior as the composition of filters
- obvious reuse
- easy to maintain and enhance
- support specialized analysis (throughput deadlock analysis)
- support concurrent execution

Con

- often leads to batch processing
- poor for interactive apps
- can be difficult to maintain synchronization between two related but separate streams
- may force lowest common denominator on data transmission, resulting in added work for each filter to parse input and format output data which can, in turn, affect performance and increase complexity of the filters
Assessing an Architecture by Scenario

- Scenario is use or change case
- Two considerations for each scenario
  - Impact analysis
    - use case => performance est
    - change case => change cost est
  - Quality Attribute prediction
    - use case => perceived performance
      (perf est) X (frequency of scenario) -> “scenario throughput”
    - change case => maintainability (avg changed LOC/scenario);
      (change cost est) X (predicted change rate) -> support cost/year
- Combine multiple scenarios for overall assessment
- Good selection of scenarios is critical
Other Ways of Assessing an Architecture

- Simulation-based (high level implementation of architecture)

- Mathematical model
  - McCabe for maintenance cost, cyclo for test costs, etc..
  - Operational research (queuing theory) very useful; Little’s Law is wonderful
    The average number of things in the system is the product of the average rate at which things leave the system and the average time each one spends in the system. (And if there is a gross ‘flow balance' of things entering and leaving, the exit rate is also the entry rate.)
  - But these are *averages*, they are *statistical* so be careful

- Experience-based assessment (qualitative)