Introduction

CS 346 Application Development

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

Contents

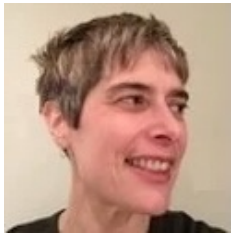
About the Course

Course Staff

Instructor: Dr. Jeff Avery

- MC 6461 (by appointment)
- jeffery.avery@uwaterloo.ca

ISC: Caroline Kierstead



- MC 4008 (by appointment)
- ctkierst@uwaterloo.ca

Teaching Assistants

- TBD

See course website for contact information [1].

Course Website

The course website has everything you need, including this lecture.

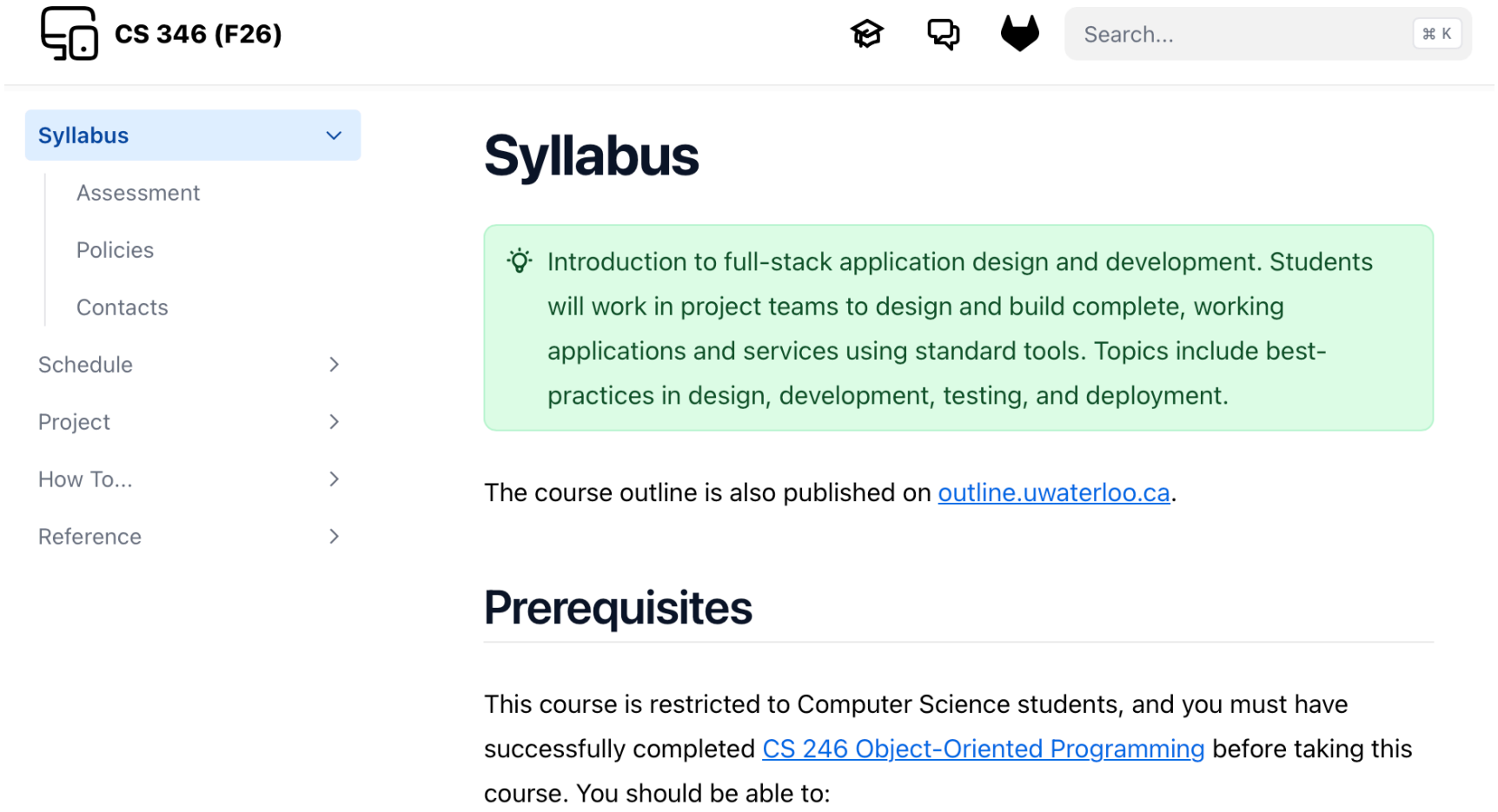


Figure 1: https://student.cs.uwaterloo.ca/~cs346

Course Syllabus

Catalog Description

From the course calendar:

“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.” [2]

Main themes:

- Designing and building complex applications.
- Learning best-practices for software development.
- Practice working on a team.

Catalog Description

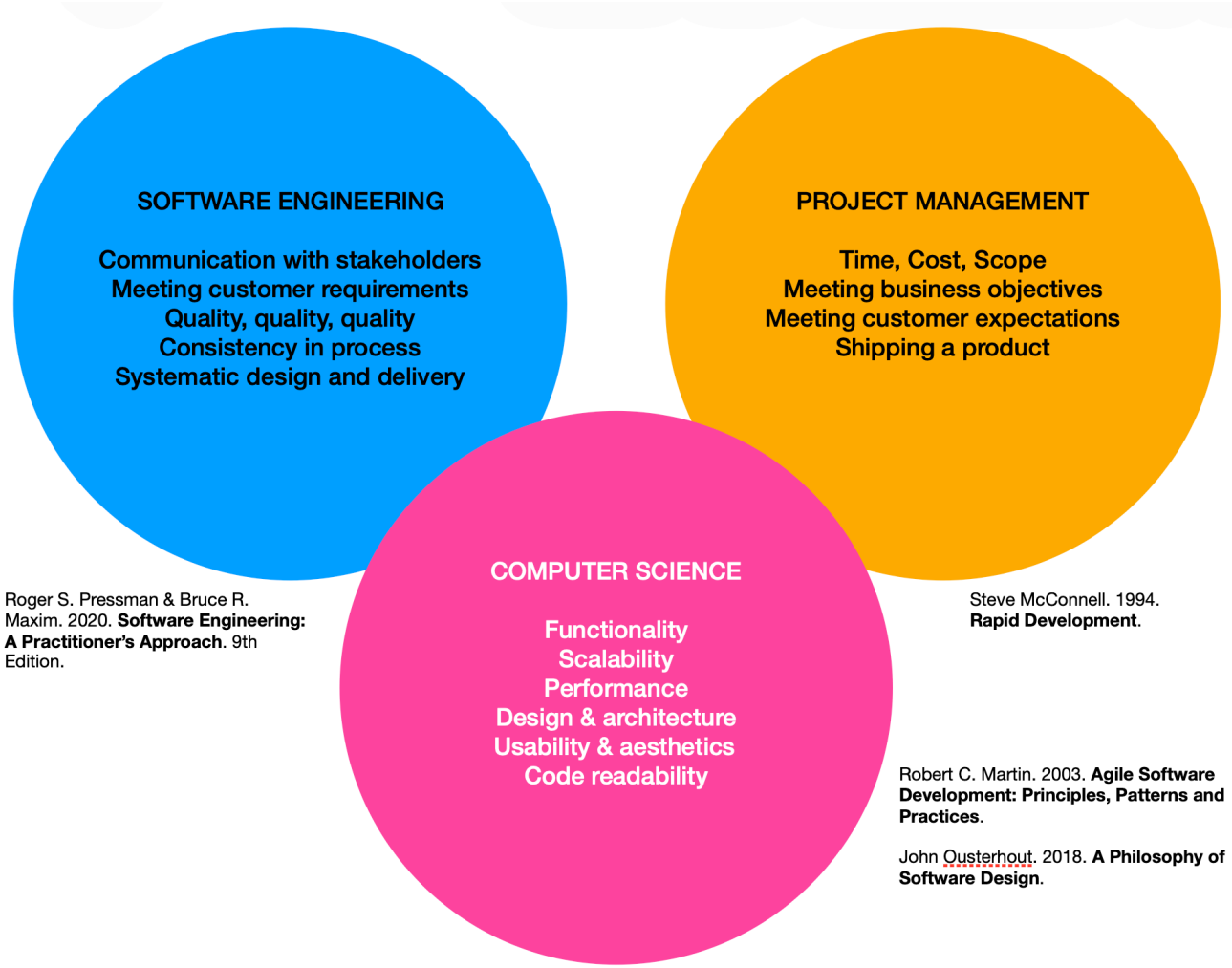


Figure 2: Although this is a Computer Science course, SE and Project Management topics will also be discussed. The books listed are suggestions, not required reading.

What will you be doing?

You will spend most of your time and effort working on a software project:

1. You will join a project team
   - four people per team, same section
   - you and your team *choose* what to design + implement

2. You will learn our tech stack, toolchain, methods and practices.
   - most material delivered through online lectures.
   - classtime is for working on your project, watching instructor demos, getting help.
   - your instructor and your TA will be available to help you out.

3. You will deliver software releases through the term (four in total).
   - demonstrate features that you have designed and implemented.
   - get constructive feedback from your TA and instructor, and address it.
   - build high-quality software releases that can be installed and used!

Course Structure

Introduction

Getting started in the course! Forming project teams, managing projects and product requirements. Common development practices.

Week	Lectures	Wed	Fri
Week 01	Welcome, Teamwork, SDLC, Requirements	in-class lecture	in-class lecture
Week 02	Kotlin, Build Systems, Testing	working	check-in
Week 03	Architecture, Design	demo 1	demo 1

Front-End

Client-side development; desktop and mobile applications. Building user interfaces.

Week	Lectures	Wed	Fri
Week 04	Desktop Applications, Compose Multiplatform	working	check-in
Week 05	Android, Navigation	working	check-in
Week 06	iOS, Kotlin Multiplatform	demo 2	demo 2

Figure 3: The course schedule shows the lecture topics and themes for each week.

Course Structure

Outside-Class

- Lectures are pre-recorded and posted online.
  - Review required lectures each week, and complete quizzes.
  - Watch optional lectures (no quizzes).
- Meet with your team regularly.
  - Planning, design, coordination.
  - Working on software releases.

In-Class (three-week cycles)

- Two weeks of working on your project
  - Wed working session: instructor present to help (optional).
  - Fri demo: demo with your TA & instructor present (graded).
- One week of demos
  - A scheduled 20-minute demo with your TA.
  - Remaining time that week is “free”.

Assessment

Personal (20%)

Component	What is it?	Weight
Section Tests	Lectures (4 x 5%)	20%

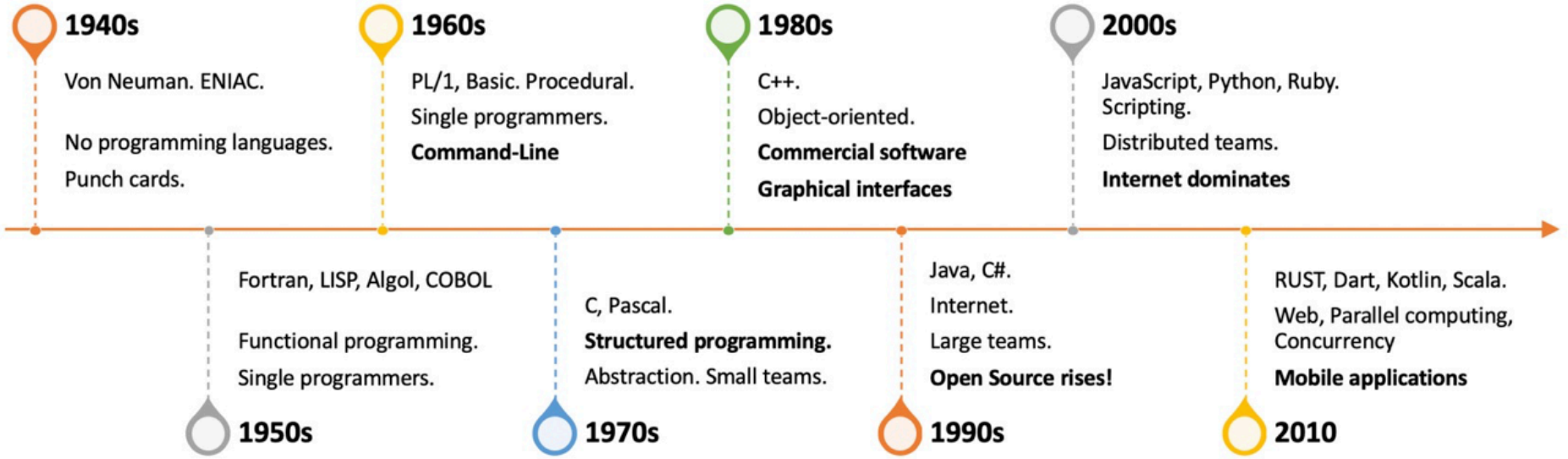
Team (80%)

Component	What is it?	Weight
Demos	Weekly demos (7 x 2%)	14%
Presentation 1	Proposal	10%
Presentation 2	Iteration + software release	15%
Presentation 3	Iteration + software release	15%
Presentation 4	Iteration + software release	20%
Docs	User guide + video	6%

NOTE: Attendance is expected for all team deliverables! If you fail to show up, or don’t participate, you get a zero for that component.

Application Development

What is Application Development?



How has software development changed?

- Complexity: our software does more than software of last decade.
- Distributed: Increasing dependency on networking, remote data.
- Platforms: textual -> graphical desktop -> web -> mobile.

Modern Applications

Application development is about designing for end-users.

- Software should be useful, and enjoyable to use.
- It can be impactful and potentially used by millions of users.

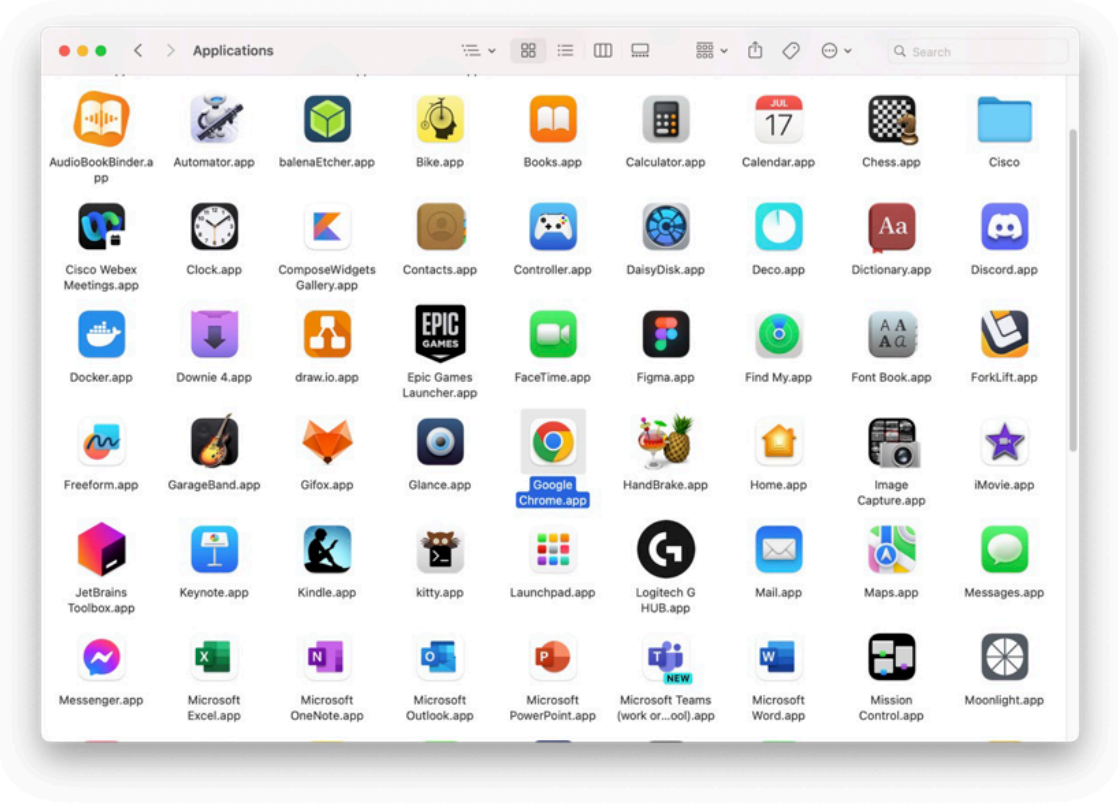


Figure 4: My applications folder, with dozens of applications.

Modern Applications

Applications often run on multiple platforms

- Desktop (Mac, Windows, Linux)
- Mobile (iOS , Android)
- Web (Chrome, Safari, Firefox)

Each platform has distinctive features

- Desktop: windowing, undo-redo, copy-paste.
- Mobile : device rotation, multi-touch gestures.
- Web: deep-linking to content, privacy, browser-compatibility.

Integration with other systems is necessary

- Local & remote data storage.
- Networking & access to remote services.
- Privacy & security features, concern for user data.

What is a Full-Stack Application?

Historical: “front-end” (UI) and “back-end” (computation).

This aligns with the way that we deploy software:

- Front-end client (local machine)
- Back-end database/server (remote machine)

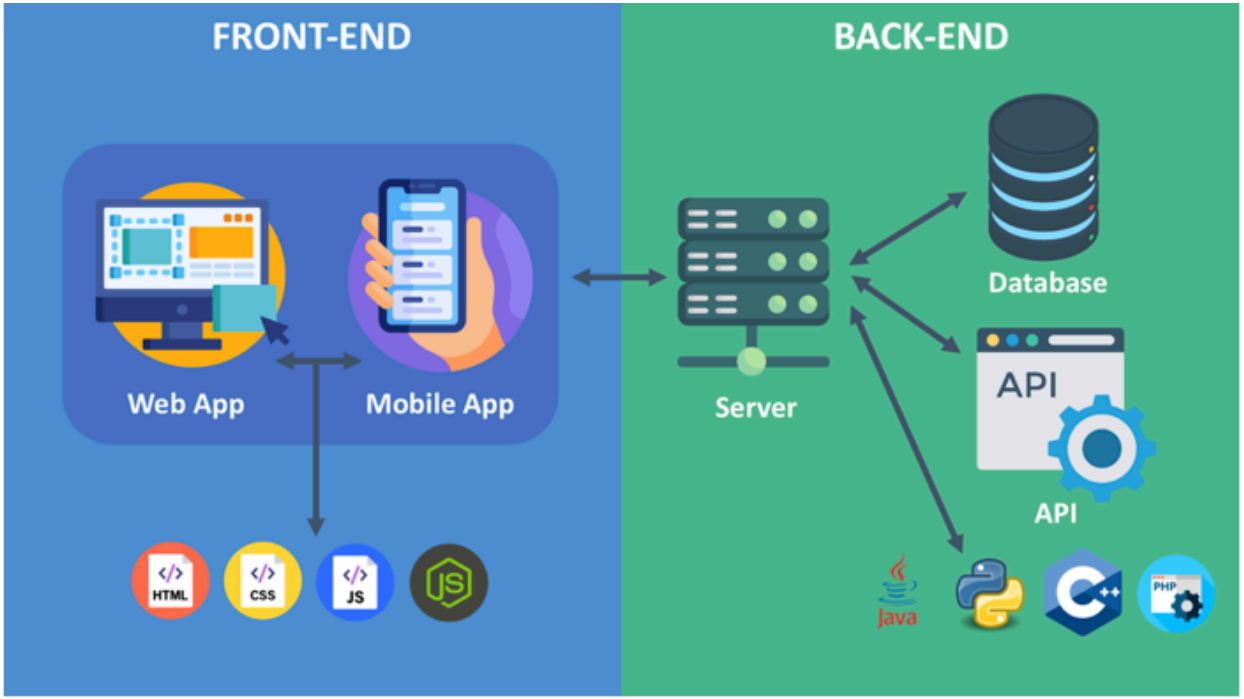


Figure 5: “Full-stack” logic is split between front and back-end.

What is a Full-Stack Application?

Example: Things

Things is an award-winning TODO application for the Apple ecosystem.

Client runs on every device/platform.

- Desktop for serious planning.
- Mobile for checking items.
- Watch for quick reminders.

“Things Cloud” hosts user data.

- Synchronizes data between devices.
- Premium subscription service.

Great example of clean, simple design that was very well-executed. They understand their users.

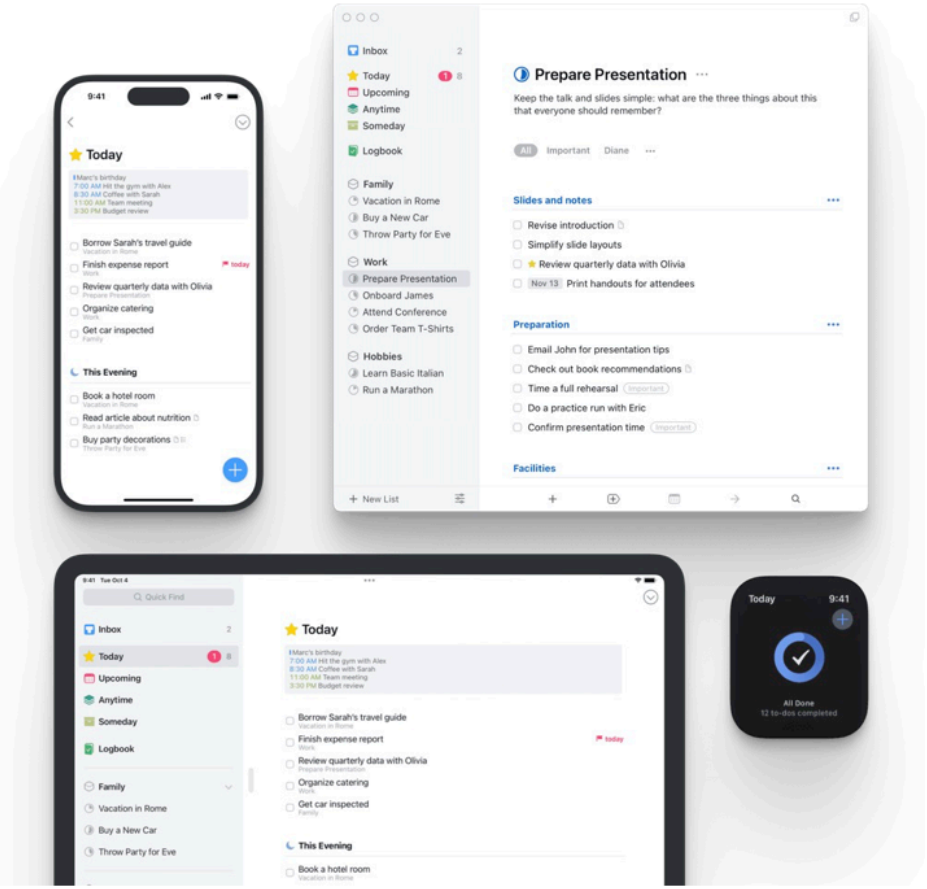


Figure 6: A cross–platform story makes an application more useful and compelling.

Modern Development

Programming language(s) + libraries are required.

- We need to develop for each target platform's native features.
- Additional requirements for graphics, database connectivity, networking, concurrency, security - not provided by language.

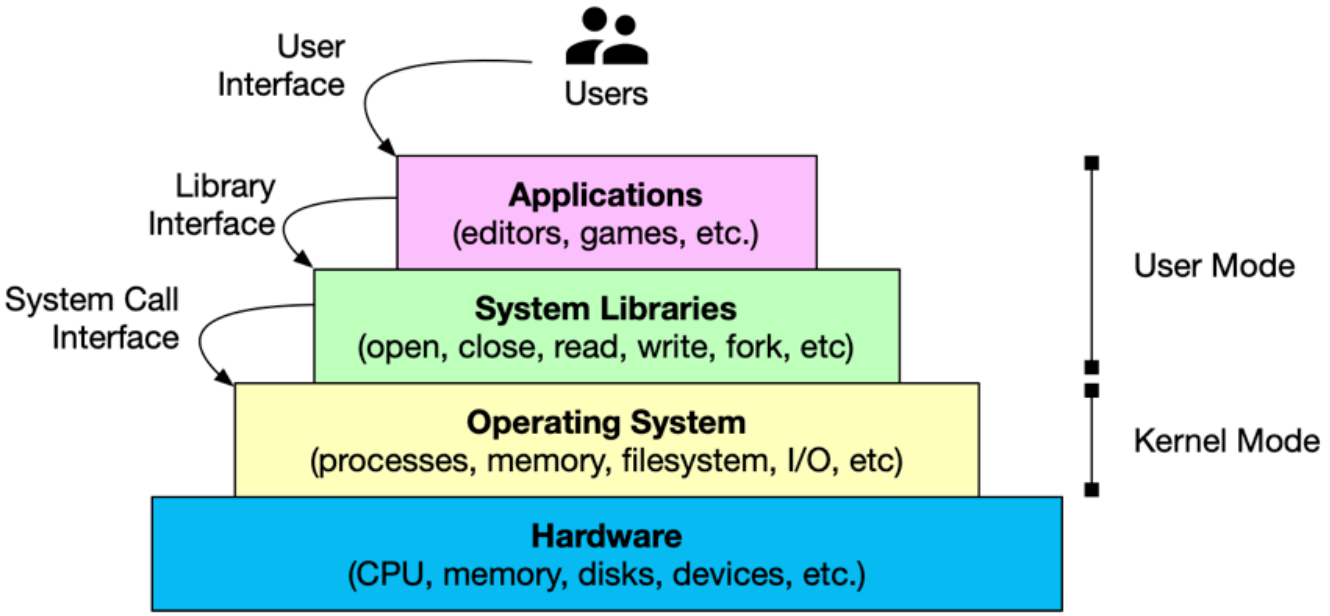


Figure 7: User libraries leverage underlying OS functionality.

Modern Development

We need modern language features:

- Automatic memory management.
- Strong, static type systems.
- Functional & OO paradigms supported.
- High portability & support for multiple platforms.

We also need an ecosystem:

- Tooling and library support for missing functionality.
- Community support, including books, forums, training materials.

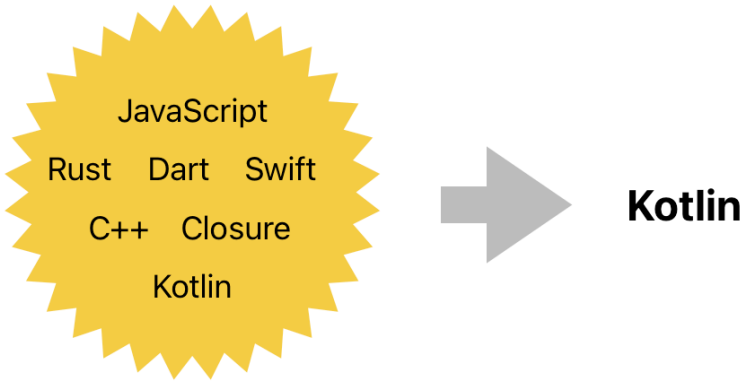


Figure 8: There are many choices for a tech stack. Kotlin is popular, and offers great tooling and multi-platform support.

Modern Development

> “Do I still need to learn to code when GenAI can do it for me?”

I’m sure many of you have used GenAI on your courses, for assignments.

However, coding was never the hardest part of software development.

- Software projects tends to struggle because of human-issues: understanding requirements, communicating, making difficult decisions with customers.

Architecture and design become increasingly important as you develop larger and more complex systems.

- Is your AI solution scalable and maintainable? How extensible is it?
- If you rely exclusively on AI, what do you do when it fails to work?

Advice: use it to augment, not replace, your ability to think and problem solve.

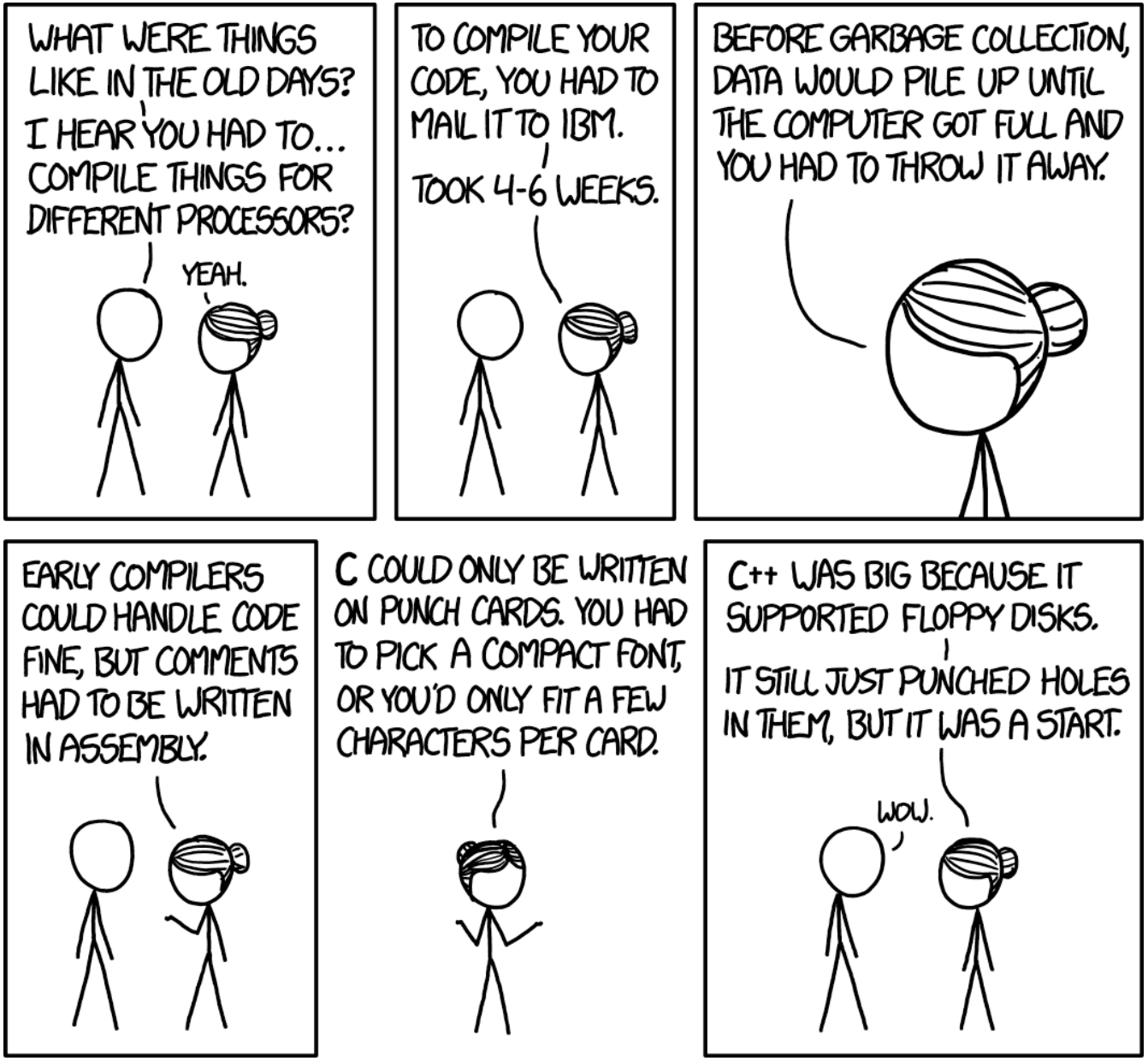


Figure 9: https://xkcd.com/1755

Bibliography

[1] J. Avery, “CS 346 Application Development.” [Online]. Available: https://student.cs.uwaterloo.ca/~cs346

[2] J. Avery, “CS 346 Course Calendar Entry.” [Online]. Available: https://uwaterloo.ca/academic-calendar/undergraduate-studies/catalog#/courses/rkSgQ5NmYh