Class Announcements

- **Assignment 4**
  - Due Tuesday, November 10

- **Business paper draft due in 1 week!**
  - Due Thursday, November 12

- **Database Assignment 2 posted**
  - Due November 19

- **Pizza Party next week: Tue or Th 3.45 pm**
  - Final on December 8, 4-7 pm, Location TBD
Two ways to design a system

System requirements

Decomposition from system requirements

Available components

Requirements

Assembly from available components

Slide adapted from slides for Understanding Networked Applications by David G Messerschmitt. Copyright 2000. See copyright notice.
Components

A component implementation is encapsulated (although often configurable)

Component: A subsystem purchased “as is” from an outside vendor

(Alternative – building your own subsystem)
The Linux OS we are buying “off the shelf” and integrating into our architecture. The Linux OS is a component.
Other Examples of components

Computer
Disk drive
Network
Network router
Operating system
Integrated circuit
Database management system

Why is a component implementation encapsulated?
Interoperability

- Components are interoperable when they interact properly to achieve some desired functionality.

- Increasingly component interoperability cannot be dependent on end-user integration:
  - PC and peripherals
  - Enterprise, inter-enterprise, consumer applications
  - Role for standardization
Outsourcing: A subsystem design is contracted to an outside vendor

Responsibility is delegated
Suppose we choose to pay another firm to develop the user interface.

This is called **Outsourcing**.

Why would we do this?
Suppose we bring together all these subsystems and test them…

This is called **System Integration**
System integration

- Bring together subsystems;
- make them work together;
- to achieve a goal.

- Requires
  - Testing
  - Making modifications to
    - architecture and/or
    - subsystem implementation
Supplier Types

- Three types of suppliers:
  - Component Suppliers
  - Custom Subsystem Developers
  - System Integrators

- (Some suppliers are 2 or even 3 of above.)
Two ways to sell Software

**Product**
- Customer installed and operated
- Often (but not necessarily) sold or licensed at a fixed price

**Service**
- Functionality provided over a wide-area network
- Often (but not necessarily) sold by subscription

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Recall: Infrastructure and Applications

Infrastructure
- Equipment and/or software used by many applications

Applications
- Provide specific capabilities and features serving individual users.
Four possibilities

<table>
<thead>
<tr>
<th>Product</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Office</td>
<td>Hotmail</td>
</tr>
<tr>
<td>Application</td>
<td></td>
</tr>
<tr>
<td>Infrastructure</td>
<td></td>
</tr>
<tr>
<td>Personal computer</td>
<td>Internet DNS</td>
</tr>
</tbody>
</table>
Application Service Provider

Two types

- **Bundled**
  - An infrastructure provider bundles applications with their infrastructure
  - **Example:** Comcast, telephony service providers

- **Unbundled**
  - A provider of an application service without providing an infrastructure service
  - **Examples?**
Examples of unbundled ASP model

- Yahoo: Web-based calendar
- gmail: Web-based email
- Schwab: Web-based stock trading
Unbundled ASP model

Advantageous to user

- Proven way to reduce installation, integration, and maintenance costs
- Contractual obligation for availability and quality
- Location independence
Unbundled ASP model (con’t)

Advantages to supplier

- Ongoing revenue stream supporting upgrade and maintenance
- Usage-based revenue better aligned with user’s value proposition
- Opportunity for price discrimination, advertising revenue, etc.
Some pricing alternatives

Price discrimination?
Usage dependent?
Terms and conditions
- fixed, leasing, per-use, subscription
- warrantee, service level agreements

Bundles
- maintenance, support, releases, provisioning and operations

Who pays?
- sometimes not the end user
Infrastructure acquisition

Infrastructure

Build and operate

Build but do not operate

Do not build but operate

Neither

Trend

Outsourced operations

System integrator

Service provider

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Application acquisition

Application

\{ Develop internally \quad Buy as product \quad Contract development \quad Product w/ customization \}

Trend

Software supplier

Outsource developer

Supplier, consultants
Stovepipe vs. Integrated Infrastructure

**Stovepipe Architecture**

--- or ---

**Integrated Infrastructure**

- Single supplier provides all encompassing solution
- (complete with infrastructure)

**Turnkey Solution**

- Separate infrastructure that can support many applications
From stovepipe to layering

Data
Voice
Video

Many applications

Integrated Infrastructure (Maybe broken into Additional layers.)

Application-dependent infrastructure
Application-independent
Stovepipe vs. Integrated Infrastructure

- What are some examples of each?

- What are the advantages of each approach?
Vertical Integration vs. Diversification

- A company is **vertically integrated** when it makes rather than buys the subsystems in its products.

- A **diversified** company produces products across different industry segments.
Vertical Integration vs. Diversification

- Why do customers favor less vertical integration?
  - Prefer competition amongst component suppliers
  - Mix and match components
  - Reduced lock in

- Disadvantages??
  - Customer needs to integrate components from different suppliers.

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Vertical Integration vs. Diversification

- Why do customers favor diversification?
  - Reduce coordination costs by having to deal with fewer suppliers.
General Trend

- Less Vertical Integration

- More Diversification

- Of course there are exceptions...
# Today’s supplier structure

- Applications
- Frameworks and components
- Middleware
- Infrastructure (network, OS) software
- Equipment (network, computers)
- Semiconductors, components

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Standardization
Purpose of a standard?

- Allow products or services from different suppliers or providers to be interoperable
Scope of a standard

**Included:**
- interfaces (physical, electrical, information)
- architecture (reference model)
- formats and protocols (FAP)
- compliance tests (or process)

**Excluded:**
- implementation
- (possibly) extensions

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Reference model

Decide decomposition of system
- where interfaces fall

Defines the boundaries of competition and ultimately industrial organization
- competition on the same side of an interface
- complementary suppliers on different sides
- hierarchical decomposition at the option of suppliers
- (possibly) optional extensions at option of suppliers
Some issues

Once a standard is set

- becomes possible source of industry lock-in; overcoming that standard requires a major (~10x?) advance
- may lock out some innovation

In recognition, some standards evolve

- IETF, CCITT (modems), MPEG
- backward compatibility

---

Slide adapted from slides for *Understanding Networked Applications*
By David G Messerschmitt. Copyright 2000. See copyright notice
Types of standards

**de jure**
- Sanctioned and actively promoted by some organization with jurisdiction, or by government

**de facto**
- Dominant solution arising out of the market
- Voluntary industry standards body

Industry consortium

Common or best practice

Examples?
Examples

de jure
- GSM, ISDN Telephone interface

de facto
- Microsoft Windows API (Application Programming Interface)
- Intel Pentium instruction set,

Voluntary industry standards body
- IEEE (Institute of Electrical and Electronic Engineers)
- IETF (Internet Engineering Task Force)

Industry consortium
- bluray

Best practice
- Windowed GUI
The changing process

- As technology and industry move more quickly, the global consensus standards activity has proven too unwieldy
  - e.g. ISO
- “New age” standards activities are more informal, less consensus driven, a little less political, more strategic, smaller groups
  - e.g. IETF
- Programmable/extensible approaches for flexibility
  - e.g. XML, Java
Old giving way to the new

The Standards Making Universe

Traditional Model Telco Bodies

International Coordinating Organizations

AT&T

Soft-ISI

TTC (LUP)

ETS

Traditional Model Information Systems Bodies

ISO-IEC

ICS

RCM

ODS

ATSA

DAS

New Model Telco Bodies

New Model Information Systems Bodies

New Model Radio Bodies

Traditional Radio Bodies

Slide adapted from slides for Understanding Networked Applications
By David G Messerschmitt. Copyright 2000. See copyright notice
Reasons for change

- From government sanction/ownership to market forces
  - Increasing fragmentation
  - Importance of time to market

Greater complexity

- Less physical/performance constraint for either hardware or software
Lock-in

(Particularly open) standards reduce consumer lock-in

- Consumers can mix and match complementary products

Increase supplier lock-in

- Innovation limited by backward compatibility
- e.g. IP/TCP, x86, Hayes command set

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice
Aside: Network Effects

- The value of owning some products goes up if lots of other people have it too.
  - Examples?

- This phenomenon is called “network effects”

- How do standards influence network effects?
Network effects

Standards can harness network effects to the industry advantage

- Revenue = (market size) x (market share)

Increases value to customer

Increases competition

- Only within confines of the standard
- But forces customer integration or services of a system integrator

Slide adapted from slides for *Understanding Networked Applications* by David G Messerschmitt. Copyright 2000. See copyright notice.
Why standards?

*de jure* are customer driven to reduce confusion and cost

*de facto* standards are sometimes the result of positive feedback in network effects

Customers and suppliers like them because they
- increase value
- reduce lockin

Governments like them because they
- promote competition in some circumstances
- May believe they can be used to national advantage

---

Slide adapted from slides for *Understanding Networked Applications*  
By David G Messerschmitt. Copyright 2000. See copyright notice
Approaches

Consensus
- ISO

Collaborative design
- MPEG

Competitive “bake off”
- IETF
Open vs. Proprietary Standards

- Open standard - a standard that is well documented, unencumbered by intellectual property rights and restrictions, and available to any vendor.

- What are the advantages?

- What are the disadvantages?
Why companies participate

Pool expertise in collaborative design
  - e.g. MPEG
Have influence on the standard
Get technology into the standard
  - Proprietary, with expectation of royalties
  - Non-proprietary
Reduced time to market
Standards applied to Business Processes?

- Can you standardize business processes?

- Yes!
  - ISO 9000
    - A set of standardized business processes for Quality Management.
    - Supports TQM (Total Quality Management)
  - RosettaNet
    - A set of standardized business processes, and accompanying standardized data interfaces/formats for conducting e-business.
**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
Application Lifecycle

- 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: Seatback system to sell seat swaps

- Business Case:
  - Increase revenue, passenger satisfaction
Conceptualization

- New in-flight seatback system
  - Sell upgrades and seat swaps
    - (People who want to get away from sick people ...)
  - Offer to exchange seats
2) Analysis

- Describe what the application will do.
- Enough info to allow “stakeholders” to review idea
- Don’t make highly detailed specifications
- Describe scenarios in which it is used
  - (Use Cases)
2) Analysis -- Example

- **Example: Scenario:**
  - **Seat Trade**
    - Passenger in 10C (aisle) offers to trade seat for frequent flyer miles
    - Business traveller in 20B (middle) offers to pay $500 to get aisle seat
3) Architecture Design

- Decompose the application into subsystems
  - Hardware, software
  - Try use commercial off the shelf subsystems
  - Try to use standard infrastructure layers
    - Operating system, network, middleware, etc.
Architecture

HEADQUARTERS

Airline Dataserver

Wireless Link

Seat back devices

Wireless Link

servers
When a module is composed of sub-modules, the architecture is **hierarchical**.
Define the functionality, interaction and interfaces of subsystems

While doing this, consider

- **Scalability**
  - How easily can we increase the number of users and maintain performance?

- **Extensibility**
  - How easily can we add new features in the future?

- **Administration**
  - How much work will it take by humans to keep this running properly?
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
  - Example - give HHC to initial group of FA’s
6) Deployment

- Convert from previous processes if necessary
  - Example: CISCO ERP (all at once)
  - Or, you could do incrementally

- Train users

- 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.)