Designing for Maintainability

Idaho State University | Computer Science

Isaac Griffith

CS 4422 and CS 5599
Department of Computer Science
Idaho State University

ROAR

Idaho State University
Computer Science

Outcomes

At the end of Today's Lecture you will be able to:

- Describe key concepts for highly maintainable designs.
- Describe general methods to achieve these designs.
- Describe how these techniques play out in enterprise software systems.

ROAR

Inspiration

"When I'm working on a problem, I never think about beauty. I think only how to solve the problem. But when I have finished, if the solution is not beautiful, I know it is wrong." – Freeman Dyson

Idaho State University
Computer Science

Designing for Maintainability

1. **Integrating Software Components**
2. Sharing Data and Message Passing
3. Using Design Patterns to Integrate

ROAR

Idaho State University
Computer Science

Modern Software is Connected

- Modern programs **rarely** live in isolation
  - They **interact** with **other programs** on the same computer
  - They use **shared library** modules
  - They **communicate** with programs on **different computers**
  - Data is **shared** among multiple **computing devices**
- **Web applications** communicate across a network
- **Mobile applications** live in a complex ecosystem
- **Web services** connect **dynamically** during execution
- **Distributed** computing is now common

ROAR

Idaho State University
Computer Science

Why Integration is Hard

- Networks are **unreliable**
- Networks are **slow**
  - Multiple orders of magnitude slower than a function call
- **Programs** on different computers are **diverse**
  - Different languages, operating systems, data formats, …
  - Connected through diverse hardware and software applications
- Change is **inevitable** and **continuous**
  - Programs we connect with change
  - Host hardware and software changes

Distributed software must use extremely low coupling

ROAR

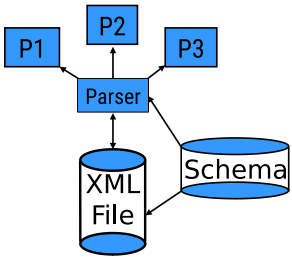
Extremely Loose Coupling

- **Tight Coupling**: Dependencies encoded in **logic**
  - Changes in **A** may require changing in logic in **B**
  - This used to be common
- **Loose Coupling**: Dependencies encoded in the **structure** and **data flows**
  - Changes in **A** may require changing data uses in **B**
  - Goal of data abstraction and object-oriented concepts
- **Extremely Loose Coupling (ELC)**: Dependencies encoded only in the data **contents**
  - Changes in **A** only affects the contents of **B**'s data
  - Motivating goal for distributed software and web applications

The issues are about how we share data...

XML for ELC

- Data is **passed directly** between components
- Components must agree on **format**, **types**, and **structure**
- XML allows data to be self-documenting




```
<book>
  <author>Steve Krug</author>
  <title>Don't Make Me Think</title>
</book>
<book>
  <author>Don Norman</author>
  <title>Design of Every Day Things</title>
</book>
```

- P1, P2 and P3 can see the **format**, **contents**, and **structure** of the data
- **Free parsers** are available

Discussion

Discuss in Groups

- Explain coupling to each other
- Have you used tight coupling?
- Have you used loose coupling?
- Have you used extremely loose coupling?

You must join a group … you may not discuss alone

Idaho State University
Computer Science

Designing for Maintainability

1. Integrating Software Components
2. **Sharing Data and Message Passing**
3. Using Design Patterns to Integrate

ROAR

Idaho State University
Computer Science

General Ways to Share Data

1. **Transferring files**
   - One program **writes** to a file that another later **reads**
   - Both programs need to **agree** on:
     - File name, location, and format
     - Timing for when to read and write it
2. **Sharing a Database**
   - Replace a file with a **database**
   - Most decisions are **encapsulated** in the **table design**

ROAR

General Ways to Share Data

❸ Remote Procedure Invocation

- One program **calls a method** in another application
- Communication is **real-time** and **synchronous**
- Data are passed as **parameters**

❹ Message Passing

- One program sends a message to a common **message channel**
- Other programs read the messages at a later time
- Programs must **agree** on the channel and message format
- Communication is **asynchronous**
- **XML** is often used to implement and encode messages

Message Passing

Message passing is asynchronous and very loosely coupled

Telephone calls are **synchronous**

- This introduces **restrictions**:
  - Other person must be available
  - Communication must be in real-time

- Voice mail and text messages are **asynchronous**
- Messages are left for **later retrieval**
- **Real-time** aspects are less important

Idaho State University
Computer Science

Benefits of Messaging

- Message-based software is **easier to change** and **reuse**
  - Better **encapsulated** than shared database
  - More **immediate** than file transfer
  - More **reliable** than remote procedure invocation
- Software components **depend less** on each other
- Several **engineering** advantages:
  - **Reliability**
  - **Maintainability** & Changeability
  - Security
  - Scalability

ROAR

Disadvantages of Messaging

- **Programming model** is different - and complex
  - **Universities** seldom teach event-driven software (including ISU)
  - **Logic** is distributed across several software components
  - **Harder** to develop and debug
- **Sequencing** is harder
  - **No guarantees for when** messages will arrive
  - Message sent in one sequence may arrive **out of sequence**
- Some programs require applications to be **synchronized**
  - Shopping requires users to **wait** for responses
  - Most web applications are synchronized
    - **Ajax** allows asynchronous communication
- Message passing is **slower**, but good middleware helps

Idaho State University
Computer Science

Discussion

Discuss in Groups

- Have you used message passing?
- Have you learned about message passing?
- If yes, describe to the other members of the group
- If not, do you understand message passing?

You must join a group … you may not discuss alone

ROAR

Designing for Maintainability

1. Integrating Software Components
2. Sharing Data and Message Passing
3. **Using Design Patterns to Integrate**

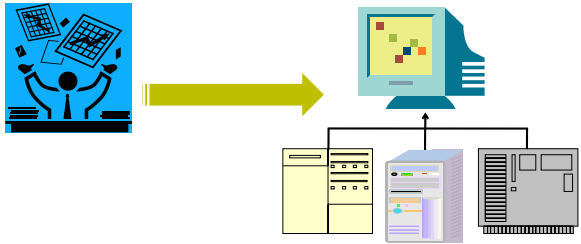
Enterprise Applications

- **Enterprise systems** contain hundreds or thousands of separate applications
  - Custom-built, third party vendors, legacy systems, …
  - Multiple tiers with different operating systems
- **Enterprise systems** often **grow** from disjoint pieces
  - Just like a town or **city** grows together and slowly integrates
- Companies want to buy the **best package** for each task

Thus – integrating diverse programs into a coherent enterprise application will be an exciting task for years to come

Information Portals

Information portals aggregate information from multiple sources into a single display to avoid making the user access multiple systems

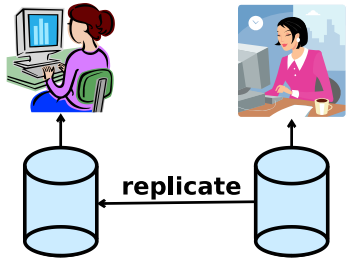


- **Answers** are accessed from more than one system
- Gradesheet, syllabus, transcript
- **Information portals** divide the screen into different zones
- They should **ease moving data** from one zone to another.

Data Replication

Making data that is needed by multiple applications available to all hardware platforms

- Multiple business systems often need the **same data**
- Student **email address** is needed by professors, registrar, department, IT, …
- When email is **changed** in one place, all copies must change

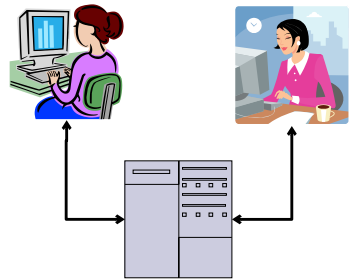


- **Data replication** can be implemented in many ways
  - Built into the **database**
  - **Export** data to files, re-import them to other systems
  - Use **message-oriented** middleware

Shared Business Functions

Same function used by several applications

- Multiple users need the same **function**
- Whether a **particular course** is taught this semester
  - Student, instructor, admins
- Each function should only be **implemented once**

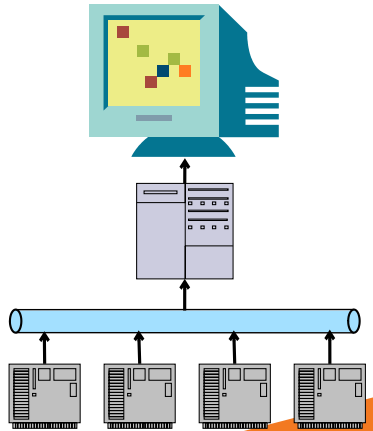


- If the function only **accesses data** to return result, duplication is simple
- If the function **modifies data**, race conditions can occur

Service-Oriented Architectures (SOA)

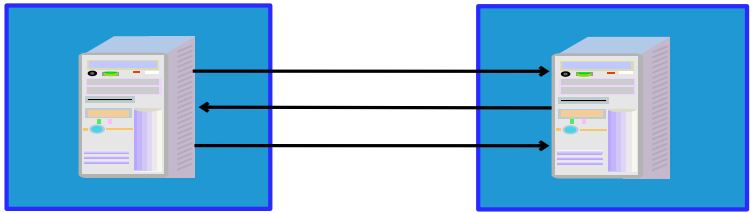
A service is a well-defined function that is universally available and responds to requests from "service consumers"

- Managing a collection of useful services is a **critical function**
  - A service **directory**
  - Each service needs to describe its **interface** in a generic way
- A mixture of **integration** and **distributed** application



Business-to-Business Integration

Integration between two separate businesses



- **Business functions** may be available from outside suppliers or business partners
- Online travel agent may use a **credit card** service
- Integration may occur **"on-the-fly"**
  - A customer may seek the **cheapest price** on a given day
- **Standardized** data formats are critical

Idaho State University
Computer Science

Coupling, Coupling, Coupling

- We have always known coupling is important
- **Goal** is to **reduce the assumptions** about exchanging data
  - Loose coupling means **fewer assumptions**
- A local **method call** is very **tight** coupling
  - **Same** language, same process, typed parameters, return value
- **Remote procedure call** has **tight** coupling, but with the complexity of distributed processing
  - The **worst of both** worlds
  - Results in systems that are **hard to maintain**
- **Message passing** has extremely loose coupling

Message passing systems are easy to maintain

Are there any questions?