Purpose of CS 3500

CS 3500 is primarily about learning to:

- Solve complex problems with software
- Afterall, software rules the world
- And that, that software should
- Contains fewer defects
- Be more maintainable

Components software systems will need

- Databases
- Networking (e.g., client/server architecture)
- Multi-threading
- GUI
- Logging

IDE

- IDE support - Visual Studio
- Debug tools
- Optimization tools
- Database interface tools
- Multi-Architecture Deployment (e.g., MAUI)
- Intellisense
- Copilot?, GPT?
- XML documentation

Pair Programming

- What is it?
- Why should it be effective?
- What can prevent it from being effective?

Dealing with Imperfections

- Ask questions
- Iterate, (Iterate, Iterate, Iterate)
- See assignment recommendations on “tutorial steps”
- Make assumptions,
- Document the assumptions
- Try not to make the assumptions too broad (or assume it is “hard”)
- Think Abstraction!
- Ask about the assumptions!

Purpose of CS 3500 (cont)

How to you use your time effectively?

- Read all documentation
- Assignment Specifications
- Microsoft Developer Documentation
- etc.
- Understand protocols and APIs
- Networking object
- Agario “Commands”
- Develop toy programs to help you understand

Networking

Networking

- Sockets and Ports
- Event based
- Await and asynchronous elements
- TCP assurances
- Client/Server Architecture
- Protocols
- Allow libraries/abstractions to “do the work” for you

ARPA NET

- Dec 1969
- 4 nodes in utah

Client-Server Architecture

- Client wants to do some “work”/“play”
- Server controls functionality
- Client usually shows the GUI
- Server usually manages the Model/Data

DNS

- DOMAIN NAME SYSTEM
- DNS ☐ Domain Name System Provides “IP Address”

Networking (cont)

IP Address: Unique Identifier

IP Address – a unique identifier for a computer: e.g., 100.200.50.1
Roughly 4 levels of subdomains* (examples below just of intuition)
255.255.255.255
category.identifier.subnet.machine
Category → high level grouping like .com, or .edu
Identifier -> division within the category (like utah or usu/google or amazon)
Subnet -> cs or ece or me
Machine -> my machine or your machine …

PORTS

"MAIL BOXES" ASSOCIATED WITH SPECIFIC PROGRAMS

Client ⇔ Server Communications

Needs:
The address of the server machine
Initial port to talk to
A unique port for future communication
The Protocol(s)!

Initial Port vs. Continuing Port

For Query/Response programs (e.g., a web server):
the server will use a specific low port that is "known" so anyone can make an initial connection
i.e., 80
Ongoing connections will be moved to a different (high) port number
So that new clients can talk to server at the same time

Networking (cont)

Ports - 1

"Mailbox Numbers" for the computer
Unique to for each program
If you try to "open" a port that is already in use you will get an error Note: this could happen if you try to run/debug two versions of the same program at the same time
Numbers Range: 0 - 64k

Ports -2

Who decides number?
Some programs have official ports
Other programs have "taken" over ports
Some ports screened/blocked by firewalls!
Especially low ports under 1000

SOCKETS

OPENING CONNECTIONS BETWEEN CLIENTS AND SERVER

Socket ☐ Unique Channel between Sender and Receiver

Client asks the Server for connection.
A Socket is defined!

Socket

An identifier representing a particular point to point communication connection between two pieces of software
My IP ADDRESS
Their IP ADDRESS
My Port Number
Their Port Number
Combined into a single unique communication channel

Protocols

Agreed order and format of data for communication

Networking (cont)

IP – Internet Protocol

Responsible for sending packets of information from host to host
Hand-wave Hand-wave Hand-wave (or Abstraction/Separation of concerns)
The internet and C#'s usage of it just works!

TCP – Transmission Control Protocol

Runs on top of IP (Internet Protocol)
One to One Reliable Communication
Data will arrive
Verified Ordering
Verified Uncorrupted
Does not verify when data arrives or how much arrives at a given time!
C# libraries do all the work for you

UDP – User Datagram Protocol

Alternative to TCP
No Handshaking – no persistent connection
No guarantee of
Delivery Ordering
Duplication Protection
Why would we use this?
Faster -less overhead

Basic Network Communication Facts:

Happen at the BYTE level!!!!
Your program must
Translate useful data into bytes and
Translate bytes into useful data (e.g., strings, objects, etc)
TCP does not guarantee
When Information Goes Out
When Information Arrives
How much information is sent at any one time…
TCP does guarantee
order and validity

Stacking Protocols

Web browsing looks something like this
HTTP
TCP
IP

Networking (cont)

WHAT THE SERVER'S CONNECTION THREAD DOES

A SERVER DOES TWO THINGS:
"NETWORK STUFF" (E.G., HANDLE MULTIPLE CLIENT REQUESTS AND CONNECTIONS)
"APPLICATION STUFF" (E.G., MANAGES A GAME)

TCP handling in C#

A TCPListener object is at the heart of the server.
Listens on a specific port for incoming connection requests.
TcpListener
BeginAcceptSocket – Wait for request to arrive
EndAcceptSocket - Obtains the Socket.

Server Main Thread Connection Listening Code (Do you see the "Loop"?)

What I call an Event Loop
Server Process
Thread 1 : // Purpose ☐ Await Connection
Start up code:
Build TCP listener
Wait for Connection(event_handler)
Function event_handler()
Build new socket identifying client
Build new Thread to handle client
Wait for Connection(event_handler)

Key Issues

How do you convert from an "object type" to an "actual" Type?
Why is IAsyncResult.AsyncState not typed….
What is meant by an Event (Receive Event) loop?

Async

Async - tells the system that the code may "pause" and "return" later
Allows other threads to execute

Networking (cont)

Await

Await - tells the system to “wait” here (pause thread) until an “event” happens

Some Key Async Ideas

Async methods should do “something” that may take a long time
E.g., DB lookup, network connection, big computation

Some Key Await Ideas

await keyword tells program to
“yield” cpu to other tasks while the long running operation completes
Then resume at the given line when task is done
WARNING: may be on a different thread

Server Issues - Multithreading

Two Threads:
Client Accepting
Message Sending
All Shared Data (e.g., Client array) must be protected
E.g., Removing items from the list
Lock (this.client_array)
{
this.client_array.Remove(x)
}

Some Key Networking Issues - Data Arrival

You never know how much data will arrive at any given time
Some or all may “Show up” - What does arrive will be in order
Therefore: all data concatenated/stored
Search “all data” for messages
Protocol of “what is a message?”
Newline/Period/etc.

Networking (cont)

Key Parts to Networking Object

Callbacks
This is how the network code communicates state to the “user” program
TcpClient/TcpListener objects
The main object you will be using
See SimpleChat[Client/Server]TcpClient code
ID
A unique identifier for the connection Settable (default → socket)

Cancellation Tokens

Long Running (possibly infinite) processes sometimes need to be interrupted.
This functionality is built into the system via:
Cancellation
Exception

Good Software Practices

Maintainable/Testable Software

Simple (as possible)
Organized/Architected (e.g., MVC)
SOLID (e.g., Separation of Concerns/Single Responsibility)
DRY
Documented/Commented/Describable
READMEs
Pictures/Figures/UML
Testable

Teamwork

Working with another developer
Versioning, e.g., git usage
Iterating
Asking questions & Discussing

Debugging

Write down four (or more) Debug tools/windows you use (should use) regularly:
Watch Window
Immediate Window
Threads Window
Stack Window
Break Points/Continue/Stepping
Output Window (e.g., for Logger)

Good Software Practices (cont)

Optimizing and Profiling

Rule 1: Don't optimize until necessary
Rule 2: Optimize algorithm
Rule 3: Optimize the code that is being run the most
Use Profiling to define this code.

Readability

Readability (understandability) is everything.
Why?
Maintainability
Defect Management
Ease of Testing

Maintainability/Readability

Proper Naming
variables/methods/classes/namespaces/etc.
Less is better/Smaller is better
Divide and conquer
Don't repeat code!
Delete code instead of add new code
Separate functionality
Code does one thing well (a SOLID principle)

Draw Pictures, Figures, Diagrams

What is the purpose?
Document your strategies with figures/pictures/diagrams.
UML is a good start

High Level Commenting

Document
Describe high level interactions of functions/classes/etc in "header" comments
Describe tricky coding details using inline comments

Good Software Practices (cont)

Versioning

Why Version?
Find changes
☐ find errors, backup code, understand dev history…
When do you version?
Every day and/or after every important change
How do you version
Commit with _useful_ messages, then push
Other?
Tags, GitHub, etc.

MVC

Model View Controller?
Why separate out these "parts"?
The smaller the code, the "closer" you are to it, the less likely there will be defects

Testing

Unit Tests should be coded for all projects, but especially the "Model" parts.
Test First
Transparent and Opaque Testing (Open/Close Box)

Common Coding Styles

Consistent Styles
I_Dont_Care_If_You_Use_underscores_But_DoNotMixWithCamelCase
Also, keep consistent with capitalization
What tools do we have to assist us?

"Life-Long" Learner

You will need to continually be reading/studying/practicing/improving your craft.
Often this will require self-motivation and drive on your part.
New Tools/Libraries can often solve old problems.
Warning: don't automatically upgrade every time a new version comes out

Good Software Practices (cont)

Project (Code) Life Cycle - Industry

Requirement Gathering
Design/Architecting
Implementation
Verification/Testing
Maintenance
Note: Nonlinear, incremental, “circular”

Self Documenting Code

“Self-documenting code is ostensibly written using human-readable names, typically consisting of a phrase in a human language which reflects the symbol’s meaning…”

Code Analysis

Apply computer programs to Analyze and Improve computer programming
Code Analysis Tools
Complexity Analysis - e.g., how many lines of code
Duplicate Code Analysis - e.g., could be refactored into methods
Style Guides - e.g., put spaces around parameters
Code Cleanup - e.g., remove unused using statements

Code Analysis Continued

Code Analysis is run upon compilation
Shown as warnings/messages in Error List

Design Patterns

Well understood and beneficial ways to effectively solve recurring problems
Two Examples:
Iterator
Null Object

Anti-Patterns

Common ways of solving problems that have unintended side effects
Examples:
hard coded constants
premature optimization

Good Software Practices (cont)

Compiler Optimization in C#

In C#, we don’t have much control
Optimization is done by the runtime (JIT)
And partly by the CIL compiler
This is fine for most purposes
C# is not the language of choice for (super) high performance…

Databases

Databases

Relational - Tables “linked” by related keys
Note: there are non-relational DBs
Two Database Table Design Goals
No repeated data
No empty data
SQL (basic) syntax
SQL tools (e.g., SSMS)
C# SQL interface

Why Databases?

You could craft your own solution to save and retrieve data…
Or you could take advantage of decades of research
Performance
Availability
Stability
Concurrent access
Backups
Interface
Shared/Remote Service
Don’t reinvent the wheel

Databases: Two major components

Database Management System (DBMS)
All the underlying machinery
(e.g., SQL Server 2019)
Query Language
Interface to the machinery

DBMS Goals (ACID)

Databases (cont)

Atomicity

Operations succeed completely, or not at all
E.g., if money is transferred from one account to another, the operation could not fail after the money has been removed from the first account and before it has been added to the second.

Consistency

Operations leave database in consistent state
E.g., cannot assign a “user” into a Roll (table) if the user does not exist

Isolation

Operations don’t interfere with each other
E.g., A Multi-threaded Database and a single threaded one act the same

Durability

Results will not be lost
E.g., if the “building loses power” any committed data will be accurate upon restart. (Data saved to disk.)

Database Tables - RELATIONAL

Structure of ed data storage
An “entity” like a person is stored as a Row in a Table

Databases (cont)

SQL Relationships

SQL takes care of managing and EFFICIENTLY traversing/merging relationships!

```
SELECT Title
FROM Patrons
JOIN CheckedOut ON Patrons.CardNum = CheckedOut.CardNum
JOIN Inventory ON CheckedOut.Serial = Inventory.Serial
JOIN Titles ON Titles.ISBN = Inventory.ISBN
WHERE Patrons.cardNum = 7
```

Basic DB Design Goals

Entries should be atoms (not complex)
Don’t store lists/arrays in a cell (use multiple rows)
Build compound information by referencing other tables
Enables powerful reasoning about data and relationships, cleaner design
Enable DBMS to optimize

SQL-Structured Query Language

Interface for accessing a relational database
Get data
Set data
Change data

Special Creation Parameters

Primary Key
Unique (no other row can have this value)
Identity (via properties)
Increment Auto
Count up giving unique value for each new row

REST – Representational State Transfer

For our purposes this means:
The URL tells the server everything it needs to know in order to take an action.
Often this is specified as an API

C#

C# programming

C# is one of many choice (albeit a good one)
it could have been C++, or Ruby, or Python, or Java, or Go, or ….
What are elements of the language that support GSPs?
Long winded essay question possibility:
“Describe multiple reasons we should use C# on our next project to support making maintainable software?”

Why learn/use C#

C# is simple, readable and easy to use
C# is all about developer productivity
C# is very flexible and allows you to develop a big variety of systems.
Console applications Desktop applications (Windows Forms, WPF) Windows Services Web Services and Web applications (ASP.NET Core, Blazor) Native Mobile Applications (.NET MAUI) AI Applications (ML.NET) Distributed and Cloud Applications (Azure) Games (Unity) IoT applications Reusable libraries
C# runs on a solid well-engineered .NET runtime
C# has built-in design patterns and best practices

Some Niceties of the C# Language

Functions as Values - Delegates, Event Handlers, etc.
Lambdas, Closures
Events - Design code around events “happening”
Threading
OOL
Library support
Exceptions
Garbage collection
GUI Support
Much much more.

C# (cont)

DLL -Dynamic Link Library

Your compiled code
Let’s take a look at the GUI Chat Client release folder
Logging dll
Communications dll
File Logger Dll
Etc., etc., etc.

What do we do with DLL

At Run Time
The system combines (links) all the DLLs as necessary

Debug vs. Release

When Deploying:
Almost always the DLL should be built in
RELEASE mode
More efficient - removes debug symbols
When Developing
Almost always the DLL should be built in
DEBUG mode
Allows debugger to interact with running program
In rare “why does it work on my machine but not when deployed” situations, you can deploy DEBUG code and ATTACH to it!

Logging

Dependency Injection

Key Ideas
Code (objects) need other Code (objects) to work (or to be more effective)
Databases, Loggers, UserManagers, etc.,
Where do those other objects come from?
Static Globals? Parameters?
Dependency injections help reduce reliance on "Globals"

Logging (cont)

Summary – Dependency Injection

Objects that need other objects to help do their job
Want to be able to use different objects without modifying source code:
Define functionality as an interface
Pass object in at construction time (and save in class field)
Important: only works for classes that the “system” builds for you
E.g., you MAUI program MainPage!

Logging Defined

Logging is the act of recording information about the what is happening during the run time of your program

Where should you put log info?

Logs can be sent to one _or more_ of the following:
Console
Debug Display in Visual Studio
GUI
Files
Databases
Windows Event Log
Etc.

Log Levels

Some information is more important (to certain people) than other.
Too much information is often not useful.
Log Levels
Refers to how much and how “high a level” (or low) the information is
E.g. Debug vs Critical Error

0 Trace

every little detail.. way too much information... turn this off most of the time
Stakeholder: Developer - Something really odd is going on…
Show me all the mouse X,Y values as the mouse traverses the screen

Logging (cont)

1 Debug

standard detail level... usually too much info to store... use this for development
Stakeholder: Developer - Standard Development Cycle
Show me every collision between two objects in the game

2 Information

high level information... for example "when a client connects"... use this to understand "meta-level" of application
Stakeholder: Developer/Site Administrator
Show me who logged into the system.

3 Warning

something is probably going wrong, but we haven't handled it yet
Stakeholder: Developer/Site Administrator
Show me when an unauthorized user attempts to access data

4 Error

something has gone wrong and we didn't know how to (or didn't want to spend the time) to handle it
Stakeholders: Site Administrators, Project Lead, (Developer)
Bank software cannot transfer to another bank

5 Critical

the application is about to die... can we do it gracefully?
Stakeholders: Site Administrator, Project Lead
Bank software “died” because database is not accessible

Logging (cont)

6 None

turn off messaging.
Reasons to do:
My code is perfect.
Disk space is very expensive.
(Note: neither of these are a reason to do this, nor are they true)

Decisions - Where in Code to place Logging

Any “big” action
Connection on the net
Any “complicated” action
For debugging complete message building
Any “catastrophic” event
Let developer manager know where to go back to look for problem

Example of levels to log messages at

Position of mouse
_logger.LogTrace($ “{MousePosition}”);
Partial messages coming over internet
_logger.LogDebug($”Data Receive: {data}”);
Information (or maybe just warning)
New Chat Client Connected
_logger.LogInformation($” {name} connected from {ip_address}”);

Setting Up Logger on a C# program

To (correctly) use Loggers and Dependency Injection, you must instrument the following places of your code:
The Main Program
The Class Constructor
Class Methods

Threading

Threads (applies to _all_ languages)

Important Issues:
Race Conditions
Locking shared data
Deadlock
Two (or more) threads each waiting on the other(s) to release a lock
Easy fix → One lock/Order Locking
Warning: Debugging changes behavior. How?
No assurance of ordering
Asynchronous!

Networking and GUI → Threads

Network communications come in “Async”
Handled on DIFFERENT threads
GUI work MUST BE DONE on the GUI thread
Solution:
Dispatcher.Dispatch(( ) => { MessagesAndStatus.Text = $"Server Shutdown!" + Environment.NewLine; }

Dispatcher

Dispatcher
Puts requests for GUI changes into a queue
When the GUI thread “has a chance”, take care of requests in queue
Always runs on GUI thread
Warning - More than one thread (GUI + networking(s)): LOCKING!!!
If your GUI thread deals with an object that a networking thread also uses, you must protect the code using locks!

Threading (cont)

Race Condition – server code

The Clients list is “shared” by all threads, thus becoming a Race Condition
How do we fix this?
Answer: locking!
Warning for Server: Don’t “hold on to” client list while doing lots of work (e.g., sending messages to all clients)
Copy client list (while locked)
Then iterate over the copied list!

MVC

MVC -Model View Controller

Tendency is to write all of our logic together
Assignment 7 Server “logics”
Networking ← Model
GUI ← View/Controller
Connected Clients ← Model

Separation of Concerns

GUI Server
Everything to do with storing multiple clients should be “abstracted” (i.e., put into a model class)
Everything to do with Networking should be put into the Networking class
Everything to do with the GUI should stay in the GUI class

Invalidate

i.e., PlaySurface.Invalidate();
Can be called anywhere (not just on GUI thread)
Tell GUI Thread: At next opportunity, redraw the affected region
Suggested Options
Always (inside Draw Scene) → BAD :(
“Burns” cycles
On Message Received → Locked with Server → Okay
Timer → X times per second → Good as well
Especially if your client did “interpolation”!

JSON

JSON Serialization - “Recursion”

Is it recursive?
Yes - all objects inside public Properties are also serialized.
Deep Copy (not shallow copy)
“Cloning”
?? Circular References ??

JSON Serialization - Default Operation

What constructor is used BEFORE data put in:
Default constructor
What is serialized? Public properties
What is not serialized?
Fields, private properties
How do we change that?
Attributes (Metadata Tags)

JSON De/Serialization - How does it work?

Reflection
Runtime - inspect code
Deserialization
Calls default* constructor
“Finds” the names of the properties
Reads the JSON looking for matching names
Puts data directly into properties
Serialization
“Finds” names of properties and auto-generates json string

Important JSON Attribute Tags and Options

[JsonConstructor]
Specify the specific constructor to use
Constructor parameters must match names of Json Fields
[JsonInclude]
Force property to be part of the serialization
[JsonIgnore]
For property to not be part of the serialization
[JsonRequire]
If field _not_ there then error

JSON (cont)

Parsing JSON

```
Food f = new Food();
List<Food> list = new();
list.Add(f);
message = JsonSerializer.Serialize( list );
JsonSerializer.DeSerialize<List<Food>>( message );
DeSerialize(message, new List<Food>().Type())
{CMD_Food}[{X:50,Y:50,ID:5..},...]
```

Web Servers

Web Server

Simply a program that listens on a port for a connection (e.g., from a web browser) and sends back web pages
Using HTTP -Hypertext transport protocol

HTTP sequence

Point browser to a web server, e.g.,
http://www.webpage.com
Browser opens socket to address
Server accepts socket
Browser sends text:
“GET / HTTP/1.1” + stuff
Server replies with text:
“HTTP/1.1 200 OK …” + “<html>…</html>”
Server closes socket – every HTTP request uses a new connection

Protocol

Just a bunch of strings sent through sockets on the net…
Request, e.g.,
GET / HTTP/1.1
Host: www.cs.utah.edu
Response, e.g.,
HTTP/1.1 200 OK
Content-Type: text/html; charset-UTF-8

Web Servers (cont)

HTTP Web Response

Just a string that follows the protocol, e.g.,
Header
[New Line]
Body

HTTP Web Response Example

```
HTTP/1.1 200 OK\n
Connection: close\n
Content-Type: text/html; charset=UTF-8\n
\n
<text or html>
```

Body → HTML (simple example)

```
<!DOCTYPE html>
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph.</p>
</body>
</html>
```

URL -Uniform Resource Locator

scheme://host:port/path?query
scheme → protocol (e.g. ftp, http, https)
host → server name or address
e.g. www.cs.utah.edu
Port → 80/443 by default
We will continue to use 11000

Web page extras

CSS
Cascading Style Sheets
Send type: “text/css”
Makes things look nice

User Secrets

Secrets - “Infrastructure” information you do not want in the code base, e.g.,
Location of DB
User of DB
Password for DB
Other data that might be changed by a System Admin (not a developer)