Practical Question - Software Engineering

Sou Chanrojame

December 4, 2025

I. Conceptual Questions

1. What are the challenges of learning software engineering?

- **Complexity:** Software systems are inherently complex with many moving parts
- **Abstract Nature:** Software is intangible, making it harder to visualize than physical engineering
- **Rapid Technology Change:** Tools, languages, and frameworks evolve constantly
- **Team Dependency:** Learning often requires understanding group dynamics, not just coding
- **Legacy Systems:** Understanding how to maintain old code is as important as writing new code.

2. List all the crucial factors that lead to software **development failure and success?**

- **Success Factors:**
  - Clear and frozen requirements
  - Strong user/stakeholder involvement
  - Skilled and motivated team
  - Realistic expectations and schedules
  - Proper planning and project management
- **Failure Factors:**
  - Unrealistic or changing requirements (Scope Creep)
  - Lack of user input
  - Poor communication within the team
  - Inaccurate cost/time estimation
  - Using immature technology.

3. Compare the **differences between traditional software developments with Modern Software Development Methodologies.**

- **Traditional (e.g., Waterfall):**
  - **Linear:** Phases happen sequentially (Requirements -> Design -> Code -> Test)
  - **Rigid:** Difficult to change requirements once a phase is done
  - **Documentation-driven:** Heavy emphasis on comprehensive documents
  - **Late Testing:** Testing happens at the end
- **Modern** (e.g., Agile, **Scrum, DevOps):**
  - **Iterative:** Development happens in cycles (Sprints)
  - **Flexible:** Welcomes changing requirements even late in development
  - **Code/Communication-driven:** Emphasis on working software and collaboration
  - **Continuous Testing:** Testing is integrated throughout the lifecycle.

4. [**Implied Question regarding Development Methodologies**] (Note: Question 4 in the source text appears to be a fragment or header. Assuming it asks to list methodologies)

- **Waterfall Model**
- **Agile Frameworks** (Scrum, Kanban)
- **DevOps**
- **Spiral Model**
- **Rapid Application Development (RAD)**

5. List all the attributes of high-quality software?

- **Maintainability:** Easy to modify or fix
- **Dependability/Reliability:** Does not crash; secure and safe
- **Efficiency:** Uses resources (memory, CPU) wisely; fast performance
- **Usability:** Easy for users to learn and use
- **Portability:** Can run on different environments/OS.

6. How to identify **software validity and software reliability?**

- **Validity (Building the *right* product):** Does the software meet the user's actual needs? Identified through User Acceptance Testing (UAT) and requirements reviews
- **Reliability (Building the product *right*):** Does the software operate without failure? Identified through stress testing, crash reporting, and tracking Mean Time Between Failures (MTBF).

7. List all the key characteristics of being a good and professional **software engineer today?**

- **Technical Competence:** Mastery of coding, algorithms, and architecture
- **Communication Skills:** Ability to explain technical concepts to non-tech stakeholders
- **Ethical Responsibility:** Respecting privacy, security, and intellectual property
- **Teamwork:** Ability to collaborate effectively using tools like Git/Jira
- **Lifelong Learning:** Adapting to new technologies continuously.

8. [Duplicate **of Q6**]

- *See answer to Question 6.*

9. How to develop the **best quality software with low cost? Explain it based on your experience.**

- **Reuse Components:** Don't reinvent the wheel; use established libraries/frameworks
- **Shift-Left Testing:** Test early and often to catch bugs when they are cheap to fix
- **Clear Requirements:** Spend time clarifying needs upfront to avoid costly rework
- **Automation:** Automate builds, testing, and deployment (CI/CD) to reduce human effort.

10. What are the key challenges in the software development profession?

- Dealing with legacy code (code written by others years ago)
- Balancing technical debt vs. new features
- Managing tight deadlines and high pressure
- Ensuring security in an increasingly hostile cyber environment.

11. Describe the differences between Software Manual Testing and Software **Automated Testing along with examples.**

- **Manual Testing:**
  - *Description:* A human tester plays the role of the user to find bugs
  - *Pros:* Good for UI/UX, ad-hoc, and exploratory testing
  - *Example:* A QA engineer manually clicks through a "Sign Up" form to see if it accepts invalid emails
- **Automated Testing:**

  - *Description:* Scripts and tools execute pre-defined tests
  - *Pros:* Fast, repeatable, good for regression and load testing
  - *Example:* A Python script (using Selenium) automatically logs into a website 1000 times to check server stability.

12. Describe the process you use for writing a piece of code, from requirements to delivery?

1. **Analyze:** Understand the specific requirement (ticket/story)
2. **Design:** Plan the logic (pseudocode or mental model)
3. **Code:** Write the implementation
4. **Unit Test:** Write tests to verify the specific function works
5. **Refactor:** Clean up the code for readability
6. **Review:** Submit a Pull Request for peer review
7. **Merge/Deploy:** Integrate into the main branch.

13. What is your process to test and find bugs in an application?

- **Reproduce:** Try to consistently trigger the bug
- **Isolate:** Narrow down the exact module or line of code causing the issue using logging or debuggers
- **Fix:** Apply the correction
- **Regression Test:** Ensure the fix didn't break anything else.

14. What is the difference between functional requirements and non-functional requirements?

- **Functional Requirements:** Define *what* the system does (features)
  - *Example:* "The system shall allow users to reset their password."
- **Non-functional Requirements:** Define *how* the system performs (quality attributes)
  - *Example:* "The password reset email must be sent within 5 seconds."

15. How can you make sure that your code is both safe and fast from software vulnerabilities?

- **Input Validation:** Sanitize all user inputs (prevent SQL Injection/XSS)
- **Code Reviews:** Have peers check for security flaws
- **Static Analysis Tools:** Use tools (like SonarQube) to auto-scan for vulnerabilities
- **Optimization:** Use efficient algorithms (Big O notation) and database indexing.

16. How to find the size of a software product in software development professional?

- **Lines** of Code **(LOC):** Counting executable lines (simple but can be misleading)
- **Function Points (FP):** Estimating based on inputs, outputs, inquiries, files, and interfaces
- **Story Points:** (Agile) Relative sizing based on complexity and effort.

17. [Duplicate of Q2]

- *See answer to Question 2.*

18. [Duplicate of Q5]

- *See* answer *to Question 5.*

19. Differentiate between Verification and Validation (V&V). Describe two non-functional testing types.

- **Verification:** "Are we building the product right?" (Adhering to specs/standards)
- **Validation:** "Are we building the right product?" (Meeting user needs)
- **Non-functional Testing Types:**

1. **Load/Performance Testing:** Checking how the system behaves under heavy user traffic. *Metric:* Response time (ms), Throughput (req/sec)
2. **Security Testing:** Checking for vulnerabilities. *Metric:* Number of open vulnerabilities, penetration test pass rate.

20. Explain the core principles of Scrum. Roles, Artifacts, and Agile vs Waterfall.

- **Core Principles:** Transparency, Inspection, Adaptation
- **Agile vs. Waterfall:** Agile handles change by working in short iterations (Sprints), allowing requirements to evolve. Waterfall locks requirements early, making change expensive
- **Roles:** Product Owner (Value), Scrum Master (Process), Developers (Work)
- **Artifacts:** Product Backlog (List of work), Sprint Backlog (Plan for current sprint), Increment (Finished work).

21. Professional and ethical responsibilities regarding privacy and bias.

- **Responsibilities:** Engineers must protect user data (GDPR/compliance) and ensure algorithms don't discriminate (e.g., AI bias in hiring)
- **Ethical Dilemma Example:** A manager asks you to scrape user data without consent to improve an ad algorithm
- **Process:** Refuse the request citing ethical codes (ACM/IEEE), propose a legal alternative (using anonymized public data), or escalate to higher management/compliance officers.

22. What **is an SRS and what are good requirement qualities?**

- **SRS (Software Requirements Specification):** A comprehensive document describing the intended purpose and environment for software under development
- **Qualities:** Unambiguous (one interpretation), Verifiable (testable), Complete, Consistent
- **Eliciting Non-functional reqs:** Use questionnaires, analyzing competitor products, and interviewing technical stakeholders (admins/security leads), not just end-users.

23. Technical **Debt and Refactoring.**

- **Technical Debt:** The implied cost of additional rework caused by choosing an easy/fast solution now instead of a better approach that would take longer
- **Trade-offs:** A PM must decide if hitting a deadline (shipping features) is worth the risk of instability or slower future development
- **Code Smells:**
  - *Long Method:* A function doing too many things
  - *Duplicated Code:* Copy-pasting logic (violates DRY)
  - *Large Class:* A class ("God Object") that knows too much.

24. Process vs. Thread and Concurrency.

- **Difference:** A **Process** is an independent program execution with its own memory space. A **Thread** is a lighter unit of execution within a process that shares memory
- **Challenges:**
  - *Race Condition:* Two threads trying to change shared data simultaneously
  - *Deadlock:* Two threads waiting on each other forever
- **Synchronization:**
  1. **Mutex (Mutual Exclusion):** Locks a resource so only one thread uses it at a time
  2. **Semaphore:** Controls access to a common resource by multiple threads (counter-based).

25. Product vs. Process Metrics.

- **Product Metrics:** Measure the software itself (e.g., Cyclomatic Complexity, LOC). Used to assess code maintainability
- **Process Metrics:** Measure the workflow (e.g., Defect Removal Efficiency, Release Cycle Time). Used to assess team efficiency.

II. Problem Solving Questions

1. Product vs. Process Diagram Analysis

- **Image Description:** The diagram connects People, Projects, Product, Process Models, and Tools
- **Explanation:** This diagram illustrates the **Software Engineering Ecosystem**
  - **Process Model & Template:** Provides the blueprint or "Template" for how the work should be done. It defines the rules
  - **Tools & Automation:** Tools interact with the Process Model to provide "Automation," making the work faster and less error-prone
  - **People:** The "Participants" who actually do the work
  - **Project:** The central hub where People follow the Process (using Templates) to work
  - **Product:** The final "Result" generated by the Project
  - **Key Difference:** The **Product** is the output (the software), while the **Process** is the method and set of steps used to create that output.

2. Importance of Software Process Diagram Analysis

- **Image Description:** A flow from Problem Statement -> Code -> Compile -> Unit Test -> Release, with a "Debug" feedback loop
- **Explanation:** The software process is important because it brings **structure to chaos**
  - Without a process, you might jump straight to coding without understanding the problem
  - The diagram shows that **Verification** (Compile/Unit Test) acts as a gatekeeper
  - The **Feedback Loop (Debug)** ensures that if a problem is found during testing, the process routes you back to fix the code before Release
  - It ensures quality control before the product reaches the user.

3. Calculation: Effort and Cost

Given Data:

- Total LOC = **55,600 LOC**
- Productivity = **1,200** LOC / **person-month**
- Team Size = **10 developers**
- Labor Rate = **$2,000 / person-month**

Solution 1:

Step 1: Calculate Total Effort (in Person-Months)

$$\text{Total Effort} = \frac{\text{Total LOC}}{\text{Productivity}}$$

$$\text{Total Effort} = \frac{55,600}{1,200} \approx \mathbf{46.33} \text{ person-months}$$

Step 2: Calculate Project Duration (in Months)

$$\text{Duration} = \frac{\text{Total Effort}}{\text{Team Size}}$$

$$\text{Duration} = \frac{46.33}{10} \approx \mathbf{4.63} \text{ months}$$

Step 3: Calculate Total Cost

$$\text{Total Cost} = \text{Total Effort} \times \text{Labor Rate}$$

$$\text{Total Cost} = 46.33 \times \$2,000 = \$\mathbf{92,660}$$

(Alternative Calculation via Duration: 10 devs × $2000 × 4.633 months = $92,660)

Solution 2:

Step 1: Calculate Cost per LOC

$$\text{Cost per LOC} = \frac{\text{Labor Rate}}{\text{Productivity}}$$

$$\text{Cost per LOC} = \frac{\$2,000}{1,200} \approx \$\mathbf{1.66} \text{ / LOC}$$

Step 2: Calculate Total Cost

$$\text{Total Cost} = \text{Total LOC} \times \text{Cost per LOC}$$

$$\text{Total Cost} = 55,600 \times \$1.66 \approx \$\mathbf{92,660}$$

Step 3: Calculate Total Effort

$$\text{Total Effort} = \frac{\text{Total Cost}}{\text{Labor rate}}$$

$$\text{Total Effort} = \frac{\$92,660}{\$2,000} = \mathbf{46.33} \text{ person-months}$$

4. Maintenance Flow Diagram Analysis

- **Image Description:** Shows the journey of a Maintenance Request (MR) from Customer to Change Control Board (CCB) to Engineer
- **Workflow Description:**
  1. **Initiation:** A **Customer** reports an issue or requests a feature via the **Help Desk**
  2. **Formalization:** This is converted into a **Written MR (Maintenance Request)**
  3. **Approval:** The MR is proposed to the **Change Control Board (CCB)**. They decide if the change is worth the cost/risk
  4. **Assignment:** If approved ("Approved MR"), it goes to the **Maintenance Engineer**
  5. **Execution:** The engineer takes the "Current source & documentation," modifies it, and produces the "Modified source & documentation."
  6. **Closure:** The update is deployed back to the user (implied by the loop).

5. **DevOps Process Diagram Analysis**

   - **Image Description:** An infinity loop symbol containing Dev (Plan, Code, Build, Test) and Ops (Release, Deploy, Operate, Monitor)
   - **Process Description:** DevOps is a methodology that unifies software development (Dev) and software operation (Ops). It is not linear; it is a continuous cycle
   - **The Cycle:**
     1. **Dev Side:** You **Plan** the feature, **Code** it, **Build** the application, and **Test** it
     2. **Ops Side:** You **Release** the build, **Deploy** it to servers, **Operate** the live system, and **Monitor** for performance/bugs
     3. **Feedback:** Data from **Monitor** feeds back into the **Plan** phase for the next update
   - **Realistic Example:**
     - *Scenario:* Netflix updating their streaming algorithm
     - *Dev:* Developers write code for the new algo and automated tests run (CI)
     - *Ops:* The code is automatically deployed to a small subset of users (CD)
     - *Monitor:* Use tools to check if the new algo causes buffering. If stable, roll out to everyone. If not, rollback.

6. **Throwaway Prototyping Diagram Analysis**

   - **Image Description:** Planning -> Analysis -> Design -> Prototype -> Implementation -> System
   - **Process Description:**
     1. **Planning/Analysis:** Determine the basic needs
     2. **Design Prototype:** Instead of building the whole system, build a quick, "mock" version of the complex parts
     3. **User Review:** Show this prototype to the user. "Is this what you meant?"
     4. **Discard & Implement:** Once requirements are clarified via the prototype, the prototype is often **thrown away** (discarded), and the real **Implementation** is built using solid engineering practices to create the final **System**
   - **Realistic Example:**
     - *Project:* Building a complex banking dashboard
     - *Prototype:* The team uses a tool like Figma or simple HTML to create a clickable dummy version of the dashboard. It looks real but has no database behind it
     - *Feedback:* The bank manager