# 1 Example Sheet 1: Mindset, Requirements & Process

**Q1. The Complexity of Scale.** Compare a solo developer building a personal "To-Do List" app versus a team of 50 building a "National Health Records System."

(a) Explain why the health system requires a "Software Engineering" approach while the To-Do app may only require "Programming."

(b) Identify three specific *emergent behaviors* (one positive, two negative) that might arise in the health system that would be impossible in the solo app.

(c) How does the *discrete state* of software increase the risk profile of the health system compared to a physical structure like a hospital building?

**Q2. Requirements Engineering: The Autonomous Drone.** A startup wants to build a drone for "ultra-fast and safe pizza delivery in urban environments."

(a) Critique the requirement "The drone must be ultra-fast and safe." Use the standard criteria for good requirements.

(b) Propose three testable **Non-Functional Requirements (NFRs)** for this drone, covering performance, safety, and reliability.

(c) Discuss the inevitable *engineering trade-offs* between your proposed NFRs. How would you justify a decision to a stakeholder?

**Q3. Brooks's Law & Project Management.** A critical software project is estimated to take 6 months with 5 developers. After 3 months, the team is only 25% complete. The CEO proposes adding 5 more developers immediately to "double the speed."

(a) Apply Brooks's Law to this scenario. What is the likely immediate effect on the project's progress?

(b) Identify two specific "hidden costs" of adding these new developers.

(c) Propose an alternative strategy for the project manager to handle the delay without adding staff.

**Q4. Technical Debt & The Legacy Trap.** A startup chooses a fast-but-messy "hack" to launch their product 3 months early.

(a) Define *Technical Debt* in this context. Is it always bad to take on debt?

(b) Two years later, the team finds that adding a simple button takes 4 weeks because the code is so tangled. Explain this using the concept of "interest" on technical debt.

(c) Describe a strategy for "refactoring" this debt without stopping all new feature work (which would kill the business).

**Q5. Process Selection & The Cost of Change.** Consider two projects: (A) Flight control software for a new passenger jet, and (B) A mobile app for Cambridge students to share lecture notes.

(a) Which process model (Waterfall, Spiral, Agile) is most appropriate for each? Justify your choice using the concepts of *risk* and *requirement stability*.

(b) Draw a conceptual "Cost of Change" curve for both. Why does the curve for the jet software likely rise much more steeply than for the mobile app?

(c) Describe how a "Sprint" in an Agile process for the mobile app addresses the *Translation Gap* described in Lecture 2.

**Q6. Requirements Elicitation: The Hospital Ward.** You are designing a tablet-based system for nurses to record patient vitals.

(a) You interview the Head of Nursing, who requests a feature requiring a 10-digit PIN for every data entry to ensure security. Why might an *Ethnographic study* (observation) lead you to reject or modify this requirement?

(b) Distinguish between a *User Requirement* and a *System Specification* in this context.

(c) How might a *Functional Requirement* for security (data privacy) conflict with a *Non-Functional Requirement* for usability (emergency response time)?

**Q7. Scrum Ceremonies & Failure Modes.** A team is performing "Scrum," but they have a 2-hour Daily Standup, and at the end of every 2-week Sprint, they have no working software to show the Product Owner.

(a) Which specific Scrum principles are being violated here?

(b) What is the intended purpose of the *Sprint Retrospective*, and how could it be used to fix the "2-hour standup" problem?

(c) Explain the difference between the *Scrum Master* and a traditional *Project Manager* in this scenario.

**Q8. The Agile Manifesto in Conflict.** A government department wants to use Agile to build a new tax system, but their legal department insists on a 500-page fixed-price contract with every feature defined in advance.

(a) Which specific value of the Agile Manifesto is in direct conflict with the legal department?

(b) Discuss the risk of following a "Plan" (the contract) versus "Responding to Change" in the context of a 2-year government project.

(c) How could the team use *Prototyping* (from the Spiral model) to satisfy both the need for legal certainty and the need for Agile flexibility?

## 2 Example Sheet 2: Quality, Evolution & AI

**Q1. VCS and Conflict Resolution.** Two developers, Alice and Bob, branch from `main` at the same time.

(a) Alice refactors a class `User` to rename `get_name()` to `get_full_name()`. Bob adds a new method `get_address()` to the same file. Describe the merge process. Will Git flag a conflict?

(b) If Bob had *also* modified the logic inside `get_name()` while Alice renamed it, how would a Distributed VCS handle this differently than a Centralized one?

(c) Propose a branching strategy for a team of 30 developers that minimizes the risk of "Merge Hell" at the end of a 2-week cycle.

**Q2. Quality Assurance: Mutation Testing.**

(a) A team achieves 95% code coverage but still finds many bugs in production. Explain how *Mutation Testing* could identify the flaw in their testing strategy.

(b) Describe the difference between a "Survived" mutant and a "Killed" mutant. What does a high survival rate tell you about your test suite?

(c) Why is mutation testing rarely applied to very large codebases (e.g., millions of lines) in every CI build?

**Q3. Testing in the Real World: Isolation.** You are testing a system that charges a user's credit card via an external API (like Stripe).

(a) Why is it a bad idea to use the real Stripe API in your automated *Unit Tests*?

(b) Explain the difference between a *Mock* and a *Stub* in this context.

(c) How does the *discrete state* nature of software (from L1) make testing the "Success" vs "Insufficient Funds" states both easier and more dangerous?

**Q4. Reliable Delivery: Deployment Strategies.** A bank is launching a new mobile payment feature.

(a) Contrast a **Blue-Green** deployment with a **Canary** deployment. Which is more appropriate for a high-risk database migration?

(b) How do **Feature Flags** (Toggles) allow for "Progressive Delivery"?

(c) Explain the concept of the **Error Budget**. If a team has exhausted their budget, how does this affect their deployment schedule?

**Q5. The Maintenance Reality.** A legacy banking system is 20 years old.

(a) Use Lehman's Laws to explain why the bank cannot simply "freeze" the code to save money.

(b) Categorize these tasks: (i) fixing a rounding error in interest rates, (ii) adding support for a new government tax API, (iii) rewriting a data-fetching loop to use a more efficient algorithm, (iv) migrating the database from an on-premise mainframe to the cloud.

(c) Why does the 80/20 rule suggest that the bank should invest in a high-quality automated test suite *now*, despite the high upfront cost?

**Q6. SRE and Chaos Engineering.**

(a) Describe the "Chaos Monkey" philosophy. Why would an engineering team intentionally break their own production systems?

(b) Identify the **Four Golden Signals** of monitoring. Why is *Saturation* often harder to measure than *Latency*?

(c) Explain the difference between *Monitoring* (Known Unknowns) and *Observability* (Unknown Unknowns).

**Q7. Legacy Evolution: The Strangler Fig.** You are tasked with replacing a 30-year-old COBOL mainframe system with a modern microservice architecture.

(a) Why is a "Big Bang" rewrite (replacing everything at once) considered high-risk?

(b) Describe the steps of the **Strangler Fig Pattern** in this context.

(c) How does *Hyrum's Law* complicate the migration of long-standing legacy systems?

**Q8. Software Archaeology and API Design.**

(a) You are tasked with modifying a critical 500-line function written in 2012. List three steps of "Software Archaeology" you would take to minimize the risk of a catastrophic failure.

(b) You are releasing version 2.4.1 of a library. You want to change a function signature from `process(data)` to `process(data, options)`. According to **SemVer**, what should the new version number be? Why?

(c) Why is an API described as a "Contract"? What happens to the software ecosystem when this contract is broken?

**Q9. API Evolution & Deprecation.** You maintain a weather API. You want to move from an old XML format to a modern JSON format.

(a) Why can't you just turn off the XML endpoint tomorrow?

(b) Describe a 3-step *Deprecation Path* that follows SemVer and minimizes customer disruption.

(c) How does a *Breaking Change* in your API affect the CI/CD (Continuous Integration) pipelines of your customers?

**Q10. AI-Augmented Engineering.**

(a) A developer uses AI to generate a function for parsing user-uploaded CSV files. The AI produces code that works but uses a library version with a known security vulnerability. Why did the AI do this, and whose responsibility is the bug?

(b) Contrast the role of a junior engineer in 2010 vs. 2026. How has the required *cognitive skill set* shifted?

(c) Explain the "Reasoning Ceiling" of LLMs. Why can't an AI currently replace a Senior Software Architect?

**Q11. The Human in the AI Loop.** A developer uses an AI-first IDE (like Cursor) to generate a complex multi-file refactor. The AI correctly updates all the logic but forgets to update the database migration scripts.

(a) Explain the concept of the *Context Window* and how it led to this specific failure.

(b) Propose a "Checklist for AI Review" that a Senior Engineer should follow before hitting "Merge" on an AI-generated PR.

(c) Why is *Software Archaeology* (knowing the history of the code) still relevant if an AI can summarize a file in 2 seconds?