ISM 50 - Business Information Systems

Lecture 20

Instructor: John Musacchio

UC Santa Cruz

December 3, 2009
Class announcements

- Final Exam:
  - Monday 12/7, 12-3pm, this room!
Student Presentations
Akamai Case
Internet Bottlenecks

- **First Mile** (Server Capacity) - 70% of website performance problems according to one study

- **Backbone** - Plentiful, but some shortage within metropolitan areas

- **Peering** - Exchange of traffic between NSPs

- **Last Mile** to home
  - 56 K modems are slow
  - Shared LAN limitations
Solutions

- **Expand Bandwidth**
  - Being done

- **Mirroring web cites**
  - Put exact copy of same web page to multiple servers
  - Tricky to duplicate content

- **Caching**
  - Problem: Stale Content
  - Problem: Hard to count “click throughs”

- **Content Distribution Networks...**
Freeflow

- Deployed in 1999
- Akamai Infrastructure
  - 13000 servers in 954 networks by 2001
- Customers -
  - Large Commercial Websites
- Revenue model - $2000 per mbps served
  - (For comparison, normal Internet access cost 500 mbps at time)
2000 Financials

- **$196 Million Loss** (Before special charges)
- $90 million revenue
- %20 gross margin, after deducting
  - server depreciation
  - payments to network partners
  - Data center space
  - But, most expenses of shouldn’t grow at same rate as number of customers, so margin should improve

- **$201.5 million SG&A**
  - (selling general and administrative)
  - (largely sales force cost)
  - Again, this might not grow at same rate as the number of customers.

- **$40 million R&D**
**Competition**

- **Hosting firms (substitute)**
  - Exodus

- **Other CDNs**
  - Sandpiper, Adero, Mirror Image

- **Content Alliances**
  - Akamai’s competitors banded together to share networks
2001 Market Changes

Bad
- Dot-coms bust
- Customers leave
  - “churn rate goes to 22% per quarter”

Good
- Hosting firms go bust (exodus)
- Some CDN competitors go bust.
- Competing CDN alliances mired in problems
Assemble dynamic pages at edges rather than just serve heavy objects

Value proposition
- Performance improvement
- Cost and complexity reduction
- Scalability
- Security

Pricing - higher than old service

Soon edge suite dominated revenue
Dynamic CDN technology: ESI (edge sides includes)

Develop as open standard why?

Akamai not big and credible enough to force a de-facto standard on market
Marketing

■ Difference in selling old vs new products:
  - Old product
    ■ Geared toward speeding up websites
    ■ Revenues of their clients depended on speed
    ■ Easier to get sale
  - New Product
    ■ Simplify company IT function
    ■ Cost vs. revenue center
    ■ Harder sell. More data driven...
  - Consequently new product needs more professional sales force

■ Channels?
  - Distribution Partners (IBM) credibility
  - Direct Sales Force too
Recent Performance

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Consolidated</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Statements of</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Operations Data:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Revenues</strong></td>
<td>$210,015</td>
<td>$161,259</td>
<td>$144,976</td>
<td>$163,214</td>
<td>$89,766</td>
</tr>
<tr>
<td><strong>Total cost and</strong></td>
<td>101,948</td>
<td>112,570</td>
<td>527,580</td>
<td>2,511,108</td>
<td>989,559</td>
</tr>
<tr>
<td><strong>operating expenses</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Net income (loss)</strong></td>
<td>34,364</td>
<td>(29,281)</td>
<td>(204,437)</td>
<td>(2,435,512)</td>
<td>(885,785)</td>
</tr>
<tr>
<td><strong>Net income (loss)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>attributable to</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>common stockholders</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(In thousands, except share data)
Chapter 9

Applications and the Organization
Build vs. Buy?

Purchase off the Shelf
+ less time and cost
+ benefits of using a “standard” solution
+ support available
- must mold org to app
- no potential for competitive advantage

Outsource
+ developers not as familiar with org as you
+ more opportunity for customizing than off the shelf
- contractor may share knowledge with competitors
- contractor may have too much bargaining power

Make
+ most customizable of 3
+ easier iteration between conceptualization and development needed
- most risky
- org may lack competency to do it
It is important to think beyond acquiring an application:

- How do we come with the idea?
- How do we architect it?
- How do we implement?
- How do we extend and maintain it?

For this reason, the software engineering community came up with:

- Application Lifecycle Model
Application Lifecycle

Stages:
1. Conceptualization
2. Analysis
3. Architecture Design
4. Development Evolution
5. Testing and Evaluation
6. Deployment
7. Operations, Maintenance, and Upgrade
1) Conceptualization

What is the vision?

- What are the objectives?
- What is the business case?

- EXAMPLE: HHC to inform flight attendants which passengers are low and high value.

- Business Case:
  - Increase repeat business from high value customers.
2) Analysis

- Describe what the application will do.
- Don’t make highly detailed specifications
- Describe scenarios in which it is used
  - (Use Cases)
3) Architecture Design

- Decompose the application into subsystems
  - **Decomposition** - Divide the architecture into interacting modules.
  - **Assembly** - Find subsystems available for purchase
3) Architecture Continued

- Considerations
  - **Scalability**
    - Can we increase number of users easily?
  - **Extensibility**
    - Ability to add new features later
  - **Administration**
    - Is it hard to keep it working?
4) Development Evolution

- **Develop the details**
  - Develop/program custom subsystems
  - Have contractor build outsourced pieces
  - Put together with off-the-shelf components

- **Incremental**
  - Start with simplest implementation and get it working
  - Later add more features.
5) Testing

- A must!

- If architected well, we can test subsystems independently.

- Alpha test
  - offline test of prototype

- Beta test
  - test in intended environment with cooperative users
6) Deployment

- Convert from previous processes if necessary
  - Example: CISCO ERP (all at once)
  - Or, you could do incrementally
- Train users
  - Example: Frito-Lay HHC
- Data importation
  - (if necessary)
7) Operations, Maintenance, Upgrade

- Maintain Security
- Repair Problems
- Correct performance short comings (Cisco ERP)
- Add features
Application Lifecycle Model
concluding remarks

- ALM rarely followed precisely
- Many times projects loop between stages
- ALM followed more closely in larger companies

Alternative:
- Rapid Iterative Prototyping
  - (Cisco did some of this in the ERP case.)
Chapter 10 – Application Architecture
Object-Oriented Architectures

- Object-Oriented Programming (OOP) Languages
  - C++
  - Java
  - Smalltalk

- The basic unit of modularity in OOP is an object.
Objects

- **Example: Bank account**
  - has a balance of $5000
  - belongs to Joe Schmoe
  - Is a checking account
  - can have money deposited to it
  - can have money withdrawn from it

Attributes

Behaviors
Objects

- An **attribute** is a numerical value or data that is externally visible, and may be changeable.
  - Ex: The bank account’s balance is $5000

- A **method** is an action available at the object interface
  - Other objects invoke method, pass parameters and get returned data or other objects.
  - We can invoke the “check_balance” method and get returned the number $5000
Object Classes and Instances

- Some objects share types of attributes and methods.
  - They have the same class

- Example
  - Class: Bank_Account
  - Instances:
    - Schmoe_Account → balance: $5000
    - Smith_Account → balance: $10000

- Each instance is a separate object with its own data
Declaring Classes

“declare” each class we plan to use in a program

- Example: class: “bank_account”
  - attributes: balance, owner, etc, ...
  - methods: check_bal, withdraw, deposit, ...
    - Later on we fill in the details of what each method does.

- Once we declare a class, we can create instances of it.
  - Schmoe_account, smith_account, etc...
Method Invocation

- Objects communicate with each other by invoking each other's methods
  - (method invocation)

```
Schmoe_Account
Attributes:
  balance: $5000

Methods:
  withdraw( )
  check_bal( )
  ...
```

```
ATM Object

invoke: check_bal( )
  return: $5000

Invoke: withdraw ( $500)
  return: “successful”

```

Terminology:
- Client object -- object invoking the method is the
- Server object - object whose methods are being invoked
Software Objects

- **In OOP an object can**
  - Represent a real-world entity
    - Bank account
  - Be a proxy of a real-world entity
    - Proxy of a customer
    - Other software talks to proxy using method invocations
  - **Model a real-world entity**
    - For purposes of simulation
    - Motion of a train
Software components

- Software components are reusable modules that can be bought from outside vendors.

How is a component different from an object?

- More importance on
  - Encapsulation
  - Well defined and documented interfaces
Component Assembly Tools

- Visual or integrated development environment (IDE)
  - MS Visual Studio
  - IBM Visual Age
  - Symantec Visual Café

- Scripting Assembly - Text based
  - TCL
  - Perl
  - JavaScript
Software Frameworks

- A preexisting architecture and library of components from a common vendor to help developers
- Enables reuse, and ensures component interoperability.
- Examples:
  - Sun J2EE/Java Beans
  - Microsoft .Net
Break
AA Case

Student Talk:
American Airlines Case- Systems Operations and Control center (SOC)

- **Flight Dispatching*** ← focus of case
  - Flight Path
  - Fuel Load
  - en route weather, problems
  - Each dispatcher assigned a geographic area

- **Load Planning**
  - optimize loading of passengers and freight
  - consider runway length, weather, plane type, etc.

- **Crew Scheduling**
  - Crews under strict regulations about amount of time can work
  - Certain crews can fly certain planes
  - seniority
  - positioning for future flights
Dispatch Automation Package

- **Flight tracking application**
  - View 1
    - List of all flights dispatcher responsible for
  - View 2
    - Dependencies of one flight on other flights.

- **Message tracking**
  - *e-mail to flight crew*
### Flight Tracking FD01

<table>
<thead>
<tr>
<th>CRIT</th>
<th>C/P</th>
<th>F/A</th>
<th>A/C</th>
<th>O</th>
<th>TYPE</th>
<th>FROM</th>
<th>FLT/DA</th>
<th>DPT</th>
<th>SKED</th>
<th>LAT</th>
<th>L AVL</th>
<th>LAT</th>
<th>S DLAY</th>
<th>QPAK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Htl</td>
<td>Htl</td>
<td>495</td>
<td>SP60</td>
<td></td>
<td>1555</td>
<td>1459/11</td>
<td>CMA</td>
<td>1355</td>
<td>1406</td>
<td>ORD</td>
<td>1517</td>
<td>0</td>
<td>2</td>
<td>53</td>
</tr>
<tr>
<td>Orig</td>
<td>Orig</td>
<td>517</td>
<td>SP80</td>
<td>Thru</td>
<td>632/11</td>
<td>STL</td>
<td>1415</td>
<td>1413</td>
<td>DFW</td>
<td>1607</td>
<td>C</td>
<td>5</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>Orig</td>
<td>Orig</td>
<td>495</td>
<td>SP80</td>
<td>Thru</td>
<td>1226/11</td>
<td>DFW</td>
<td>1412</td>
<td>1417</td>
<td>BOS</td>
<td>1700</td>
<td>10</td>
<td>2105</td>
<td></td>
<td></td>
</tr>
<tr>
<td>345</td>
<td>345</td>
<td>110</td>
<td>7271</td>
<td>783</td>
<td>1640/11</td>
<td>DFW</td>
<td>1418</td>
<td>1446</td>
<td>ORD</td>
<td>1755</td>
<td>C</td>
<td>-15</td>
<td>2</td>
<td>54</td>
</tr>
<tr>
<td>991</td>
<td>991</td>
<td>495</td>
<td>SP80</td>
<td>361</td>
<td>189/11</td>
<td>LAX</td>
<td>1422</td>
<td>1426</td>
<td>ORD</td>
<td>1817</td>
<td>0</td>
<td>3</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>600</td>
<td>1012</td>
<td>517</td>
<td>SP80</td>
<td>901</td>
<td>632/11</td>
<td>STL</td>
<td>1435</td>
<td>1447</td>
<td>DFW</td>
<td>2007</td>
<td>C</td>
<td>5</td>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>444</td>
<td>444</td>
<td>279</td>
<td>SP80</td>
<td>1832</td>
<td>1216/11</td>
<td>DFW</td>
<td>1435</td>
<td>1419</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Status:** Last update time 16:03:04 GMT

**Note:** This figure is a reconstruction, all data is fictional.
AA IT Architecture

"Passenger Service System" (Does Reservations)

"Fare Pricing Complex" Responsible for updating Fares (3 mainframes)

SABRE LAN

File Servers (Cache FOS data every 2 min)

Dispatcher automation package

SABRE SOC

SOC LAN

Dispatcher Workstations

"Flight Operating System" Maintains Critical Data 1-Mainframe

AMR Intranet
Flight Tracking Application

- Fall 90 - Built as prototype as a way for someone to teach himself OOP

- May 91 - OK to develop application

- Work divided (one person in charge of each)
  - User interface
  - data model
  - data exchange with FOS

- Nov 91 - production installation complete

- 2 months testing
Flight Tracking Application

- **Facts and Figures:**
  - written in Smalltalk
  - 210 classes
  - 2000 methods
  - 160000 lines of assembler code
  - 150000 object instances in memory at all times
Flight Tracking Application

- OOP + good architecture made 3 changes easier
  - Changed how flight was referenced, major change to data model
    - (1.5 weeks)
  - Introduced File servers to cache FOS data
    - (1 day, 4 weeks test)
  - Developed message queuing monitor
    - (1 wk, test 3 wk)
Flight Tracking Application

- **Good architecture allowed extensions later**
  - Feature to allow dispatcher to focus on very really late flights
  - Flight lock - stop flights to airport for bad weather
  - In flight fuel calculation
  - Geographical flight monitor
Did AA follow Application Lifecycle Model?

Fall 90 - Built as prototype as a way for someone to teach himself OOP

May 91 - OK to develop application

Work divided (one person in charge of each)
- User interface
- data model
- data exchange with FOS

Nov 91 - production installation complete

2 months testing

Extensions added Later

1) CONCEPTUALIZATION
2) ANALYSIS?
3) ARCHITECTURE
4) DEVELOPMENT
5) TESTING
6) DEPLOYMENT
7) OPERATION, EXTENSION, MAINTENANCE