

Introduction

CS 346: Application
Development

Introduction

Dr. Jeffery Avery

- Associate Professor, Teaching Stream
- Cheriton School of Computer Science

Caroline Kierstead

- Instructional Support Coordinator

Teaching Assistants x 6

- See [website](#)



Prof. Avery aka “Jeff”

jeffery.avery@uwaterloo.ca

MC 6461

Introduction

This course is a course about “Application Development”.

What makes this course different from the other software development courses you’ve taken?

- CS 135 – Designing Functional Programs (Racket)
- CS 136 – Elementary Algorithm Design and Data Abstraction (C)
- CS 246 – Object-Oriented Software Development (C++)

So far, you’ve mostly worked on small, single-purpose programs.
Text-based, command-line.

We're building applications

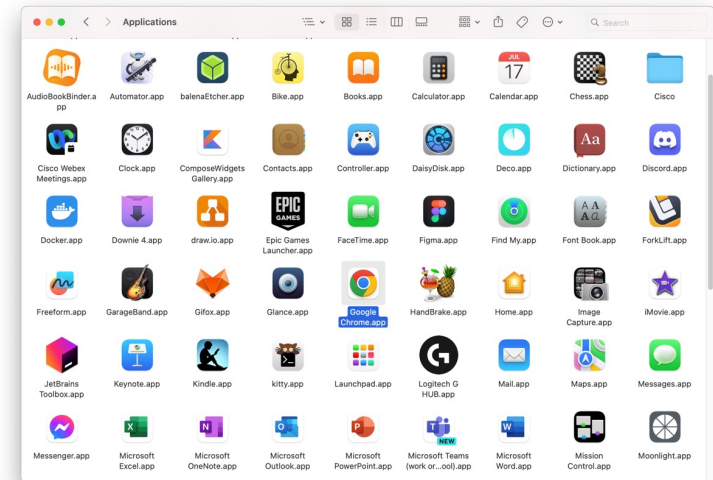
Modern, consumer software

Software that solves problems for users!

- Can include mobile, desktop, web apps.
- e.g., Diablo, WhatsApp, Word.
- Graphical, media-rich, interactive.

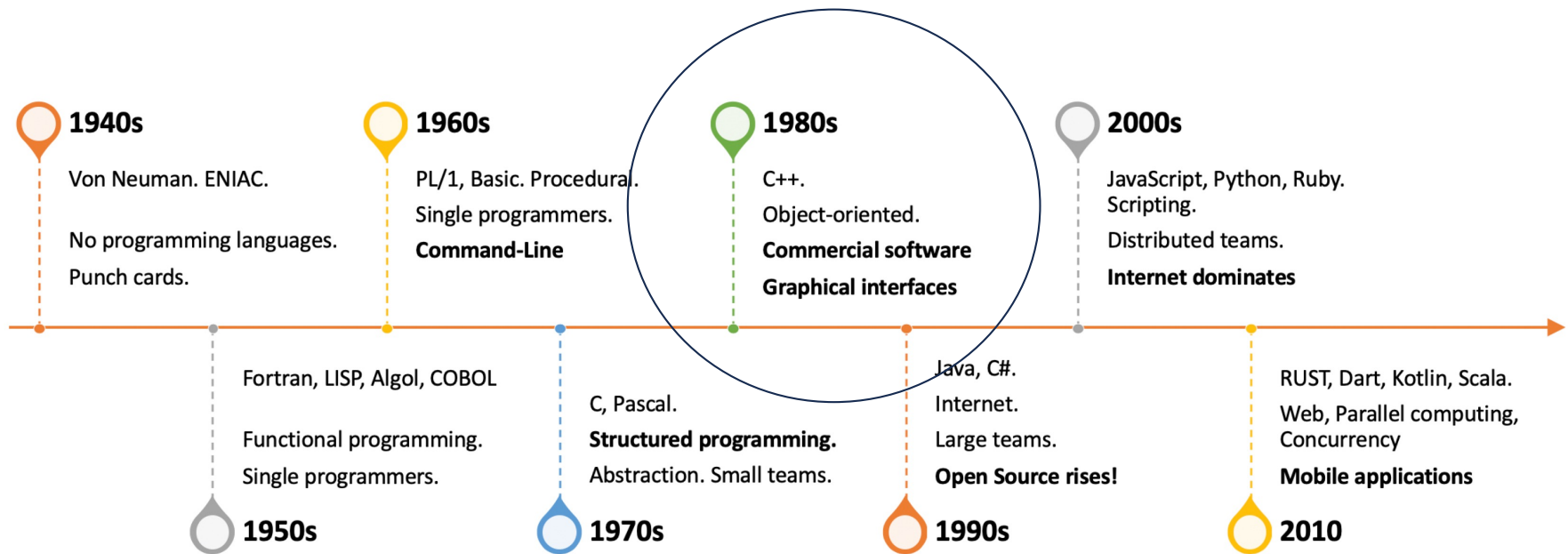
Challenges

- Increased complexity, that requires some discipline.
- We can't write the entire thing from-scratch!
- We can't do it alone anymore.

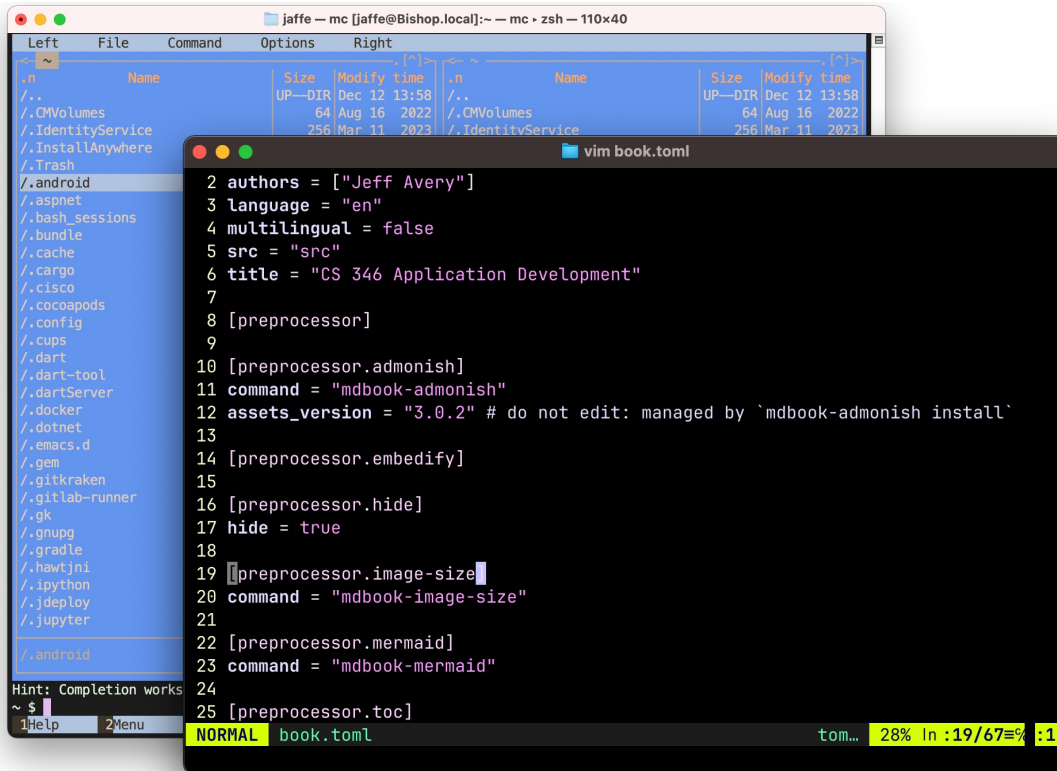


What is an “application”?

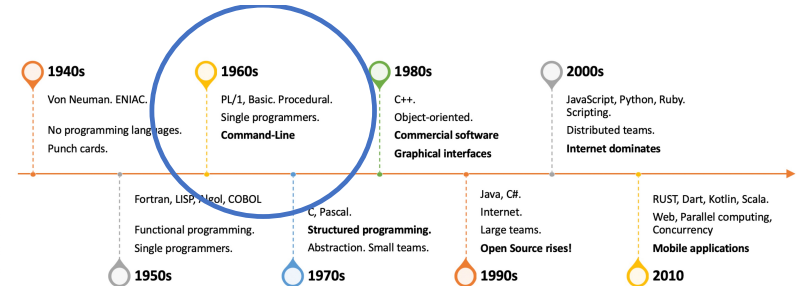
Our definition of “application” has evolved



Software has evolved to keep pace with research, and hardware innovations.
What we build and how we build it has changed drastically over time.

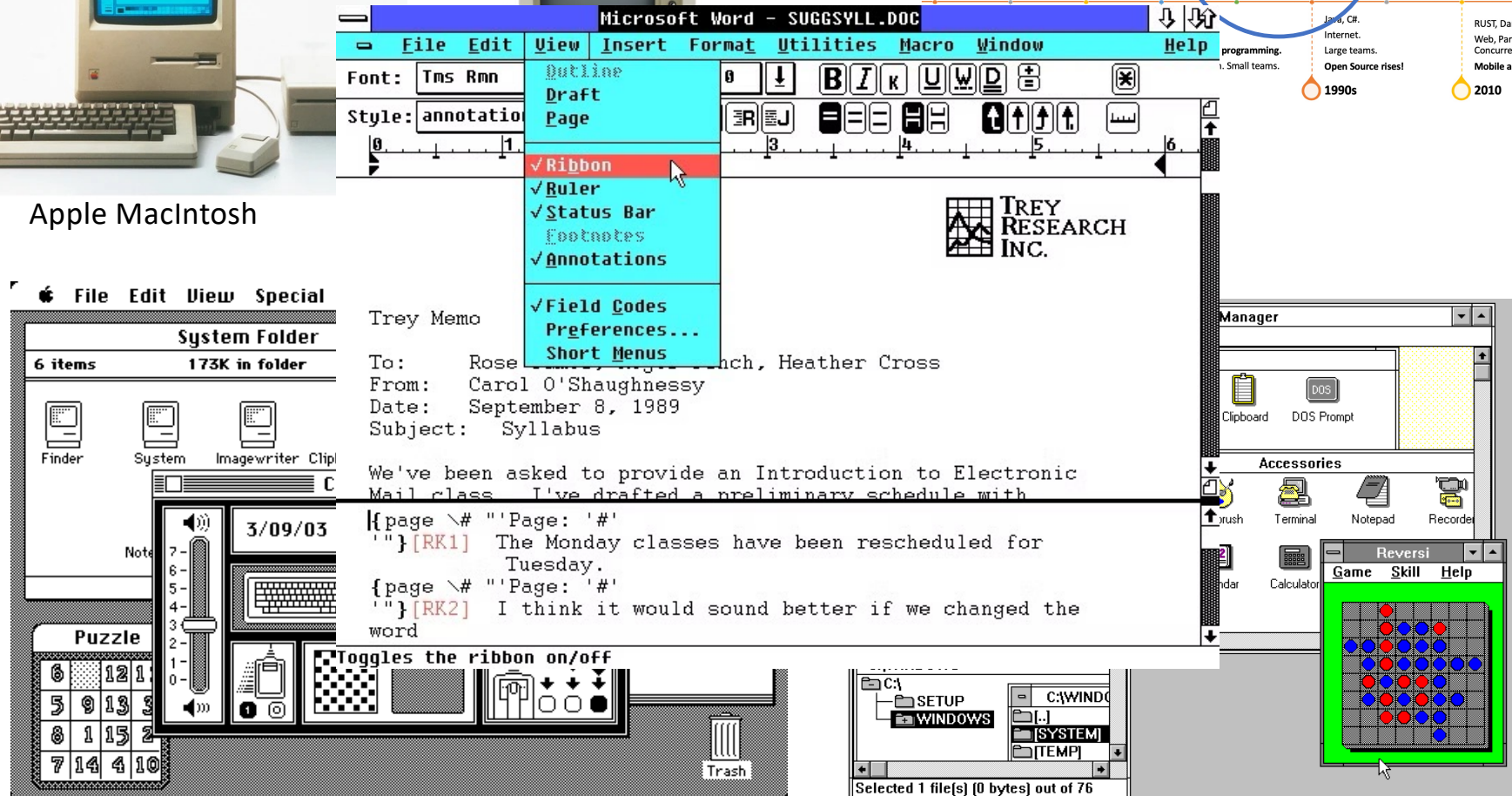
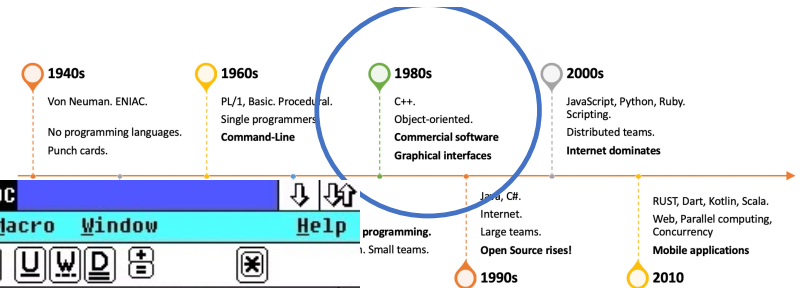


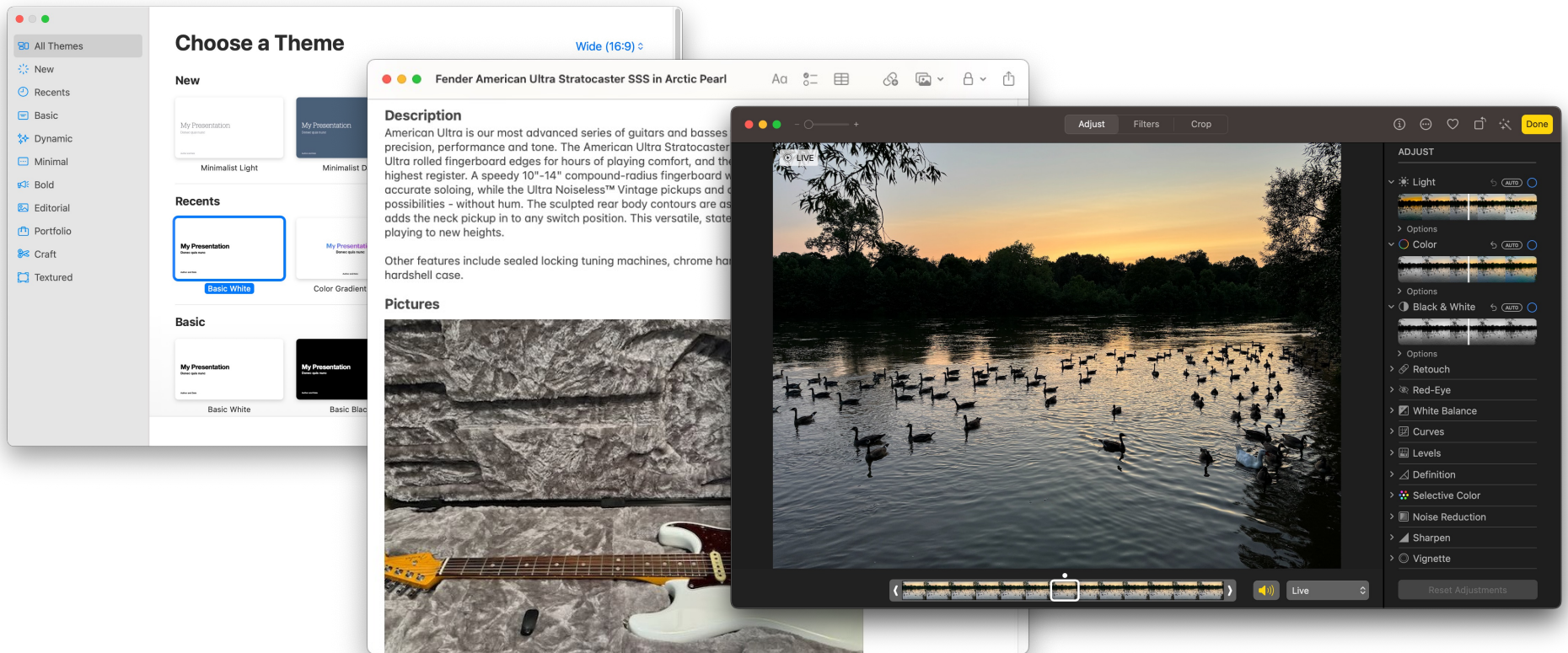
Terminal applications



DEC VT-100 terminal

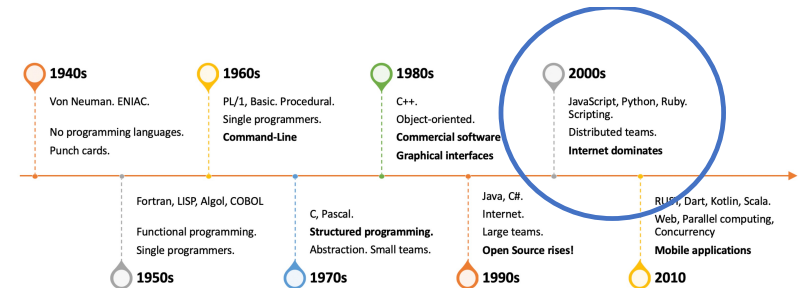
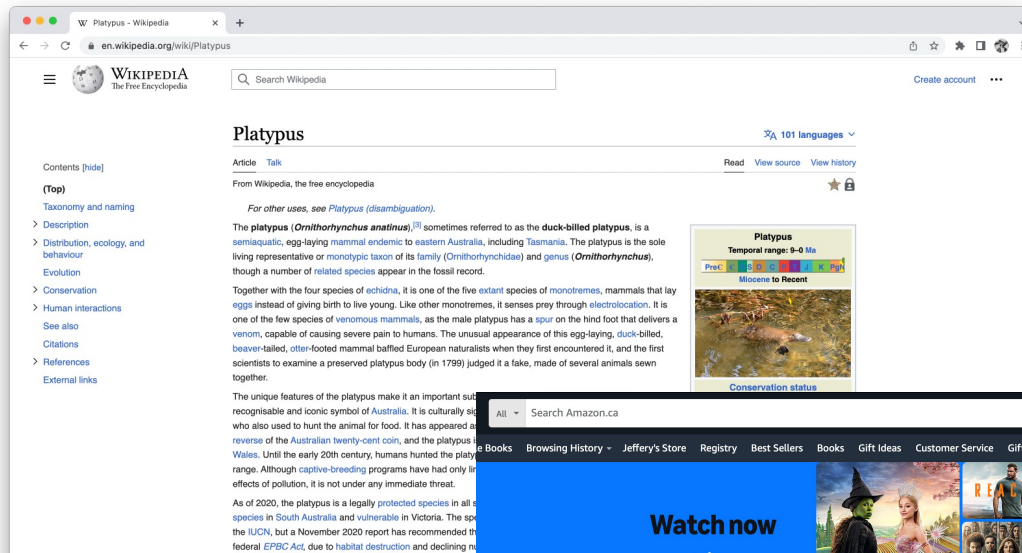
Terminal applications dominated for decades. Console applications tend to be small, fast, and resource friendly. They are still popular within specific domains e.g., system administration, software development. All modern operating systems include shells where these applications can execute.



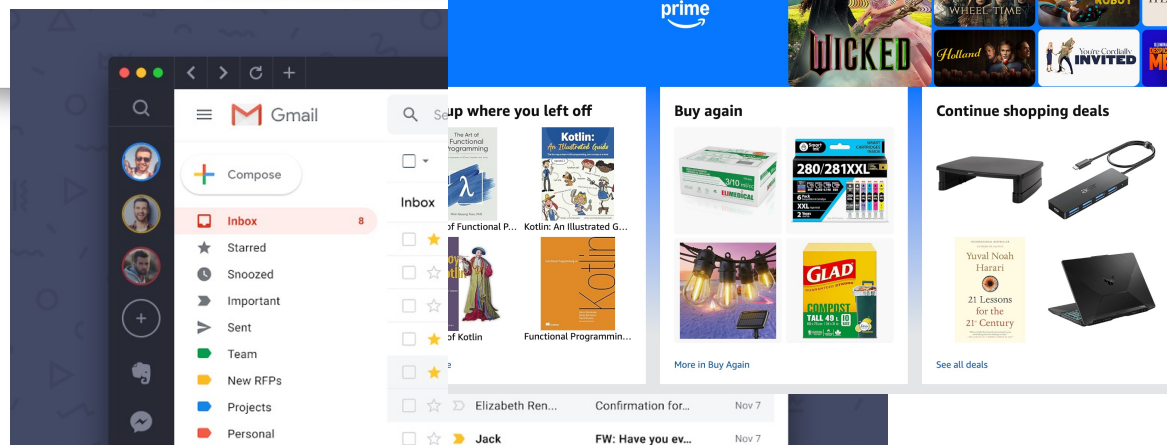


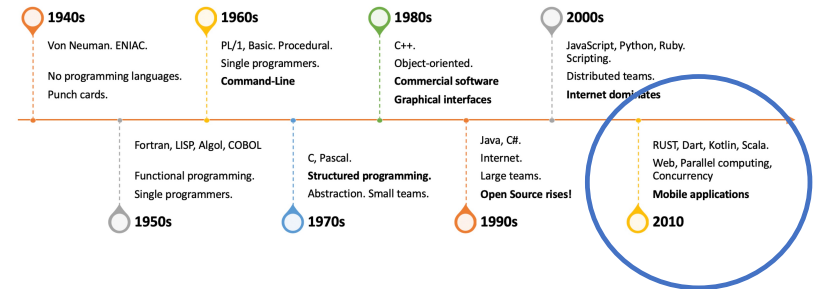
Desktop applications are still extremely popular!

They are suitable for tasks that require a lot of screen real-estate, or that need to be used for long periods of time. They also support mouse/trackpad and other input devices which makes them suitable for precise input as well.



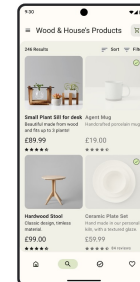
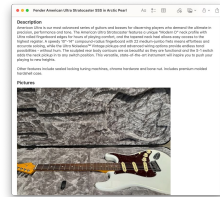
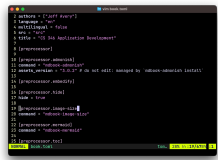
We cannot overstate the impact of the **internet & www** on software development and distribution.





Mobile applications are usually designed for casual, on-the-go use. They tend to favor content consumption over creation, where touch-input isn't a significant restriction. Otherwise, mobile and desktop paradigms are similar.

Present Day



1950s - 1960s

1970s – 1980s

1990s– 2000s

2010s – 2020s

Console

C++ (Python, C)

Imperative

Desktop

C++, Visual Basic

Object-Oriented

Graphics

Databases

Web

JS, HTML, CSS

Functional

Databases

Networking

Concurrency

Mobile

Kotlin, Swift

OO & Functional

Graphics

Databases

Networking

Concurrency

A “modern application”?

- Multiple application styles
 - Console, Desktop, Mobile, Web
- Multiple operating systems
 - Desktop: Windows, macOS, Linux
 - Mobile: iOS, Android
 - Web: Chrome, Firefox, Safari, Edge
- Network capable
 - Internet, local network, Bluetooth
 - Works with online services
- Graphical
 - 2D, 3D, animations
- Rich interaction
 - Keyboard, mouse, tablet, touch



The Spotify desktop, mobile, and tablet apps. Via Spotify


*How can we tackle something
this complex? It seems like a lot.*

Software is written by teams

Most software is developed by teams: small (<5) to very large (100s).

- Team roles:

- Team lead (technical lead)
- Product manager (non-technical)
- Project manager (non-technical)
- Architect x n_1
- Developers x n_2
- Quality analyst x n_3
- Writer x n_4



In your career, you may do many of these things! Non-technical doesn't mean uneducated about CS, it just means your primary responsibility may not be coding. e.g., developer advocates, Kotlin documentation writers.

- **Communication skills are critical** to being successful.

- You will need to coordinate your work with other team members!
- You will often need to communicate with customers, or non-technical people.

Successful teams follow “best practices”

- Software engineering practices provide better results.
 - Project management practices: tracking the work that you are doing and are planning on doing over time.
 - Team practices: discussing and working through challenges together (not going solo).
 - Development practices: ways to produce better designs, more flexible code, higher-quality output.
- As a developer you want to avoid:
 - Developing something with the expectation that you will replace it later.
 - Rewriting someone else’s code because you know a “better way to do it”.
 - Surprising your team (“I know I was supposed to get to this two days ago, but...”)
- What do we want?
 - Careful, deliberate decisions, made by a team.
 - The practiced use of best-practices that will help you produce “better” software.

Building

Every applic

- **Asset:** If it
 - **Liability:** I
- easy to ov

GTA 6 could have a budget ranging between \$1 billion to \$2 billion, per a report from Insider Gaming. This would make GTA 6 the most expensive game to ever be made. Development of the game likely started after the release of GTA 5 in 2013 and was ramped up following the release of Red Dead Redemption 2 in 2018. A game that has been in development as long as GTA 6, likely including a new GTA Online and backed by a big budget for voice acting and music licensing, would cost a lot to warrant an \$80 price.

If you're interested in creating an app, you've probably considered what it will cost.

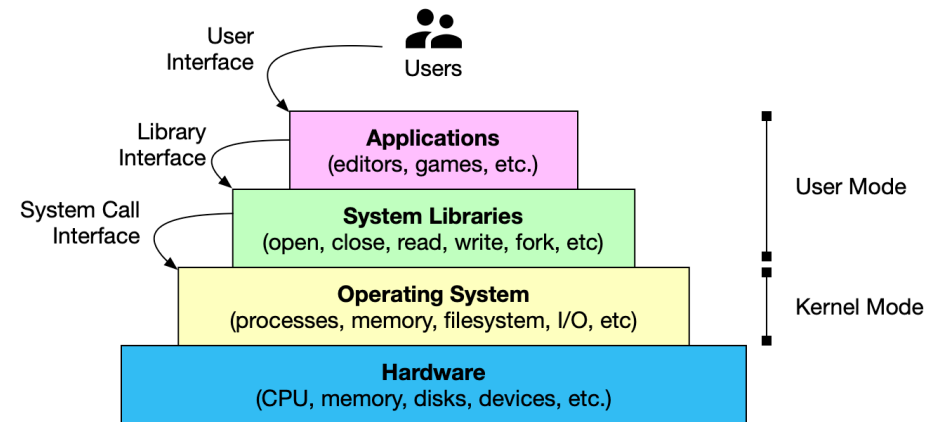
Most industry reports say the cost of building a **mobile app** varies from **\$30,000 to \$250,000**, with the average cost coming in at **\$171,450**. Nonetheless, these numbers are only an estimate since the actual cost can change quite a bit depending on the job.

Libraries == code reuse

Recognize that libraries are (usually) highly optimized, well-tested software!

Don't create new functionality when you can reuse someone else's work.

- **System libraries** expose core functionality to your application. Part of the OS; “low level”. e.g., raw graphics, network.
- **User libraries** sit “above” system libraries to provide extra capabilities, or a better abstraction. e.g., 3D graphics, database.



Applications leverage the capabilities of the underlying operating system, often through system or user libraries.

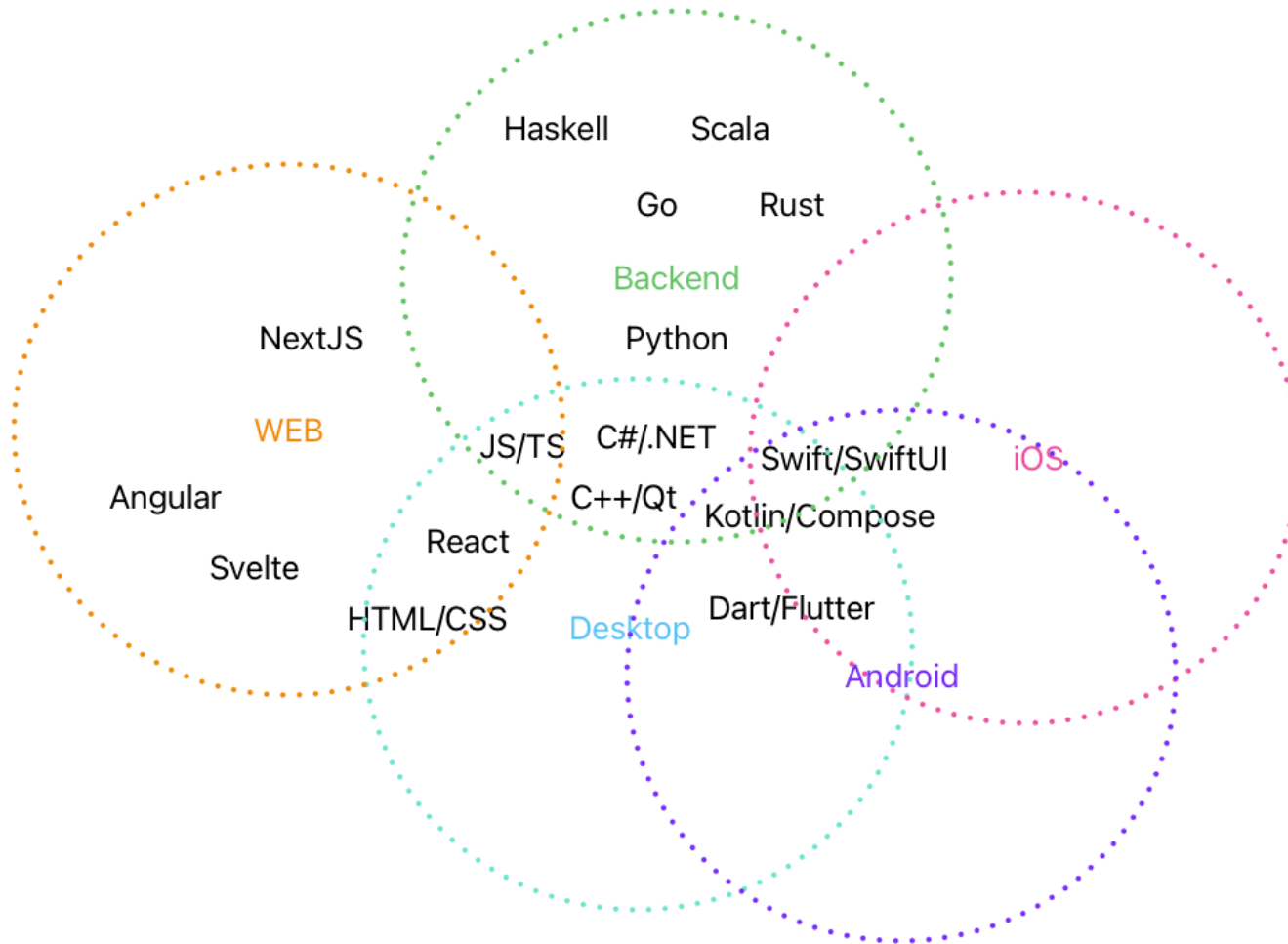
A well-supported “tech stack” is critical

“[A technology stack aka “tech stack”] refers to a set of tools, programming languages, and technologies that work together to build digital products or solutions such as websites, mobile, and web apps”. - <https://fullscale.io>

Your tech stack includes

- Programming language
- System + user libraries available
- Your target operating system

Choice of technology stack will determine what capabilities you can leverage in your application. e.g., building a graphical application in Rust *sounds great*, until you realize that there is limited UI support.



No tech stack does everything. You need to find the best “fit” for your situation.

*So this is a lot. What specifically
are we doing in this course?*

Course description

CS 346 Application Development
LAB, LEC, TST 0.50

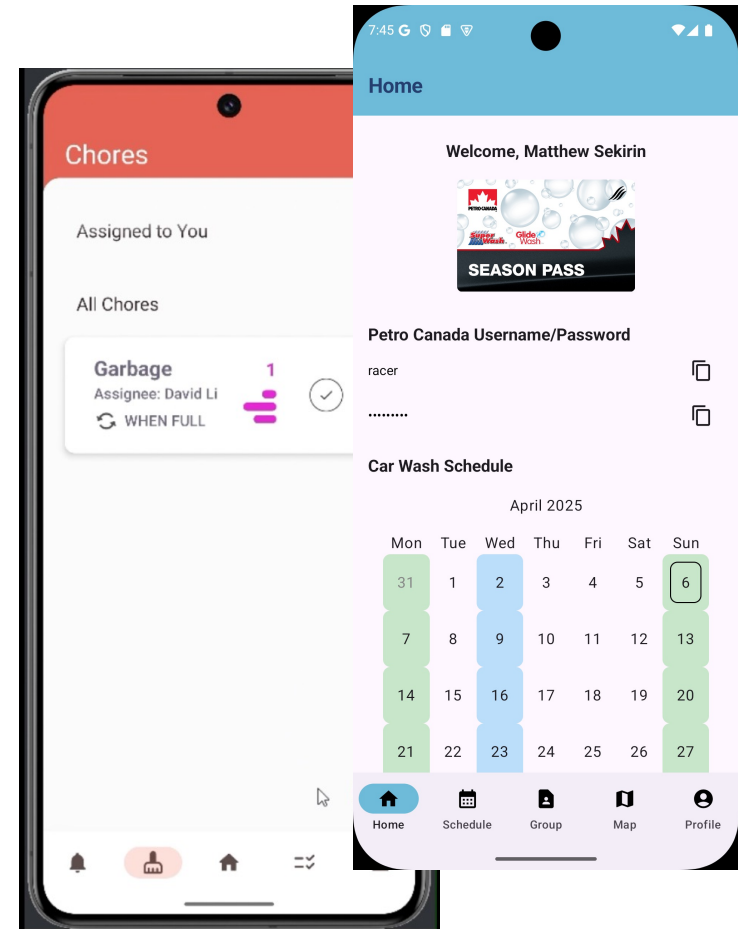
Introduction to **full-stack application design and development**. Students will work in **project teams** to design and build complete, working applications and services using standard tools. Topics include **best practices** in design, development, testing, and deployment.

Prerequisites: CS 246; Computer Science students only.

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

We build applications!

- Form teams of 4 people
- Design and develop an application
 - You choose what application to build
 - Focus on mobile or desktop applications
 - Basic requirements for each platform (we provide).
 - Advanced features for your application (that you propose).
- Bi-weekly team submissions
 - Documentation, software releases
 - Demo to your TA and get feedback.
- Get help from us in-class!
 - Lectures + LAB sessions



[See the Project Gallery](#)

Our tech stack is meant for this!

- Targets

- Desktop: Windows, macOS, Linux
- Mobile: Android (iOS)
- Web: (WASM)



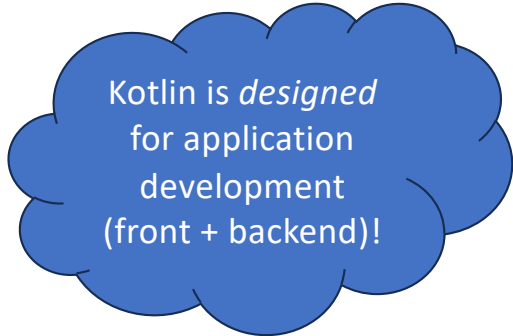
Brackets means “talk to Jeff before committing to this”

- Kotlin programming language

- Modern language, with features desirable for application development
- One of very few choices if we want to support broad-targets.

- Libraries

- **Compose**: user-interface library for desktop/mobile/web.
- **Exposed**: library for working with SQL databases e.g., SQLite.
- **Ktor**: Kotlin-first networking library for web services.
- **Koin**: dependency injection library.



Kotlin is *designed*
for application
development
(front + backend)!

What will you learn?

- Development
 - Application styles, Design, Testing – broadly applicable approaches.
 - Learn an interesting and useful tech stack/modern programming language.
 - Learn mobile development, graphical user interfaces, database connectivity.
 - Apply relevant design practices i.e., design principles and patterns.
- Best practices
 - Build software the way that you would in industry. Software development practices e.g., code branching/merges, issue tracking, unit testing, software releases.
 - Just like real-life, you will demo your progress!
- Teamwork
 - Collaborate and coordinate work across a development team.

Choose to do
“everything”
or focus on
one area.



Course Website

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

○ CS 346 Fall 2025

[Course Info](#) [Reference](#) [About](#)

Syllabus



- Overview
- Text & Materials
- Assessment
- Policies
- Contacts

Lectures



- Sections
- Schedule
- Agenda**
- Quizzes

Project



- Overview
- Requirements
- Project Teams

[Course Info](#) > [Lectures](#) > [Agenda](#)

Agenda

What's covered each week. This page will be updated as the term progresses.

Week 01: Introduction

Wed Lecture

- [Introduction](#)
- [Agile & SDLC](#)
- [Project guidelines](#) & [requirements](#)

Fri Lab

- [Teamwork](#)
- [Setup GitLab](#)


Schedule

Lectures: in-class presentations

Labs: demos/examples, and free time

Course Info > Lectures > Schedule

Schedule

Week	Date	Type	Details
Week 01: Introduction	Wed Sept 3	LEC	Introduction. Agile & SDLC.
	Fri Sept 5	LAB	Teamwork. Setup GitLab.
		DUE	n/a
Week 02: Kotlin	Wed Sept 10	LEC	Learning Kotlin. Documentation.
	Fri Sept 12	LAB	Install the toolchain. Setup the repository.
		DUE	Setup due Fri @ 6:00 PM 

Assessment

You are graded primarily based on your team project.

You will propose, design, and iteratively deliver a project.

Team activities

- Everyone participates!
- Deadlines every 2 weeks with a demo to your TA.
- Final release

Individual Grade (30%)

Component	What it addresses	Grade
Quizzes	Understanding lecture content. 5 quizzes.	5 x 4% = 20%
Participation	Level of contributions to the project.	10%

Team Grade (70%)

Item	What it addresses	Grade
Proposal	Project identified, requirements captured.	10%
Sprint 1: Architecture	Features complete; process followed.	5%
Sprint 2: User Interface	Features complete; process followed.	5%
Sprint 3: Databases	Features complete; process followed.	5%
Sprint 4: Final Review	Features complete; process followed.	5%
Software Release	Completed project, including documentation.	40%

Policies

Group Participation

- **You must form teams by the end of week 2.** We will help, but you are responsible.
- **You must participate during the term!** We can remove you from the course or adjust your final grade if you fail to participate.
- **You cannot take this course while on a (remote) work term.** In-person only.
- **You must attend every demo!** You lose significant marks if you skip demos.

Code “sharing”

- You are allowed to use someone else’s code (up to 25 lines) with appropriate citation.
- You cannot use projects from previous terms or other courses.

Review the course website + project requirements!

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