Answers to Guide 14

* Software Evolution:
  + What is software evolution?
    - Refers to the process of developing software initially, then repeatedly updating it for various reasons.
    - Focuses on adaptation and migration.
    - Ensuring functional relevance, reliability, and flexibility of the system.
  + Is the concept of software evolution compatible with agile methods?
    - Yes, they stem from maintenance-like activities in and around web-based technologies, where the bulk of the capability comes from frameworks and standards.
  + What are the four types of software maintenance?
    - Corrective maintenance: reactive modification of a software product performed after delivery to correct discovered problems.
    - Adaptive maintenance: modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.
    - Perfective maintenance: modification of a software product after delivery to improve performance or maintainability.
    - Preventive maintenance: modification of a software product after delivery to detect and correct latent faults in the software product before they become effective faults.
  + Be generally familiar with Lehman’s laws of software evolution, particularly numbers 1 & 5-7
    - Continuing change – an E-type system must be continually adapted or it becomes progressively less satisfactory.
    - Increasing complexity – as an E-type system evolves, its complexity increases unless work is done to maintain or reduce it.
    - Self-regulation: E-type system evolution processes are self-regulating with the distribution of product and process measures close to normal.
    - Conservation of Organizational Stability – the average effective global activity rate in an evolving E-type system is invariant over the product’s lifetime (work rate)
    - Conservation of Familiarity– as an E-type system evolves, everything associated with it, must maintain mastery of its content and behavior to achieve satisfactory evolution. Excessive growth diminishes mastery, hence average incremental growth remains invariant.
    - Continuing Growth – the functional content of an E-type system must be continually increased to maintain user satisfaction over its lifetime.
    - Declining Quality – the quality of an E-type system will appear to be declining unless it is rigorously maintained and adapted to operational environment changes.
    - Feedback system – E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base.
* UML – Deployment Diagrams:
  + Is this diagram type a structural or behavioral diagram?
    - A structural diagram that models the physical deployment of artifacts on nodes.
    - i.e. hardware components (nodes), software components (artifacts) that run on each node, and how the different pieces are connected.
  + What are the basic building blocks of a deployment diagram, that is, what do the nodes and arcs represent?
    - Nodes appear as boxes and artifacts allocated to each node appear as rectangles within the boxes.
    - Nodes can have sub-nodes, which appear as nested boxes.
    - Arcs represent the different physical or logical connections between each node and artifact.
    - Node may conceptually represent multiple physical nodes.
    - Two types of nodes – device node and execution environment node.
  + Does it make sense to label an arc? If so, what would such a label specify? If not, why not?
    - It could make sense to label an arc if you wish to specify more details concerning the connection between that node and another node.
    - For example, the type of network connection between the two physical nodes (wireless, Ethernet, fiber-optic, etc.)
* An interview with Fred Brooks: Quoted Often, Followed Rarely:
  + Does Brooks’ Law apply only to software development?
    - No, it applies to many different fields in general as human nature doesn’t change and there are always issues in communication between people working together in teams.
  + What advice does he have for young software managers? Is it more traditional or more agile?
    - Definitely more towards the AGILE side of matters.
    - Should have relatively clear goals, know which direction to take to achieve those goals, and be on the path to those goals.
    - Don’t sit still and do nothing.
    - “The problem is not to make the right decision; it’s to make the decision right”
  + How does Brooks view open-source development?
    - Advantage of early, large-scale testing, to find bug, and a culture in which people communicate fixes to each other.
    - Advantage of multiple version of same component is built, and marketplace decides which one to implement.
    - Disadvantage of versioning issues if there are multiple versions of a single component resulting in compatibility issues.
    - Works best when the clients are the developers.
    - Economically, difficult to compensate people except with prestige.