Software Metrics - Tools and Methodology Exam (Andrei Kovalenko, 2318083)

**Test (questions)**

1. Design phase — SDLC stage that turns requirements into a blueprint: architecture, components, interfaces, data models, algorithms, constraints, and NFRs (e.g., performance, security). Outputs: HLD/LLD, diagrams, API specs.
2. Determine the list of quality requirements — Elicit and prioritize NFRs (reliability, performance, security, usability, maintainability, portability) via stakeholders, scenarios, standards (e.g., ISO/IEC 25010), and risks; define measurable acceptance criteria (e.g., “p95 latency ≤ 200 ms”).
3. Drawback of code coverage measurement — High coverage ≠ good tests. Coverage can be gamed, ignores assertion quality and data/value partitions, and doesn’t prove requirements are tested. It may give false confidence.
4. Requirements traceability — Ability to link each requirement ↔ source ↔ design ↔ code ↔ tests ↔ defects. Supports completeness checks and impact analysis (forward/backward traceability).
5. Software Design — The structured decomposition of a system into modules/components and interactions using architectures/patterns to meet functional & quality requirements with explicit trade-offs.
6. Goal/Question/Metric (GQM) paradigm — Define a Goal, derive Questions that assess it, then pick Metrics that answer those questions. Ensures measurement is purposeful and interpretable.
7. Organization measures — Company-level KPIs for process & outcomes: lead/cycle time, throughput, defect density, MTTR/MTBF, customer satisfaction/NPS, predictability (planned vs delivered), cost variance, employee turnover.
8. Cohesion — How tightly related a module’s responsibilities are. Higher is better (e.g., functional cohesion); low (coincidental/logical) signals a “grab-bag” module.
9. Downtime — Interval when a service is unavailable (planned or unplanned). Ties to availability: A=uptime/uptime+downtime​. Related: MTBF, MTTR.
10. Depth in Tree (DIT) — CK metric: longest path from a class to the root of the inheritance tree. Larger DIT ⇒ more reuse but higher complexity/coupling and harder understanding/testing.