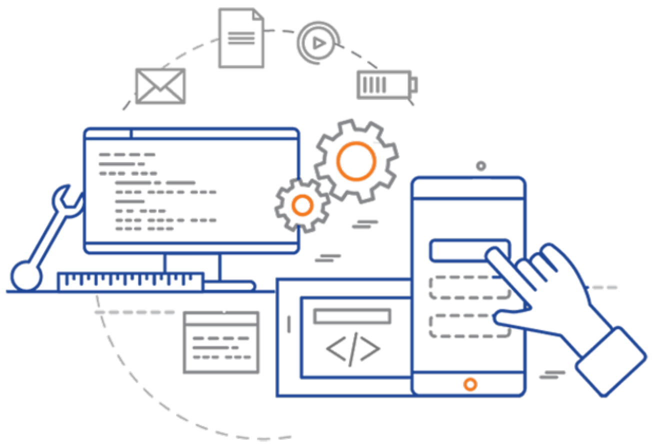


CS 398: Application Development

Week 01 Video: Introduction

What is software?

Software is a collection of instructions [a computer program] that tells a computer how to work. This is in contrast to hardware, from which the system is built and which actually performs the work.

— Wikipedia, 2021.

Two types of software:

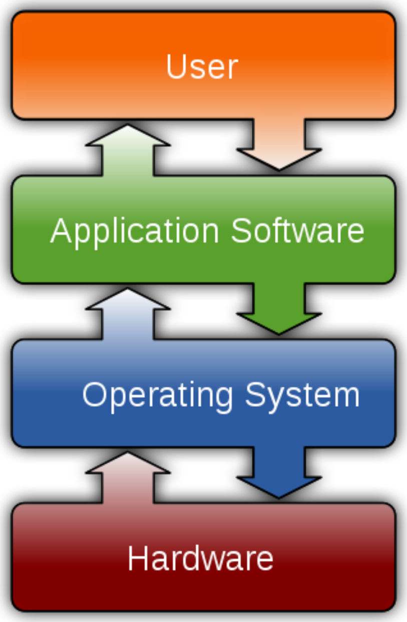
1. **System software** is software that helps your computer function effectively. It includes software designed to assist other systems. e.g. Linux kernel, a printer driver, Apache web server.

2. **Application software** (also *application* or *app*) is designed to help people perform tasks. e.g. MS Word, Google Chrome, Fortnight, GCC, Apple Keynote, Calculator, Notepad. Applications tend to be *interactive*, and targeted towards end-users.

Although we will certainly leverage system software, we're concerned about designing and building application software in this course.

Full-stack simply means that we want to build software that can leverage remote services.

Every computer system consists of hundreds or thousands of very specialized, small-to-medium sized programs working in harmony to deliver a functioning system.



Users interact with appliation software, which in turn relies on services provided by system software/services and the underlying operating system.

Application software doesn't interact with hardware directly, but relies on the OS for any low-level services. This includes things like graphics, and networking.

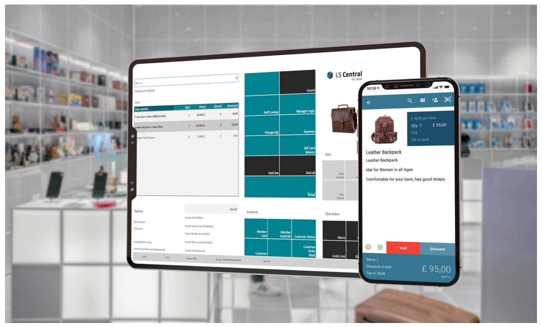
From now on, when we use the word “software”, we’re referring to “application software”.

Software can be complex

These are all applications, developed by teams of tens or *hundreds* of people. They're also incredibly complex pieces of software. How do we build these?

Gmail

iCloud



Software Engineering + **People Skills** + **Artistry and Creativity** + **Craftsmanship and Competence** + **Domain Knowledge**

Software Engineering

- Requirements
- Design
- Coding
- Testing
- Deployment
- Documentation
- Maintenance
- Reengineering
- Configuration
- Management
- Processes
- Principles
- Patterns
- Practices

People Skills

- Communication
- Social skills
- Teamwork
- Group dynamics
- Leadership
- Negotiation
- Systems Thinking

Artistry and Creativity

- Solving Problems
- Naming
- Constructive creativity
- Destructive creativity
- Pragmatism
- Intuition
- Innovation
- Experimentation
- Complexity Thinking
- Vision

Craftsmanship and Competence

- Clean Coding
- Teaching Yourself
- Lifelong Learning
- Business Knowledge
- Technical Knowledge
- Quality Thinking
- Time Management
- Risk Management
- Passion
- Discipline
- Culture
- Values
- Ethics

Domain Knowledge

- Life Critical Products
- Security Models
- User Roles
- Terminology
- Archetype
- Standards
- Legislation
- Healthcare Knowledge
- Integration
- System Topology

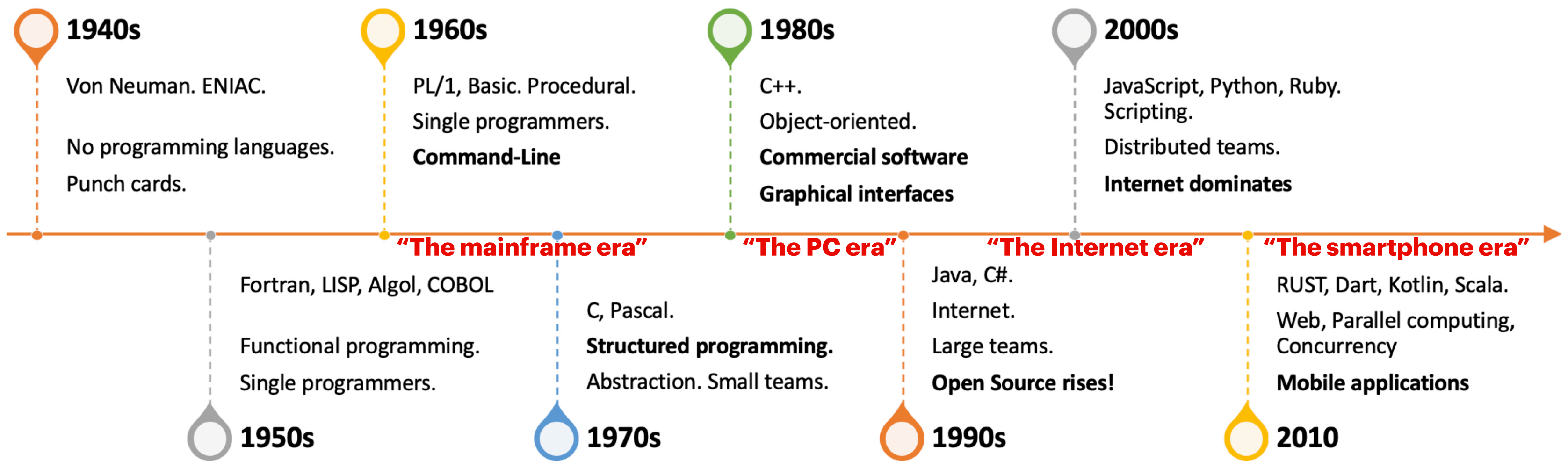
One of the reasons that we need a software process, and *teams of people contributing*.

Software isn't *usually* written by a rogue developer sitting in a dark basement eating pizza.

Why does software matter?

- **Software has become important in virtually every aspect of our lives**. Although software was once the domain of experts, this is no longer the case, and software is routinely used by consumers and non-experts.
- **Software is now embedded inside everything** from consumer electronics to medical devices to autonomous vehicles. Design is a critical activity to ensure that we address requirements and build the "right thing".
- Software itself has become **increasingly complex to build and maintain** as we handle more and more difficult problems.
- If the software fails, people and major enterprises can experience anything from minor inconvenience to catastrophic consequences. **The quality and reliability of software systems is increasingly important**. Software needs to be high-quality, safe and reliable.
- As its user base and time in use increase, demands for adaptation and enhancement will also grow. **Software needs to be designed to be adaptable and scalable**.

The History of Software



Software has evolved to meet the evolving capabilities of our hardware, and changing demands from consumers. The discipline of software development has also evolved over time.

Of particular interest: the development of console applications in the 60s and 70s, graphical applications in the 1980s and later, and mobile applications starting in the early 2000s.

Early Computing (1950s-1980s)

Research in programming languages led to the rise of procedural and then structured programming. Object-oriented programming was invented with Smalltalk in the 1960s but wasn't very widely used yet.

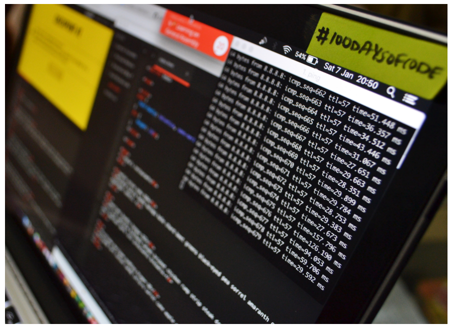
Unix was invented in 1969 by Ken Thompson and Dennis Ritchie. It ran on a PDP-7 minicomputer.

The 1970s saw the creation of many foundational technologies: C, Pascal, ADA. Commercial software started to appear towards the end of the 1970s.

Software was primarily textual! Graphical interfaces appeared in the 1970s but were't really popular until the early 1980s.

Console-based interfaces still exist....

Terminal session on a mainframe.


Modern shell

The PC Era (1983-2000)

Computers were finally cheap enough that everyone could own one. This was the "PC" era when computers were sold to individuals (vs. universities or large companies).

Single-user systems needed to be networked, which led to innovations in **networking**, distributed systems, security.

Keyboard and mouse became standard, using point-and-click interfaces. We finally had affordable **computer graphics**!

PC operating systems — MS Windows, macOS — were developed in this era. Linux arose in the late 1990s and made Unix mainstream.

Object-oriented programming finally caught on commercially with C++. Java, C# and other languages quickly followed.

Internet Era (1995-present)

In the 2000s, we saw the rise of always-on, networked, portable computing.

The Internet and world-wide web changed how consumers use and interact with software, but the resulting network and infrastructure changes had a *significant* impact.

There was a corresponding shift towards performing computationally expensive tasks remotely.

- **Remote services**, supporting execution and delivery of results asynchronously.
- Shift towards **web applications** in many domains e.g. Gmail.

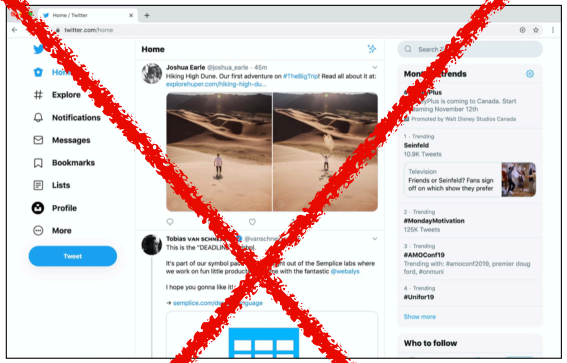
The modern **smartphone era** introduced **smartphones**, tablets, smartwatches.

- People expect to carry computing devices with them, and access **online data**!
- This was one of the largest shifts in technology that we've ever seen. It's significance cannot be understated.

The Modern Era?

Desktop applications
- Multitasking
- Applications running in windows
- Resizable, movable
- Keyboard, mouse/trackpad
- Copy/paste, undo/redo
- Sound, graphics, animation

Windows, macOS, Linux



~~Web applications~~
- ~~Multitasking~~
- ~~Single window~~
- ~~Resizable, movable~~
- ~~Keyboard, mouse/trackpad~~
- Copy/paste, undo/redo??
- Sound?

Chrome, Firefox

Mobile applications
- Single application focus
- Applications run full-screen
- Single window
- Multitouch
- Copy/paste, undo/redo
- Sound, graphics, animation

iOS vs. Android

The Modern Era?

What are the characteristics of our modern era*?

- **People use multiple devices**. There are more smartphones than people in the world, and in developed nations, most people own a smartphone + computer. Many of us also own tablets, smartwatches, and maybe a second computer.
- **Our data resides "in the cloud"**. We commonly share personal information online, and in many cases, the cloud" is our primary storage for that data. e.g. Gmail storing all person email; Google Photos storing photo albums to share with family.
- **We love software ecosystems**. Apple users buy Apple devices in-part because they work well together. Microsoft includes Android functionality in Windows, and so on. Companies strive to offer "everything" to keep you in their ecosystem.
- **Users expect "good design"**. Good design used to be rare. Now, it's our expectation, even with "cheap" software.

* This list is highly subjective of course, and based on the priorities of the person suggesting the list. For the purposes of the course, we care about applications, so those are the features I'll call out.

What does it mean to be a developer?

How do these trends affect me?

- **Cross-platform support is essential.** You need to be able to support your software running on different operating systems and different environments. It's common for application vendors to produce versions of their software for both Windows and macOS for desktop, and both Android and iOS for mobile.
- **Software ecosystems means that every major platform is proprietary.** Companies don't have much interest in helping you develop for their competitors. e.g. Swift is really viable for Apple operating systems. Microsoft doesn't really promote C# outside their own platform*. With very few exceptions, it is difficult to write code once and have it run on multiple platforms.
- **Software is complex**. We are solving extremely difficult problems. You need to become a domain expert.
- **You require external frameworks and libraries.** Companies cannot design and produce all their own software. Software is complex enough that best-practice is to use well-known libraries when you can. These are typically very well designed, and thoroughly tested. e.g. Boost for C++, SwiftUI for Apple, Jetpack Compose for Android.

* Yes I realize that companies commonly Open Source these technologies. If you look carefully though, they always keep back the most critical parts of these systems. Android is famously Open Source, but all of the Google services are closed source and not available under the same licensing agreements.

Kotlin Language Features

- Class-based, object-oriented language. ★
- Statically typed with type inference.
- Encourages immutable data structures, closed inheritance model.
- Default arguments, variable arg lists, NULL handling.
- Excellent async support using coroutines (lightweight threads). ★
- Full interoperability with Java source code and libraries.

Multiplatform Deployment

- Desktop: Windows, Linux, Mac. ★
- Mobile: Android (JVM) and iOS (native). ★
- Web: native web services, or transpile to JS.