Welcome!

CS 346 Application Development

https://student.cs.uwaterloo.ca/~cs346/1259/

Introduction

Dr. Jeffery Avery

- Associate Professor, Teaching Stream
- Cheriton School of Computer Science
- Teaches CS 346 & CS 349

Caroline Kierstead

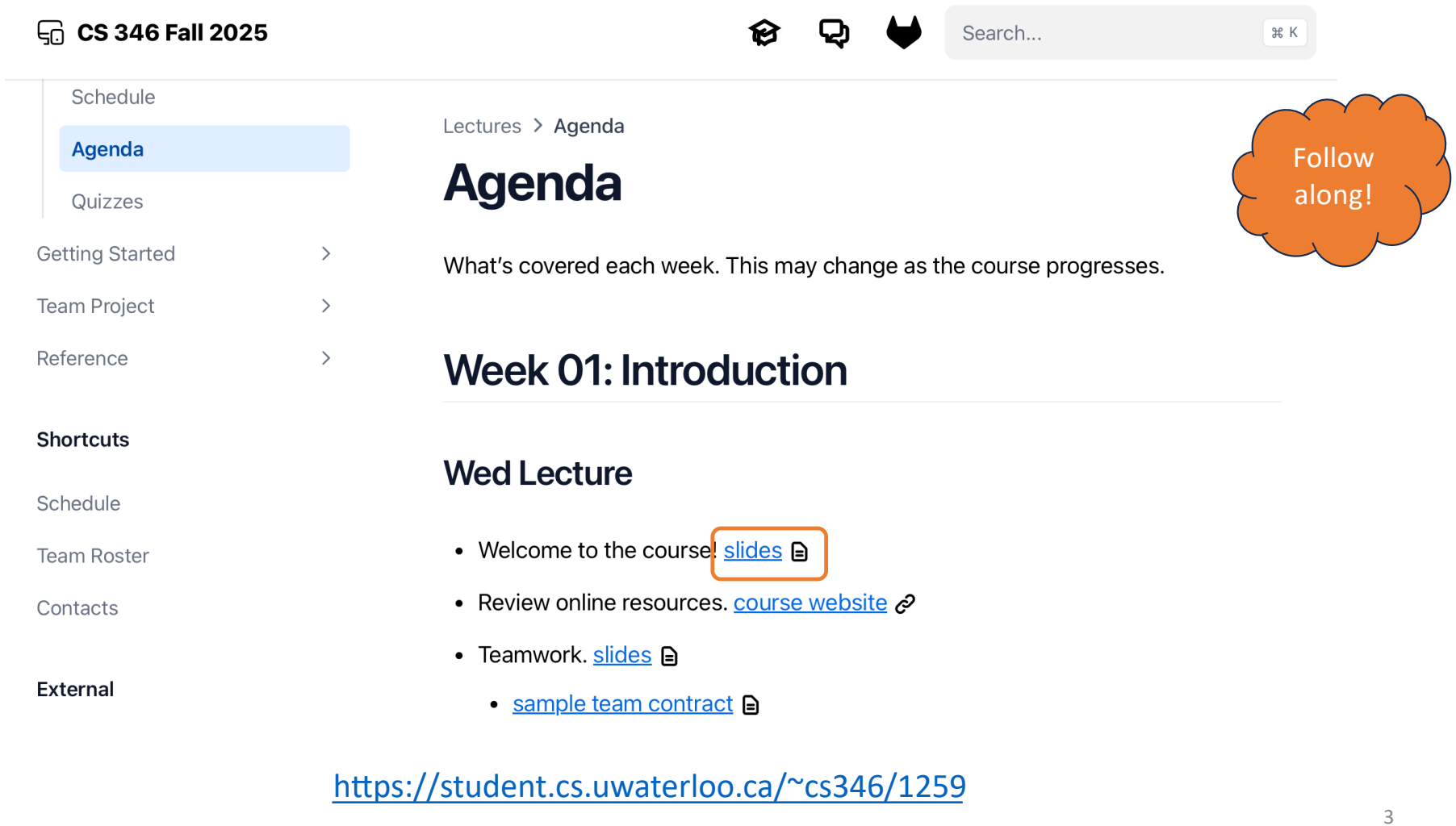
- Instructional Support Coordinator

Teaching Assistants x 6

- See website

Prof. Avery aka “Jeff”

jeffery.avery@uwaterloo.ca
MC 6461



https://student.cs.uwaterloo.ca/~cs346/1259

Course Overview

What are we going to be doing this term?

Course Description

CS 346 Application Development
LAB, LEC, TST 0.50

Introduction to **full-stack application design and development**. Students will work in **project teams** to design and build complete, working applications and services using standard tools. Topics include **best practices** in design, development, testing, and deployment.

Prerequisites: CS 246; Computer Science students only.

https://student.cs.uwaterloo.ca/~cs346/1259/

Application Development

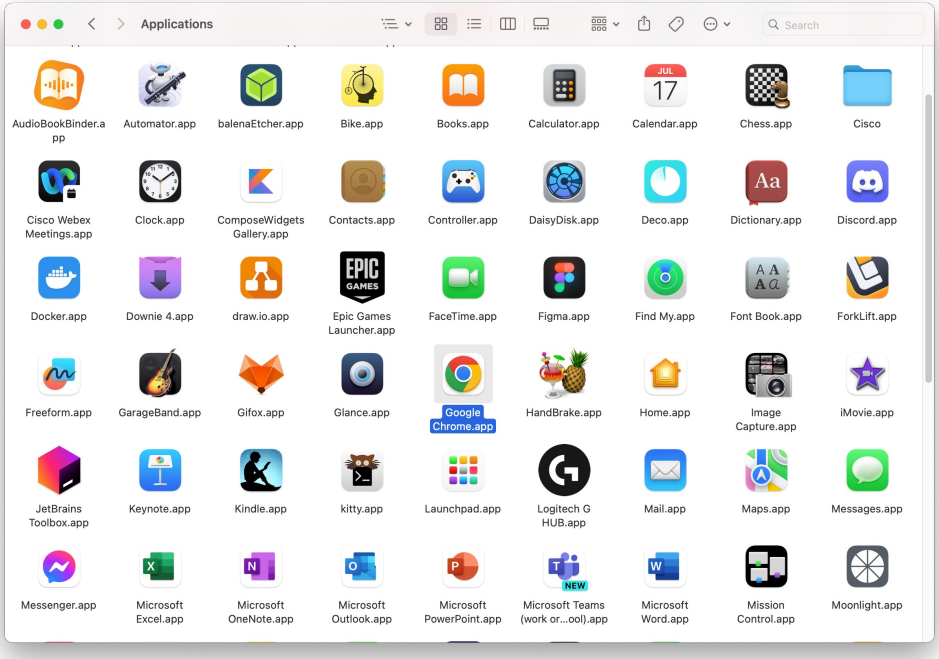
This is a course about building “applications” -- **software that solves problems for people.**
Most of the software that you use on day-to-day basis are considered applications.

Application software tends to be:

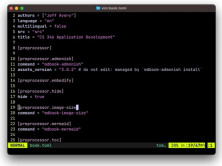
- Focused on providing value for a person.
- Interactive and graphical.
  - e.g., calculator, MS Word, Fortnite.
- Tends to be very personalized.
  - Installed on personal devices.
  - Save profiles/preferences.
  - Personal data (confidential?)

Contrasts with other types of software:

- Embedded -- controls hardware systems.
- System software -- provides services for other software.



Styles of Applications

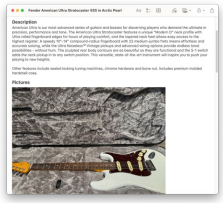


1950s - 1960s

Console

C++ (Python, C)

Imperative



1970s – 1980s

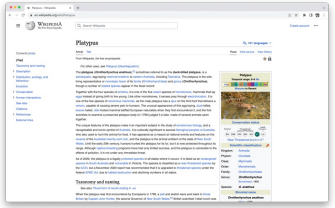
Desktop

Pascal, C++, VB

Object-Oriented

Graphics

Databases



1990s– 2000s

Web

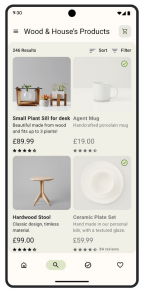
Java, JS, HTML/CSS

Functional

Databases

Networking

Concurrency



2010s – 2020s

Mobile

Kotlin, Swift

OO & Functional

Graphics

Databases

Networking

Concurrency

What Are We Building?

You and your team will pick a project that you think will be interesting, and which expertly solves a problem for users.

Requirements

- Graphical application, desktop (Windows, macOS, Linux) or Android.
- Your application should be *reasonably complex* and *challenge your team*.
- You must define which features to implement. They should excel in solving the problem you have identified.

Technical Requirements

- Kotlin programming language, Gradle for builds.
- Need to use an online Web API (service).
- Also need to store data in a SQL database.

Collaborative Development

A great team can accomplish amazing things together.

- This project will be too complex for a single person.
- It's a chance to improve teamwork and leadership skills.
- You can also learn to how to "code together" (branching, merging, writing meaningful documentation).
- You work together to choose a project, make design decisions, and help one another in the implementation.

Teams need to be four people

- Everyone registered in the same section.
- Physically present during the term.

Past Projects

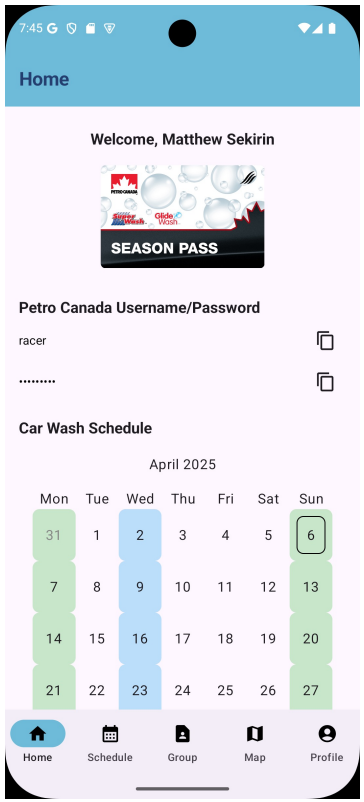
Typically, 90% design Android applications, 10% desktop.

Examples

- Recipe Keeper: a made-up recipe sharing application.
- Software Design Tool: a tool to collaborate on UML diagrams.

Past Projects

- WatCourse: a course planner that lets you build a class schedule.
- FlickPicks: rate movies and share movie reviews with friends.
- Roomie: track chores (assign work to your housemates!)
- Washare: share a car-washing card & subscription.



https://student.cs.uwaterloo.ca/~cs346/1259/project/project-gallery/

Why take this course?

- Technical skills
  - Kotlin programming.
  - Functional + OO paradigms.
  - Databases, User interfaces, Concurrency.
- Software engineering
  - Software Architecture, Design, Testing.
  - How to use standard development practices.
- Teamwork
  - How to be effective on a team.
  - Leadership, responsibility.

I was going to place “pillars of knowledge” under these headings, but that seemed too cheesy, even for me.

Course Structure

Let’s discuss how the course is put together!

Class Structure

Day	Morning Sections	Afternoon Sections
Wednesday	LEC 001 @ 10:30a - 12:20p / MC 4040	LEC 002 @ 2:30p - 4:20p / MC 4040
Friday	LAB 101 @ 10:30a - 12:20p / MC 4040	LAB 102 @ 2:30p - 4:20p / MC 4040

Wednesday (Lecture)

- 1h 20m lectures + 30 min free
- Regular lectures e.g., software architecture, concurrency, user interfaces.

Friday (Lab)

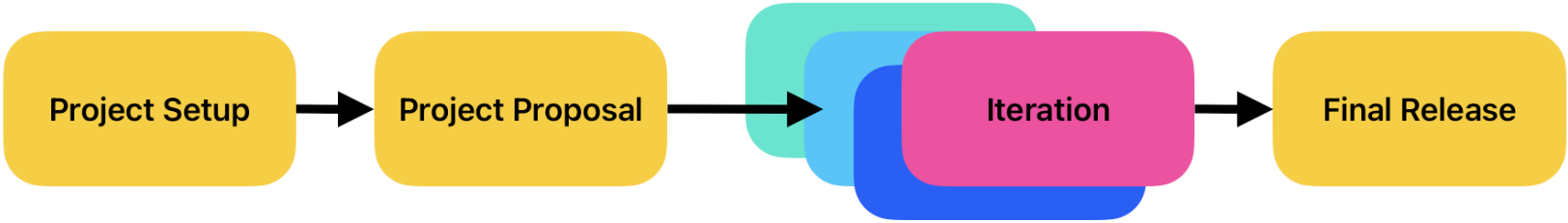
- 20 min demo/Q&A + 1h 30m free
- Time to work on your project.
- TAs and instructor present to answer questions and help!

Term Structure

Week	Day	Topics
Week 01	Wed LEC	Introduction. Teamwork. Project details.
	Fri LAB	Setup GitLab. How to add documentation.
	DUE	*Nothing due in week 01. Find a team!*
Week 02	Wed LEC	Software engineering. Determining requirements.
	Fri LAB	Entering requirements in GitLab.
	DUE	***Project Setup** due Fri @ 6:00 PM*
Week 03	Wed LEC	Learning Kotlin
	Fri LAB	Install the toolchain. Setup the Git repository.
	DUE	***Quiz 1** (Weeks 1-2) due Fri @ 6:00 PM*

See Schedule for the entire term.

Project Structure



Deliverable	What is it?	Date
Project Setup	Form your team, Setup your project space.	Week 02
Project Proposal	Pitch your product idea to your TA!	Week 04
Sprint 1	Architecture, Core classes, Unit tests.	Week 06
Sprint 2	UI prototypes and working user interface.	Week 09
Sprint 3	Database working, Cloud functional.	Week 11
Sprint 4	Final release, all features complete.	Week 13
Final release	Final software package + documentation.	Week 14

Assessment

There are individual grade items, but most of your grade comes from team activities.

Deadlines every 2 weeks

- Submit what's completed
- Demo to your TA (feedback!)

Everyone participates!

- You get a zero for a missed component.

Individual Grade (20%)

Component	What it addresses	Grade
Quizzes	5 quizzes covering lecture and lab content.	5 x 2% = 10%
Participation	Contributions to the team project.	10%

Team Grade (80%)

Item	What it addresses	Grade
Proposal	Project identified, requirements captured.	5%
Sprint 1: Architecture	Features complete; process followed.	10%
Sprint 2: User Interface	Features complete; process followed.	10%
Sprint 3: Databases	Features complete; process followed.	10%
Sprint 4: Final Review	Features complete; process followed.	10%
Software Release	Completed project, including documentation.	35%

General Policies

- **Illness**
  - You must submit a VIF, email the instructor, and coordinate with your team. A missing component grade will be redistributed across other components.
- **Short-term absences**
  - STA cannot be used for team deliverables i.e. demos, final submission. It can be used for quizzes, which would shift the weight of the quiz you missed.
- **Exams & INC**
  - This course has no final exam and an INC will never be granted for missing team deliverables. Other policies would apply – *see next slide*.

https://student.cs.uwaterloo.ca/~cs346/1259/syllabus/policies/

Team Policies

- **You must form teams by the end of week 2.**
  - We will help, but you are responsible for finding a team.
  - You will have time after-class to match with teams. See Piazza as well.
- **You cannot take this course while on a (remote) work term.**
  - You must be physically present to meet with your team & attend class.
  - You *cannot* take it remotely and "call in" for demos.
- **You need to physically attend and participate in team demos.**
  - You will receive a grade of zero for a deliverable if you don't participate in the demo.
  - In extreme cases, we can remove you from the course or adjust your final grade (down) if you repeatedly fail to engage.

Code Policies

- **“Borrowed Code”**
  - You may use external code (up to 25 lines) with appropriate citation
  - You cannot use projects from previous offerings of this course.
  - You cannot use any portion of a project from a different course
    - e.g., CS 446 or CS 449 are also project courses. You cannot normally submit the same project in two different courses.

- **Generative AI / LLMs**
  - Can be used for code analysis. e.g., ”Gemini, what does this code do?”
  - If used to generate code, it should be treated like any other source. i.e., must be documented, subject to restrictions like any other citation.
  - Cannot use an LLM to generate more than this!

Week 01: Introduction

- Wed lecture (today)
  - Introduction ✓
  - Course website
  - Teamwork
  - *Meet people and form teams*
- Fri lab
  - Forming and registering teams
  - Setting up your project space
  - How to write project documentation
  - *Free time for teams to work ahead*