Software Explained: From Code to Career

Developed by Mwenda E. Njagi at

InsightHub Analysis Program at GitHub

(INSAPROG)

Link: https//Github.com/MwendaKE/InsightHub/Re Papers/ How Software is Built: The Insider's Guide to a Life in Code.pdf

Date: 24th September, 2025

Abstract: This document is a simple guide to software engineering and development. I will explain the difference between coding and engineering, the entire process of building software, the best languages for different tasks, and the legal rules (licenses) that govern it. I will also cover the journey to becoming a great developer and what the future holds with AI.

1. What's the Difference? Coding, Programming, Engineering

People use these words interchangeably, but there are subtle differences.

- **Coding**: This is the simplest term. It means writing instructions in a language the computer understands (a programming language). It's like writing words in a sentence.

- ❖ **Programming**: This is a broader term than coding. It includes coding, but also involves thinking about the logic, solving the problem, and testing the code. It's like writing a full paragraph with a clear meaning.
- ❖ **Software Development**: This is the entire process of creating a software application. It includes programming, but also designing how it will look, testing it, and releasing it.
- ❖ **Software Engineering**: This is the most professional and structured approach. It applies engineering principles to software creation. It emphasizes planning, teamwork, reliability, and building software that is safe, scalable, and can be maintained for years. Think of it as the difference between building a treehouse (coding) and building a skyscraper (software engineering).

Which is "**Super**"? For building small, personal projects, programming is sufficient. For building critical, large-scale systems used by millions (like banking apps or airplane control systems), Software Engineering is superior because it ensures safety and reliability.

2. Types of Software Development

Developers often specialize in a specific area:

- Desktop Programming: Creating software that runs directly on your computer (Windows, Mac, Linux). Examples: Microsoft Word, Photoshop, VLC Media Player.
- Web Development: Building websites and web applications you access through a browser like Chrome or Firefox. This is split into:
  - ✓ Front-End: The part you see and interact with (buttons, colors, layouts). Languages: HTML, CSS, JavaScript.
  - ✓ Back-End: The hidden engine that runs on a server, manages data, and does the logic. Languages: Python, Java, C#, JavaScript (Node.js).
- Mobile Development: Creating apps for smartphones.
  - ✓ Android: Apps for Android phones. Main languages: Kotlin, Java.
  - ✓ iOS: Apps for iPhones. Main languages: Swift, Objective-C.
- Game Development: Creating video games. This requires high performance. Common languages: C++, C# (with the Unity game engine).

3. The Software Engineering Lifecycle: From Start to End

Building software is a process, not a single event. Here's how it works:

1. **Requirement Analysis**: First, you talk to the client or users. What problem should the software solve? What should it do? You write down all these needs clearly.
2. **Planning & Design:** You plan how the software will be built. You create blueprints (like an architect for a house) showing how different parts will connect.
3. I**mplementation (Coding/Programming):** This is where developers write the actual code based on the design.
4. **Testing:** The code is rigorously tested to find and fix bugs (errors). There are many types of testing:
   a) Unit Testing: Testing small, individual pieces of code.
   b) Integration Testing: Testing how different parts work together.
   c) System Testing: Testing the entire, complete application.
5. **Deployment:** The software is released to the public or the users. It's installed on their devices or made available on app stores.
6. **Maintenance:** The job isn't over after release. Software needs constant updates to fix new bugs, add new features, and stay secure. This is a continuous cycle.

4. Programming Languages and Their Uses

There is no single "best" language. Each is a tool for a specific job.

- **Python** (Data Science, AI, Web Back-End, Automation) - Simple to read and learn, has powerful free libraries.
- **JavaScript** (Web Development (Front-End and Back-End)) - The only language that runs natively in web browsers.
- **Java** (Large Enterprise Systems, Android Apps) - Very stable, secure, and runs on any device ("write once, run anywhere").
- **C#** (Windows Desktop Apps, Game Development (Unity)) - Created by Microsoft, powerful and versatile.
- **Swift** (iOS and MacOS Apps) - Modern, fast, and designed by Apple to be easy for their platform.
- **C++** (Game Engines, Operating Systems, High-Performance Software) - Very fast and gives the programmer a lot of control over the hardware.

5. Software Licenses: The Rules of Use

When you write code, you own it. A software license is a legal rulebook that tells others how they can use your code.

- **Proprietary License**: This is commercial software. The code is secret, and you only buy a license to use it. You cannot see or modify the code. Example: Microsoft Windows, Adobe Photoshop.
- **Open-Source Licenses**: The code is public and free for anyone to use, modify, and share. This is how most modern software is built. There are different types:
  - ✓ MIT License: Very simple and permissive. Anyone can use your code, even in proprietary software, as long as they give you credit. Best for beginners to use for their projects.
  - ✓ GPL License: More restrictive. If someone uses your GPL-licensed code in their project, their entire project must also be open-sourced. It forces the spirit of sharing to continue.

6. GitHub and Code Hosting

GitHub is a website (a "code hosting platform") that is essential for modern developers. Think of it as "Google Docs for code."

- What it does: It lets developers store their code, track every change, and collaborate with others without overwriting each other's work.
- Importance: It's the portfolio of a developer. Employers look at your GitHub profile to see what you can do. It's also home to millions of open-source projects.

7. How to Become a Great Software Developer

Becoming the best is a marathon, not a sprint. The journey involves:

- Master the Fundamentals: Don't just jump to the trendiest language. Learn basic programming concepts like variables, loops, and logic very well. These concepts are the same in every language.
- Build Things: Theory is useless without practice. Start building small projects—a calculator, a simple website, a to-do list app. Then make them bigger and more complex.
- Learn to Read Code: Being able to understand other people's code is as important as writing your own. Explore projects on GitHub.
- Practice Problem-Solving: Websites like LeetCode and HackerRank offer coding challenges that sharpen your mind, just like a workout for your brain.

- Collaborate: Work on open-source projects on GitHub. This teaches you how to work in a team, use professional tools, and accept feedback.

8. Renowned Software Developers

- Linus Torvalds: The creator of the Linux operating system (which runs most of the internet's servers) and Git (the technology behind GitHub).
- Tim Berners-Lee: The inventor of the World Wide Web. He wrote the first web browser and server.
- Ada Lovelace (Ancient): Considered the first computer programmer. She wrote the first algorithm for a theoretical computing machine in the 1840s.
- Margaret Hamilton: Lead software engineer for the NASA Apollo moon missions. She coined the term "software engineering" to give her work the seriousness it deserved.
- Brendan Eich: Created the JavaScript programming language in just 10 days. It now powers the modern web.

9. Computer Science vs. Software Engineering

- **Computer Science**: This is the theory. It's like studying physics and mathematics. It focuses on algorithms, how computers work at a deep level, artificial intelligence theory, and what is possible to compute.
- **Software Engineering**: This is the practical application. It's like civil engineering. It takes the theories from computer science and applies them to building reliable, efficient, and usable software systems.

You can be a great software engineer without a deep computer science background, but knowing the theory makes you a much stronger problem-solver.

10. The Future with AI

AI is not replacing software engineers; it is becoming their most powerful assistant.

- AI Assistants: Tools like GitHub Copilot can suggest lines of code as you type, like an autocomplete for programming. This helps developers write code faster and with fewer errors.
- The Change: In the future, the value of a developer will shift from just writing code to designing great systems, solving complex problems, and instructing AI tools effectively. The job will be more about high-level thinking and less about repetitive typing.

Conclusion

Software is the invisible language of our modern world. Becoming a software developer is a journey of continuous learning, problem-solving, and creation. It starts with writing your first simple line of code and can lead to building systems that change how billions of people live and work.