

CSCI0320

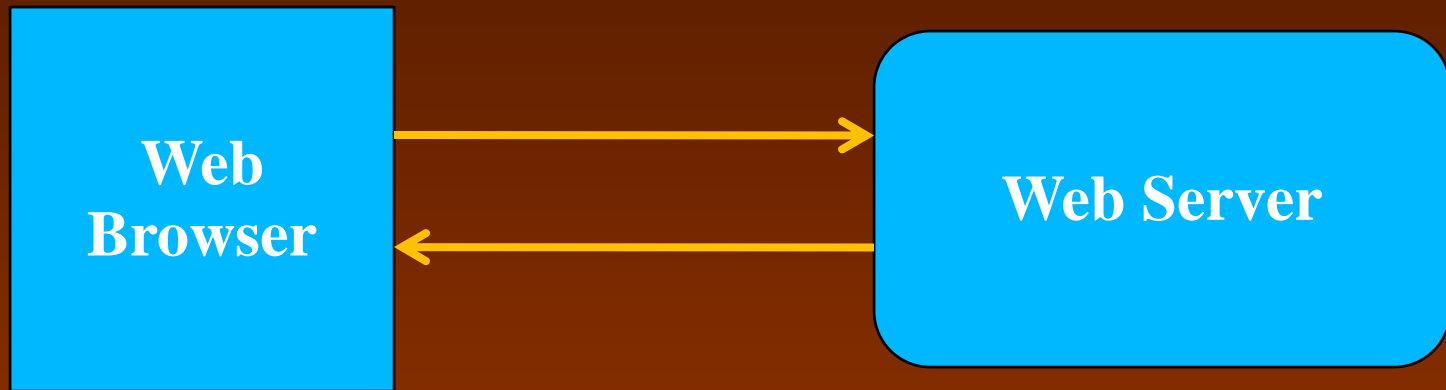
Introduction to Software Engineering

Lecture 9

Team Design

Web Browsing

- Web architecture



- It all works using **sockets** and **messages**
- Server listens on port 80
- Browser connects socket to that port
 - Sends a message requesting a page
 - Using **HTTP** format

HTTP Requests

- Request consists of **header** plus **contents**
 - Header starts with HTTP line
 - **GET <url> HTTP/1.1**
 - **POST <url> HTTP/1.1**
 - Then has a set of name-value pairs
 - **Content-type: text/html**
 - **Content-length: 20393**
 - **User-Agent: cs032-newsrawler**
 - Then has a blank line
 - Then has any content
- Request specifies length
 - Or text itself can be {length,data} elements

HTTP Responses

- **Header + Content**
 - Headers starts with status line
 - **HTTP/1.1 <status code> <message>**
 - **1xx: OK**
 - **2xx: Success**
 - **3xx: Redirection**
 - **4xx: Client error**
 - **5xx: Server error**
 - **Name-value pairs (as before)**
 - **Blank line**
 - **Content**

Java HTTP

- Java provides classes that do all the work
 - **URL**: represents a URL
 - **URLConnection**: handles the connection to the web browser
 - **HttpURLConnection**
- Create a URL
 - Call **url.openConnection** to get connection
 - Then set request properties
 - Can access **output stream** to add data
- Then get **input stream** to read the result
 - Can read header properties and status
 - These cause the connection to be established
 - Use character set **ISO-8859-1** for a Reader

Team Design

- The split between the **browser** and **server**
 - Well-defined interface (175 pages)
 - Allows **separate** implementations
- This is an example of team design
- **Why is this needed?**

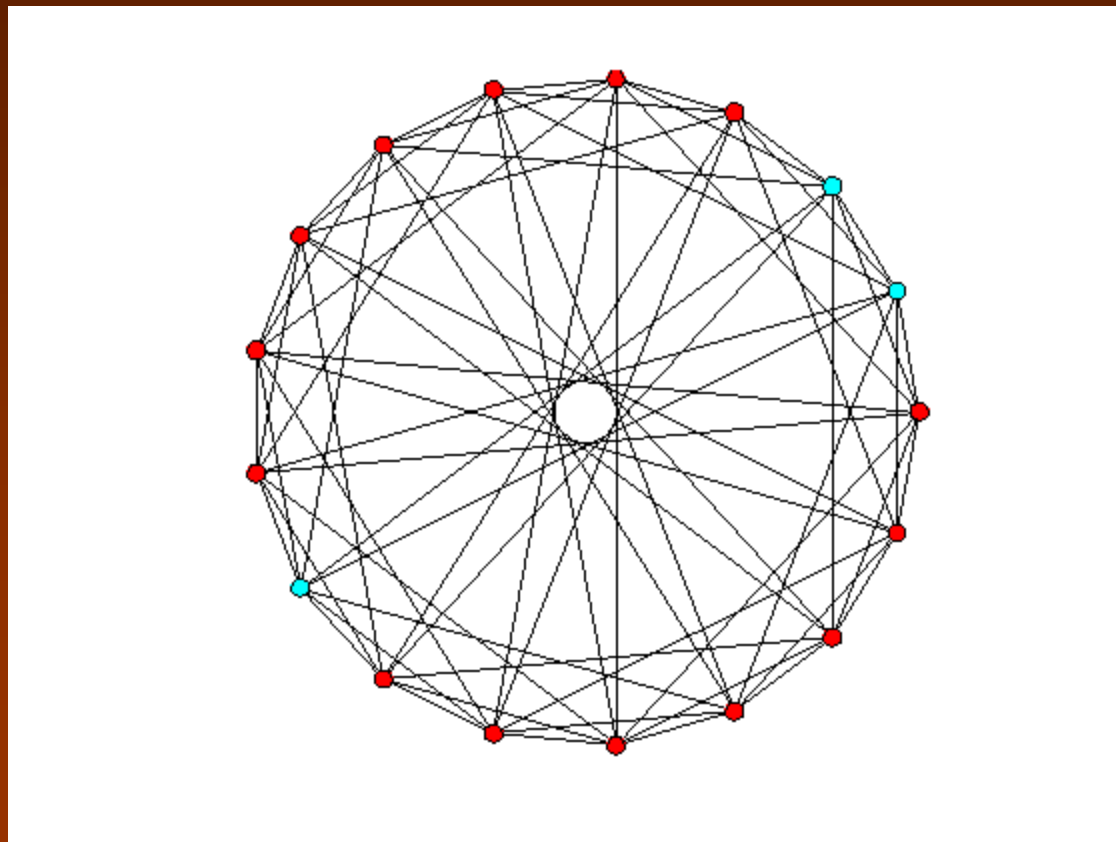
Mythical Man Month Lab

Team Problems

- If **one** person works on a project
 - Works alone, keeps data in head
- If **two** people work on a project
 - They need to communicate all relevant info
 - One channel of communications
- If **four** people work on a project
 - They need to communicate all relevant info
 - Six channels of communication
- If **ten** people work on a project?
- If **100** people work on a project?

Communications

- Number of channels grows as n^2



Communication Problems

- It takes **time**
 - Meetings
 - Waiting for people (finding people)
 - Non-productive time
- It is **error-prone**
 - Miscommunication is likely
 - Missing assumptions
 - Different vocabularies
- Want to **minimize** it
 - Maximize “real” programming

Principles of Team Design

- **Separation of concerns**
- **Abstraction**
- **Simplicity**
- **Well-defined interfaces**
- **Minimize connections**
- **Minimize risk**

Separation of Concerns

- Divide the project into **independent** parts
- **Implementation independence**
 - How each part is implemented
 - Shouldn't affect other parts
 - Design implementations independently
- **Typically matches separation of people**
 - One person might have multiple concerns
- **Examples**

Abstraction

- **Isolate** the implementation from its uses
 - Implementation hidden by an abstraction
 - Uses work with the abstraction
- **Isolate** code of different programmers
 - Each programmer provides an abstraction
 - Others code to this abstraction
- Work in terms of **manageable** units
 - Don't implement something too big at once
 - Able to keep track of what to use, others work, etc.
- **Put off decisions** as long as possible
 - Especially those that affect implementations

Simplicity

- Complexity is generally not worth it
- Don't worry about performance
 - At least not at first
 - Hard to tell where performance is important
 - Hard to tell where fancy code is required
- **Abstract away complexity** where possible
 - Keep things as simple as possible
 - Keep things as abstract as possible

Simplicity

- **Interface simplicity**
 - Minimize exposed information
 - Keep interfaces small
 - Small number of methods
 - Only what is needed, no more
 - No fields should be exposed
- **Code simplicity**
 - Minimize number of arguments
 - Minimize number of auxiliary types
 - Restrict to basic types where possible

Interfaces

- **Work in terms of interfaces**
 - Not data structures
 - Not algorithms
- **Keep the interfaces simple & abstract**
 - Interfaces represent the design
- **For each piece of the program**
 - **Minimize** info it needs to have
 - **Minimize** info it needs to provide
 - Then define the interfaces (Java or UML)
- **Ensure the interface is understood**
 - By all team members (including implementer)
 - **Well documented**

Minimize Communication

- **Between system components**
 - By using **minimal interfaces**
- **Between classes and interfaces**
 - Minimize number of methods in interfaces
 - Minimize communications paths
- **Coupling & cohesion**
 - Minimize dependencies between classes
 - Maximize unity of each class

Minimize Risk

- **Start the design w/ what you don't know**
 - Determine what is hard about your problem
 - Determine what you don't understand
 - Determine what might not work
- **Encapsulate** these problems
 - Isolate the implementation of these
 - From the rest of the system
 - Prototype or experiment as needed

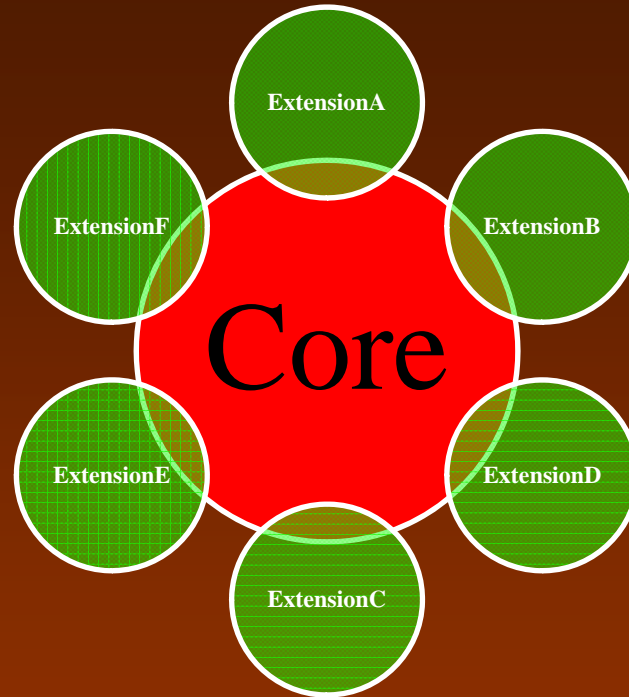
Working in Teams

- **Principles are great**
 - Provide guidelines
 - But they don't tell you what to do
- **High-level system designs**
 - Software architecture
 - What works well for teams
 - What approaches are best

Client-Server Design

- **This is one we've seen**
 - **Client** and **server** are independent
 - **What are their interfaces?**
 - Messages that go back and forth
 - These need to be simple and well-defined
- **Problems**
 - Only provides 2 components
 - Each of these can be complex
 - We need other architectures as well

Core + Extensions



Example: GARDEN

Example: S6

Core + Extensions

- Identify a **minimal system core**
 - Essential elements used by all of system
 - The heart of the application
- Everything else is a **separate extension**
 - **Extensions only talk to the core**
 - **Extensions don't talk to each other**
- Core gets written first
 - Tested the most
- Extensions can be added as needed
 - **Independently**
- Implementation strategy (people)

Features

- Using features/scenarios
 - Fits in with core+extensions model
- Develop base system
 - **Core** implementation
- Add independent features
 - Separate modules where possible
 - Act as **extensions**
 - Augment core as needed (minimally)
- **Microsoft** approach
- **Agile programming** approach
 - Refactor as needed (recompute core)

Library Usage

- **Consider libraries that need to interact**
 - Swing
 - HTML parser
 - SAX xml parser
- **Coded & used by different programmers**
 - Designed for “team” programming
- **How do they accomplish this?**

Publish-Subscribe

- Each module defines **extension** points
 - Where it needs information from outside
 - Tooltips
 - Where it might provide information
 - Mouse events
- Other Modules **subscribe** to these points
 - If they need the information
 - If they want to provide input
- Original module **publishes** response
 - Calls subscribed modules

Publish-Subscribe Frameworks

- **Callbacks** (Swing, HTML parser)
 - Publisher defines an interface class
 - Subscriber provides implementation
 - Passed to the publisher
- **Subclassing**
 - Publisher provides inheritable class
 - With abstract/dummy methods for messages
 - Subscriber provides subclass
 - Providing proper method implementations

External Frameworks

- **Publisher provides a register interface**
 - Subscribers tells publisher their interests
 - Publisher then sends out messages to those who have subscribed for particular data
 - Can work across network
 - Messages stating interest
 - Messages with information
- **Central message service**
 - Handles subscription requests
 - Handles all outgoing messages
 - Resends messages to proper targets

Publish-Subscribe

- **Provides a clean separation of concerns**
 - Each module defines extension points
- **Extension points**
 - Provide small, well-defined interfaces
 - Can request or provide functionality
- **Uses**
 - **Eclipse**: all about plug ins with extensions
 - **FIELD**: central message server

Team Design

- **Iterative Process**
 - You won't get it right the first time
 - Interfaces will change
 - Changing requirements, specifications
 - To match actual implementations
- **Continually refactor the design**
 - Always ask how can I do this better
 - How can it be simpler
 - What can be eliminated
 - Get the best design for the long haul

Final Projects

- **Think about creating independent parts**
 - Each person takes on one or more parts
- **Each part defines a minimal interface**
 - Defined as java interfaces
 - Defined as messages
 - Defined as callbacks
- **Determine interaction between parts**
 - Core + extensions
 - Façade patterns

Final Projects

- All your projects interact with **users**
 - **User interface design** is a critical part
 - Want to provide a good user interface
 - User-friendly
 - Easy to use
 - Easy to learn
 - User efficient
 - Something people want to use
- **Is this easy to do?**
 - How do you achieve it?

Law of Demeter

- **Each unit should have limited knowledge of other units**
 - Each unit should only talk to its friends
 - Don't talk to strangers
- **Friends of a method**
 - Methods of this
 - Methods of argument classes
 - Methods of computed types of values
 - Methods of fields of this
- **Between programmers**
 - Don't rely too much on others