Software Development Methods and Tools: A Learning Journey

SENG 265 Term Portfolio Project - Part 1

Student Name: Tanuj Dargan
Student ID: V01040822
Course: SENG 265 - Software Development Methods
University: University of Victoria
Semester: Spring 2025
Date: March 13, 2025

Purpose of This Portfolio

This portfolio documents my learning journey through SENG 265: Software Development Methods and Tools. It serves as a reflection of my experiences, challenges, and growth throughout the course.

Initial Expectations and Goals

When enrolling in SENG 265, my primary goals were to:

- Develop proficiency in command-line tools and environments
- Gain practical experience with version control systems
- Enhance my programming skills in C and Python
- Learn industry-standard software development methodologies

My Background in Software Development

Prior to this course, my experience with software development included:

- Languages: Python, Java, C++, JavaScript, TypeScript, R, Bash/Linux, SQL, LATEX
- Projects: Ray-Traced Environment in C++ using Hazel Game Engine | Plantbay using Python, Django | UVIC CourseMap and WebScraper using Next.js, Tailwind CSS, MongoDB, Python
- Frameworks/Tools: Familiar with Node.js, Next.js, React, Three.js, Vite, Django, Tailwind CSS, PyTorch, Tensorflow, OpenCV Pandas, Flask, Streamlit, GitHub, Git, Docker, AWS Suite

---

1. Weekly Learning Journal

Week 1: Jan 13–Jan 19: Learned basic Unix commands for navigating directories, managing files, and remote login to UGLS. Initially struggled with adapting to the command line and remembering commands but successfully connected to UGLS and completed file operations like copying, moving, and deleting.

Week 2: Jan 20–Jan 26: Practiced Vim for file editing and set up Git on Unix/Windows. Faced difficulties with Vim keybindings and Git push errors when switching OS, but eventually became comfortable editing files and managing repositories with Git commands like clone, add, commit, and push.

Week 3: Jan 27–Feb 2: Compiled C programs using gcc and structured code for Assignment 1 while introducing makefiles for build automation. Debugging compiler errors and organizing multiple source files were challenging at first, but troubleshooting skills improved, leading to better code management and execution.

Week 4: Feb 3–Feb 9: Worked with multi-file C projects, pointers, and wrote a factorial program. Faced segmentation faults when dealing with pointer arithmetic but gradually improved debugging skills. Successfully linked multiple files and ensured proper function calls using pointers, resulting in a well-structured program.

Week 5: Feb 10–Feb 16: Started with Python, learning basic syntax and using pandas for large dataset analysis. Encountered issues with Python's dynamic typing and CSV performance but created a script to filter and analyze F1 race-winner data efficiently. Appreciated Python's ease of use compared to lower-level languages.

Week 6: Feb 24–Mar 2: Explored advanced Python concepts, including tuples, dictionaries, and the datetime module for date calculations. Managed complex nested data structures and worked through challenges with time conversions. Successfully implemented a feature to track upcoming lab dates, improving both efficiency and readability in code.

Week 7: Mar 3–Mar 9: Implemented linked lists in C, handling insertion, traversal, and deletion while ensuring dynamic memory allocation with malloc. Faced challenges with memory leaks and segmentation faults but refined debugging techniques. Also improved makefile usage for multi-file compilation, leading to more organized and maintainable code.

Week 8: Mar 10–Mar 16: Focused on Python test automation by writing unit tests and using assertions to verify function correctness. Faced challenges in structuring tests for multiple cases and handling edge scenarios but successfully created a robust testing suite. Implementing automated testing streamlined debugging and helped prevent regressions.

---

2. What is the core functionality of Jupyter Notebooks?

Core Functionalities:

- **Notebook Document:**
  Jupyter Notebook documents combine executable code (e.g., Python) with rich text elements like paragraphs, equations, figures, and links. They are both human-readable and executable, enabling seamless coding, analysis, and documentation in one place.
- **Language Support:**
  Jupyter supports over 40 programming languages, including Python, R, Julia, and Scala, making it a versatile tool for data science, machine learning, and computational research.
- **Sharing Capabilities:**
  Notebooks can be shared via email, Dropbox, GitHub, and Jupyter Notebook Viewer, allowing for easy collaboration and knowledge exchange.
- **Interactive Output:**
  Jupyter Notebooks generate rich, interactive outputs, including HTML, images, videos, LaTeX equations, and custom MIME types, enhancing data presentation and visualization.
- **Extensions:**
  Jupyter supports extensions that enhance functionality, such as "Table of Contents" for navigation and "Variable Inspector" for viewing active variables, improving workflow efficiency.
- **Data Analysis and Visualization:**
  Jupyter is widely used for data cleaning, transformation, numerical simulation, visualization, machine learning, and deep learning, making it a comprehensive tool for data science.
- **Open-Source and Free:**
  As an open-source web application, Jupyter is free to use, modify, and distribute, with strong community support for continuous improvements and accessibility.

---

3. Simple Jupyter Notebook Markdown (Including Links and Images)

Markdown is used in Jupyter Notebooks to format text cells. Below are the essentials:

1. **Headings**

   # Heading 1

   ## Heading 2

   ### Heading 3

   More hash marks indicate deeper levels of headings.

2. **Lists**

   - Bulleted lists use asterisks or dashes:
     - Item 1
     - Item 2
   - Numbered lists use numbers:

     A. First
     B. Second

3. **Links**
   Use the bracket and parenthesis syntax:

   Link Text

4. **Images**
   The syntax matches links but with an exclamation mark in front:

![image description] (https://www.example.com/image.jpg)

4. How can you typeset mathematical formulas including Greek letters using LaTeX Markdown in Jupyter Notebooks? LaTeX is used extensively used for documents with mathematical formulas.

Jupyter Notebooks support LaTeX for mathematical formulas:

- Inline equations use single dollar signs: $E = mc^2$ The input is `$E = mc^2$`
- Display equations use double dollar signs:

$$\frac{d}{dx}(x^n) = nx^{n-1}$$

Input LaTeX: `$$\frac{d}{dx}(x^n) = nx^{n-1}$$`

$$\int_a^b f(x)dx = F(b) - F(a)$$

Input LaTeX: `$$\int_a^b f(x) dx = F(b) - F(a)$$`

Greek letters are commonly used in formulas:

- Alpha: α
- Beta: β
- Gamma: γ
- Delta: Δ
- Pi: π Example Input for greek letters: `$\alpha$`

Matrix example:

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$$

---

5. What is the history of Unix and Linux? How prevalent is Unix or Linux in software development today?

History of Unix and Linux

Unix, initially developed by Ken Thompson, Dennis Ritchie, and others at Bell Labs in the late 1960s and early 1970s, evolved through collaborations with Unix System Laboratories and Sun Microsystems. The release of BSD Unix in 1992 led to derivatives like NetBSD and FreeBSD. Linux, introduced in 1991 by Linus Torvalds, is an open-source Unix-like OS built around the Linux kernel. Inspired by MINIX, Linux is distributed with system software and libraries, often from the GNU Project, forming various Linux distributions.

Prevalence in Software Development Today

Unix and Linux dominate software development, replacing many proprietary Unix systems. Their presence extends beyond servers into consumer desktops, mobile devices, and embedded systems. Notable Linux distributions include Red Hat, Fedora, SUSE, Debian, and Ubuntu. Linux is particularly significant in server environments, often running without a graphical interface or using stacks like LAMP. Mac OS X, a Unix-based system, integrates BSD and the Mach kernel, further cementing Unix's influence in the consumer market.

Linux and Unix-based systems are essential in:

- Web servers and cloud infrastructure
- Mobile OS platforms like Android
- DevOps and system administration
- Scientific computing and data science

---

6. What is the core functionality of the Bash? command and scripting language?

What is Bash? 

Bash (Bourne Again Shell) is the default command-line interpreter for most Unix and Linux distributions. It is a command processor that operates in a text window, allowing users to input commands to trigger actions. Bash serves as a command language interpreter and is the default on many GNU/Linux systems. The name is a play on words, referencing the Bourne shell it replaces and the concept of being "born again."

Core Functionality of Bash

Bash executes commands interactively or from a file, known as a shell script. A script comprises standard commands and advanced scripting constructs. Any command executable in a script can also be run directly in the terminal. Bash supports features such as filename globbing, piping, here documents, command substitution, variables, and control structures. It inherits elements from sh while incorporating enhancements from csh and ksh, making it a POSIX-compliant shell with extensions.

Bash Scripting

Bash scripting automates tasks, eliminating the need for manual execution. It is essential for:

- Automation and scripting
- System administration
- Software development and deployment

Bash scripts support variables, conditional statements, and loops, similar to programming languages. They are widely used for file manipulation, routine tasks (e.g., backups), and general automation, enabling the execution of multiple commands as a single command.

---

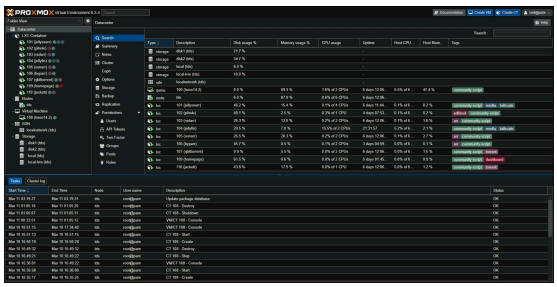
7. What is your background in the scripting language Bash? How prevalent is Bash in software development today?

My background with Bash

My background with Bash is extensive and rooted in practical, hands-on experience. I manage a homelab where I host numerous services using Proxmox, and Bash plays a critical role in automating and maintaining this environment. I've written scripts to automate service installations, container creation, and other repetitive tasks, which has significantly streamlined my workflow.

Beyond my personal projects, I actively contribute to the community by sharing these scripts, helping others set up and manage their own services efficiently. This has deepened my understanding of Bash, as I often troubleshoot and optimize scripts to ensure they work across different environments. My experience has taught me how powerful Bash can be for system administration, automation, and orchestration tasks.

Take a look at my Proxmox Homelab:



View My Github

How prevalent is Bash in software development today?

Bash remains a cornerstone of software development, particularly in environments that rely on Unix/Linux systems. It is widely used for automating tasks, managing system configurations, and orchestrating workflows. System administrators and DevOps engineers rely on Bash for tasks like server provisioning, container orchestration (e.g., Docker), and CI/CD pipelines. Developers also use Bash to automate repetitive tasks, such as compiling code, running tests, and deploying applications.

Bash is especially prevalent in homelabs, like mine, where it simplifies the management of services and infrastructure. It's also a beginner-friendly scripting language, making it an excellent starting point for those new to programming or system administration. While more versatile languages like Python are often preferred for complex tasks, Bash excels in scenarios where quick, lightweight automation is needed.

In summary, Bash continues to be a vital tool in software development and system administration, and my experience with it has been both extensive and impactful in my personal and community-driven projects.

---

8. How can Bash be used for automating tasks? Frequent midterm question.

Bash Scripting

Bash scripting automates tasks, eliminating the need for manual execution. It is essential for:

- Automation and scripting
- System administration
- Software development and deployment

Bash scripts support variables, conditional statements, and loops, similar to programming languages. They are widely used for file manipulation, routine tasks (e.g., backups), and general automation, enabling the execution of multiple commands as a single command.

How Can Bash Be Used for Automating Tasks?

Bash is widely used for task automation, reducing manual work and improving efficiency. Some common applications include:

- **Automating System Administration**: Automating software installations, system updates, log file analysis, and scheduled backups using cron jobs.
- **File and Directory Management**: Automating file organization, renaming, and cleaning up directories.
- **Data Processing**: Extracting, filtering, and transforming data using commands like `awk`, `sed`, and `grep`.
- **Deployment Automation**: Automating code deployment, build processes, and CI/CD pipelines.
- **Monitoring and Logging**: Automating system health checks and alerting administrators of critical events.

---

9. Why are version control and configuration management critical in software development projects? Frequent midterm question.

Version control and configuration management are essential for maintaining code integrity, collaboration, and consistency in software development projects.

Version Control

- Tracks changes to code, enabling rollback to previous versions if needed.
- Facilitates collaboration among multiple developers, preventing conflicts.
- Provides a history of modifications for debugging and auditing.
- Supports branching and merging, allowing parallel development efforts.
- Examples: Git, Subversion (SVN), Mercurial.

Configuration Management

- Ensures consistency across different environments (development, testing, production).
- Automates software setup and dependency management.
- Reduces errors caused by manual configuration changes.
- Enhances scalability by maintaining infrastructure as code (IaC).
- Examples: Ansible, Puppet, Chef, SaltStack.

Both version control and configuration management contribute to efficient development workflows, reducing risks and increasing software reliability.

---

10. What are the core functionalities of Git, Github, and Gitlab? Frequent midterm question.

Git

Git is a distributed version control system designed to track changes in files and coordinate work among multiple developers. It allows users to manage file revisions, branch and merge efficiently, and work offline. Git is open-source, enabling users to modify its source code, and is widely used for collaborative software development due to its speed, reliability, and support for non-linear workflows.

GitHub

GitHub is a cloud-based hosting service for Git repositories. It extends Git's functionality by providing tools for collaboration, project management, and deployment. GitHub allows developers to push and pull changes, manage repositories, and collaborate globally. It also includes features like pull requests, issue tracking, and actions for CI/CD workflows, though its security features are considered less robust compared to GitLab.

GitLab

GitLab is a comprehensive DevOps platform that integrates Git repository hosting with tools for project planning, monitoring, and security. It supports both open and private repositories, issue tracking, wikis, and CI/CD pipelines. GitLab is known for its enhanced security features, support for various attachments, and organization-wide permissions. It streamlines development workflows and reduces product lifecycles, making it a powerful tool for teams.

---

11. How is Git used as a version control system?

Git is used as a version control system to track changes in files and manage collaborative development. It allows developers to create repositories where they can store and organize code. Key functionalities include:

- **Version Tracking**: Git records every change made to files, enabling developers to view the history of modifications and revert to previous versions if needed.
- **Branching and Merging**: Developers can create branches to work on features or fixes independently, then merge them back into the main codebase without disrupting others' work.
- **Collaboration**: Git enables multiple developers to work on the same project simultaneously by managing conflicts and integrating changes efficiently.
- **Distributed Workflow**: Each developer has a complete copy of the repository, allowing them to work offline and sync changes later.
- **Commit History**: Git maintains a detailed log of changes, including who made them and why, providing transparency and accountability.

Git's flexibility and efficiency make it an essential tool for managing codebases in software development.

---

12. How can Git be used effectively in a highly distributed software development environment involving multiple countries?

Git is highly effective in distributed environments due to its decentralized nature, allowing developers across different countries to collaborate seamlessly. Each developer has a complete copy of the repository, enabling offline work and asynchronous contributions, which is ideal for teams in different time zones.

Key features like branching and merging allow developers to work on isolated features or fixes without disrupting the main codebase. Remote repositories (e.g., GitHub or GitLab) act as central hubs for syncing changes, while pull requests and code reviews ensure high-quality contributions. Git's conflict resolution tools help integrate changes smoothly, and its detailed commit history provides transparency and accountability.

Additionally, Git integrates with CI/CD pipelines to automate testing and deployment, ensuring a stable codebase. These features make Git an essential tool for managing code in globally distributed teams.

---

13. What is your background in the programming language C? How prevalent is C in software development today?

My background with C

My experience with C comes from working on embedded systems and low-level programming projects. As part of the UVic Formula Hybrid Team, I developed sensor systems for a hybrid car using the CAN communication protocol in C, ensuring efficient data integration and control. This hands-on experience has given me a strong understanding of C's role in hardware-level programming and real-time systems. Additionally, my academic background in computer science has reinforced my knowledge of C's syntax, memory management, and performance optimization.

How prevalent is C in software development today?

C remains highly relevant in software development, especially for systems that require high performance and direct hardware interaction. It is widely used in operating systems, embedded systems, game engines, and other performance-critical applications. Despite the rise of higher-level languages, C's efficiency, portability, and control over system resources make it indispensable for foundational software. Many modern languages, like C++ and Python, are influenced by C, further cementing its importance in the software development ecosystem.

---

14. What is your background in the programming language Python 3? How prevalent is Python 3 in software development today?

My background with Python 3

I have extensive experience with Python 3, using it across various projects and roles. At Nanodock, I developed workflows using Python, Docker, and NixOS to generate developer environments. During my research internship at Makers Asylum, I analyzed water contamination data with Python and Excel. Additionally, I've built Python-based tools like a web scraper for UVic CourseMap, optimized with BeautifulSoup, and a Django app for analyzing plant health. My work also includes developing a quantitative finance library and participating in hackathons, where I used Python for machine learning and reinforcement learning models.

How prevalent is Python 3 in software development today?

Python 3 is one of the most widely used programming languages today due to its simplicity, versatility, and extensive libraries. It is prevalent in fields like web development, data science, machine learning, automation, and scientific computing. Python's readability and ease of use make it a popular choice for beginners, while its powerful frameworks and tools make it indispensable for professionals. Its adoption in academia, startups, and large enterprises ensures its continued relevance in modern software development.

---

15. What are the most popular programming languages and why?

The most popular programming languages today as per the 2024 Stackoverflow Developer Survey include Python, JavaScript, Typescript, Java, C, and C++. Python is favored for its simplicity, versatility, and extensive libraries, making it ideal for data science, machine learning, and web development. JavaScript dominates web development due to its ability to create interactive frontends and backends. Java is widely used for enterprise applications and Android development because of its stability and scalability. C and C++ are essential for systems programming, game development, and performance-critical applications due to their speed and low-level control. These languages remain popular because they address diverse needs across industries and have strong community support.

---

16. C & Python are among the most popular programming languages. You are learning them in SENG 265. How cool is that for your software development career?

Learning C and Python in SENG 265 is incredibly beneficial for your software development career. C provides a strong foundation in low-level programming, memory management, and performance optimization, which are critical for systems programming and embedded systems. Python, on the other hand, is versatile and widely used in fields like data science, machine learning, and web development. Mastering both languages equips you with a diverse skill set, enabling you to tackle a wide range of projects and making you highly competitive in the software development industry.

---

17. Describe the fundamental differences between C and Python.

Feature	C	Python
Type System	Static typing	Dynamic typing
Memory Management	Manual memory allocation/deallocation	Automatic garbage collection
Syntax	Verbose, requires semicolons	Concise, uses indentation for blocks
Compilation	Compiled to machine code	Interpreted (bytecode)
Performance	Generally faster execution	Generally slower execution
Development Speed	Slower development cycle	Rapid development
Use Cases	Systems programming, embedded systems	Web development, data science, automation
Libraries	Standard library is minimal	Extensive standard library and third-party packages

C and Python differ fundamentally in their design and use cases. C is a low-level, statically typed, compiled language that provides fine-grained control over memory and system resources, making it ideal for systems programming and performance-critical applications. Python, on the other hand, is a high-level, dynamically typed, interpreted language known for its simplicity and extensive libraries, making it perfect for rapid development, data science, and automation. While C prioritizes performance and efficiency, Python emphasizes ease of use and developer productivity. Both are powerful tools, excelling in different domains of software development.

---

18. How challenging was learning C for you?

Learning C was slightly difficult for me initially because its syntax and manual memory management are quite different from higher-level languages like Python. However, I found it rewarding as it gave me a deeper understanding of how computers work at a low level. Working on projects like developing sensor systems for the UVic Formula Hybrid car using C helped me appreciate its power and efficiency, making the learning process worthwhile.

---

19. How challenging was completing Assignment 1 for you?

Completing Assignment 1 was a bit difficult as I frequently confused C's syntax with other languages I know, like Python. Debugging and understanding manual memory management also took some time. However, AI tools like Claude were incredibly helpful in clarifying concepts and troubleshooting errors. Despite the challenges, the assignment was a great learning experience and helped me strengthen my understanding of C. GPT-4o AI agent thumbnail

---

20. Required: What are your personal insights, aha moments, and epiphanies you experienced in the first part of the SENG 265 course?

In the first part of SENG 265, I had several insights and "aha" moments that deepened my understanding of programming and problem-solving. One key realization was how foundational concepts like memory management and pointers in C provide a deeper understanding of how computers operate at a low level. This was an eye-opener, as I had previously relied on higher-level languages like Python, which abstract these details. Another epiphany came when I successfully debugged a segmentation fault for the first time—it taught me the importance of careful memory handling and how small mistakes can have significant consequences. Additionally, learning to write efficient, concise code in C helped me

appreciate the balance between performance and readability. These moments not only enhanced my technical skills but also shifted my perspective on programming, making me more detail-oriented and methodical in my approach.

---

21. Required: How did you experience GenAI tools (e.g., OpenAI chatGPT, Microsoft CoPilot, Google Gemini, Perplexity) as learning tools?

I found GenAI tools, particularly Claude, to be incredibly helpful as learning aids during the course. Claude provided clear explanations of complex concepts, helped debug code, and offered alternative approaches to solving problems. It was especially useful when I struggled with C syntax or debugging errors, as it provided step-by-step guidance that enhanced my understanding. Unlike static resources, Claude's interactive nature allowed me to ask follow-up questions and refine my learning process. It acted as a supportive tutor, helping me bridge gaps in my knowledge and boosting my confidence in tackling assignments. Overall, using Claude made learning more efficient and engaging.

---

22. Record your Generative AI prompts for Bash, C, and Python inquiries for easy recall.

Bash:

- "How to use scp and basic UNIX commands to interact with ssh external servers?"
- "Key `vim` commands and Git setup steps on UNIX/Windows?"

C:

- "How to compile and link C programs using `gcc` and makefiles?"
- "Explain dynamic memory allocation and linked lists in C."
- "Example of using `qsort` in C to sort random values."
- "Steps to process CSV data in C and generate output files."

Python:

- "Basics of Python: pandas, loops, lists, and dictionaries?"
- "How to analyze CSV data with pandas (e.g., F1 race winners)?"
- "Write Python code for unit testing and assertions."
- "Use `datetime` to calculate remaining lab days."

---

23. References and Bibliography

Course Materials

- SENG265, Dr. Hausi A. Müller, University of Victoria, 2025 Spring
- Course slides and lecture notes from SENG265 Brightspace

Online Resources

- Python Software Foundation. (n.d.). *Python Documentation*. Retrieved from https://docs.python.org/3/
- Git Project. (n.d.). *Git Documentation*. Retrieved from https://git-scm.com/doc
- Jupyter Project. (n.d.). *Jupyter Notebook Documentation*. Retrieved from https://jupyter-notebook.readthedocs.io/en/stable/notebook.html
- Red Hat. (n.d.). *A Brief History of Unix and Linux*. Retrieved from https://www.redhat.com/sysadmin/unix-linux-history
- DigitalOcean. (n.d.). *A Brief History of Linux*. Retrieved from https://www.digitalocean.com/community/tutorials/brief-history-of-linux
- Devhints. (n.d.). *Bash Cheatsheet*. Retrieved from https://devhints.io/bash
- Tables Generator. (n.d.). *Markdown Tables Generator*. Retrieved from https://www.tablesgenerator.com/markdown_tables
- FrontPageLinux. (n.d.). *Guide Through the History of Unix/Linux*. Retrieved from https://frontpagelinux.com/articles/guide-through-history-of-unix-linux-everything-you-need-to-know/
- FreeCodeCamp. (n.d.). *Git and GitHub Crash Course for Beginners*. Retrieved from https://www.freecodecamp.org/news/git-and-github-for-beginners/
- Towards Data Science. (n.d.). *Mastering Jupyter Notebooks*. Retrieved from https://towardsdatascience.com/mastering-jupyter-notebooks-f6a1db30c43f
- Linux Handbook. (n.d.). *Essential Linux Commands*. Retrieved from https://linuxhandbook.com/linux-commands-cheat-sheet/

GenAI-Assisted Content

- Claude 3.5 Sonnet, Claude 3.7 Extended, OpenAI o1, ChatGPT
- You.com AI Search
- Claude AI Platform (https://claude.ai)
- OpenAI ChatGPT (https://chat.openai.com/)