“Systems”
ILS, DAMS, and other Acronyms

Week 12
LBSC 690
Information Technology
The System Life Cycle

• Systems analysis
  – How do we know what kind of system to build?

• User-centered design
  – How do we discern and satisfy user needs?

• Implementation
  – How do we build it?

• Management
  – How do we use it?
Systems Analysis

• First steps:
  – Understand the task
    • Limitations of existing approaches
  – Understand the environment
    • Structure of the industry, feasibility study

• Then identify the information flows
  – e.g., Serials use impacts cancellation policy

• Then design a solution
  – And test it against the real need
Types of Requirements

• User-centered
  – Functionality

• System-centered
  – Availability
    • Mean Time Between Failures (MTBF)
    • Mean Time To Repair (MTTR)
  – Capacity
    • Number of users for each application
    • Response time
  – Flexibility
    • Upgrade path
Analyze the Information Flows

• Where does information originate?
  – Might come from multiple sources
  – Feedback loops may have no identifiable source

• Which parts should be automated?
  – Some things are easier to do without computers

• Which automated parts should be integrated?

• What existing systems are involved?
  – What information do they contain?
  – Which systems should be retained?
  – What data will require “retrospective conversion”?
Analyzing Information Flows

• Process Modeling
  – Structured analysis and design
  – Entity-relationship diagrams
  – Data-flow diagrams

• Object Modeling
  – Object-oriented analysis and design
  – Unified Modeling Language (UML)
Some Library Activities

• Acquisition
• Cataloging
• Reference
  – Online Public Access Catalog (OPAC)
• Circulation
• Weeding
• Reserve, recall, fines, interlibrary loan, …
• Budget, facilities schedules, payroll, ...
Discussion Point:
Integrated Library Systems
Digital Asset Management Systems

• What functions should be integrated?

• What are the key data flows?

• Which of those should be automated?
Some Commercial Integrated Library Systems

- (ExLibris) Aleph [academic]
- (Follett) Destiny [schools]
- (SirsiDynix) Symphony [public]
- WorldCat Local
Some Open Source Digital Asset Management Systems

- Archivist’s Toolkit
- Collective Access
- Greenstone
- Omeka
The Waterfall Model

- Requirements
- Specification
- Implementation
- Verification
The Waterfall Model

• Requirements analysis
  – Specifies what the software is supposed to **do**
• Specification
  – “Specification” defines the **design** of the software
• Implementation
• Verification
  – “Test Plan” defines how you will **know** that it did it
• Maintenance
The Spiral Model

• Build what you think you need
  – Perhaps using the waterfall model
• Get a few users to help you debug it
  – First an “alpha” release, then a “beta” release
• Release it as a product (version 1.0)
  – Make small changes as needed (1.1, 1.2, …)
• Save big changes for a major new release
  – Often based on a total redesign (2.0, 3.0, …)
The Spiral Model
Some Unpleasant Realities

• The waterfall model doesn’t work well
  – Requirements usually incomplete or incorrect

• The spiral model is expensive
  – Redesign leads to recoding and retesting
“Rapid” Prototyping

• Goal: explore requirements
  – Without building the complete product
• Start with part of the functionality
  – That will (hopefully) yield significant insight
• Build a prototype
  – Focus on core functionality, not in efficiency
• Use the prototype to refine the requirements
• Repeat the process, expanding functionality
Rapid Prototyping + Waterfall

Initial Requirements

Update Requirements

Choose Functionality

Build Prototype

Write Specification

Create Software

Write Test Plan
Strategic Choices

• Acquisition strategy
  – Off-the-shelf ("COTS")
  – Custom-developed

• Implementation strategy
  – “Best-of-breed”
  – Integrated system
Architecture Choices

• Self-contained (e.g., PDA)
  – Requires replication of software and data

• Client-server (e.g., Web)
  – Some functions done centrally, others locally

• Peer-to-peer (e.g., Skype)
  – All data and computation is distributed

• “Cloud computing”
  – Centrally managed data and compute centers
What do Oregon, Iceland, abandoned mines have in common?
Cloud Computing: Rent vs. Buy

• Centralization of computing resources
  – Space
  – Power
  – Cooling
  – Fiber

• Issues:
  – Efficiency
  – Utilization
  – Redundancy
  – Management
Management Issues

• Policy
  – Privacy, access control, appropriate use, …

• Training
  – System staff, organization staff, “end users”

• Operations
  – Fault detection and response
  – Backup and disaster recovery
  – Audit
  – Cost control (system staff, periodic upgrades, …)

• Planning
  – Capacity assessment, predictive reliability, …
Total Cost of Ownership

- Planning
- Installation
  - Facilities, hardware, software, integration, migration, disruption
- Training
  - System staff, operations staff, end users
- Operations
  - System staff, support contracts, outages, recovery, …
Total Cost of Ownership

According to open source proponents

According to proprietary companies

- Open Source
- Proprietary

- Other Costs
- Price

$
## Some Examples

<table>
<thead>
<tr>
<th></th>
<th>Proprietary</th>
<th>Open Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Windows</td>
<td>Linux</td>
</tr>
<tr>
<td>Office suite</td>
<td>Microsoft Office</td>
<td>OpenOffice</td>
</tr>
<tr>
<td>Image editor</td>
<td>Photoshop</td>
<td>GIMP</td>
</tr>
<tr>
<td>Web browser</td>
<td>Internet Explorer</td>
<td>Firefox</td>
</tr>
<tr>
<td>Web server</td>
<td>IIS</td>
<td>Apache</td>
</tr>
<tr>
<td>Database</td>
<td>Oracle</td>
<td>MySQL</td>
</tr>
</tbody>
</table>
Open Source “Pros”

- More eyes $\Rightarrow$ fewer bugs
- Iterative releases $\Rightarrow$ rapid bug fixes
- Rich community $\Rightarrow$ more ideas
  - Coders, testers, debuggers, users
- Distributed by developers $\Rightarrow$ truth in advertising
- Open data formats $\Rightarrow$ Easier integration
- Standardized licenses
Open Source “Cons”

- Communities require incentives
  - Much open source development is underwritten
- Developers are calling the shots
  - Can result in feature explosion
- Proliferation of “orphans”
- Diffused accountability
  - Who would you sue?
- Fragmentation
  - “Forking” may lead to competing versions
- Little control over schedule
Iron Rule of Project Management

• You can control any two of:
  – Capability
  – Cost
  – Schedule

• Open source software takes this to an extreme
Open Source Business Models

• **Support Sellers**  
  Sell distribution, branding, and after-sale services.

• **Loss Leader**  
  Give away the software to make a market for proprietary software.

• **Widget Frosting**  
  If you’re in the hardware business, giving away software doesn’t hurt.

• **Accessorizing**  
  Sell accessories:  
  books, compatible hardware, complete systems with pre-installed software
Summary

• Systems analysis
  – Required for complex multi-person tasks

• User-centered design
  – Multiple stakeholders complicate the process

• Implementation
  – Architecture, open standards, …

• Management
  – Typically the biggest cost driver
The Grand Plan

Policy

Building and Deploying Systems

Multimedia  Databases  Programming  Search

Web, XML, Social Software

Computers, Networks