**SW Metrics exam**

**Pieskov Viacheslav**

**Question 1:**  
**Design phase**

Design phase is the stage in software development where the overall system architecture is conceptualized and documented. It involves identifying core components, defining how they interact, and choosing appropriate technologies. The main outcome is a set of design artifacts - such as architectural diagrams and component interfaces - that guide the implementation and ensure consistency across the development team.

**Question 2:**  
**Determine the List of Quality Requirements**

Determining the list of quality requirements involves gathering and analyzing stakeholder needs to identify non-functional attributes (such as reliability, performance, security, and usability). These requirements are then documented with clear acceptance criteria to ensure that the final product meets the defined quality standards. This process provides a solid basis for measuring and validating quality throughout the development lifecycle.

**Question 3:**

**Drawback of Code Coverage Measurement**

While code coverage metrics indicate how much of the source code is executed during testing, a key drawback is that high coverage alone does not guarantee meaningful tests. It is possible to have 100% coverage with trivial or incomplete assertions, resulting in missed edge cases and inadequate validation of software functionality. Thus, focusing solely on coverage can foster a false sense of test effectiveness.

**Question 4:**

**Requirements Traceability**

Requirements Traceability refers to the ability to link each software requirement to its corresponding design elements, implementation, and test cases. It ensures that no requirement is overlooked or duplicated and provides transparency for stakeholders to track progress and manage change. By maintaining clear traceability, teams can quickly see which parts of the system satisfy specific requirements and confirm that nothing critical is omitted.

**Question 5:**

**Software Design**

Software Design encompasses all activities related to defining both high-level architecture and detailed internal structures of the software system. It involves choosing design patterns, class structures, data models, and interfaces. Good design aims to create a coherent, maintainable, and scalable solution by ensuring each component has a clear purpose and well-defined relationships with other components.

**Question 6:**

**Goal/Question/Metric (GQM) Paradigm**

The GQM paradigm is a structured approach to defining and using metrics to assess software quality. It begins with a specific Goal (what needs to be achieved), which is broken down into Questions that illuminate how the goal can be reached or measured. Each question is answered by one or more Metrics that provide quantitative data. Through this method, teams can systematically evaluate progress and make data-driven decisions.

**Question 7:**

**Organization Measures**

Organization Measures are high-level metrics used to evaluate and improve performance at the organizational level. Typical examples include lead time (time from concept to production), productivity indicators (such as story points completed per iteration), defect rates, and employee satisfaction scores. These measures guide strategic planning, resource allocation, and process improvements across the entire organization.

**Question 8:**

**Cohesion**

Cohesion is the degree to which the elements within a module or class are functionally related to each other. High cohesion means the module has a single, well-defined responsibility, making it easier to understand, test, and maintain. Low cohesion can lead to tangled responsibilities and increased complexity. Aiming for high cohesion in software design typically results in more robust and maintainable systems.

**Question 9:**

**Downtime**

Downtime refers to the periods when a system is non-operational or inaccessible to end-users, often caused by maintenance, upgrades, or unexpected failures. Minimizing downtime is crucial for user satisfaction, as prolonged unavailability can violate service-level agreements (SLAs) and result in financial or reputational damage. Tracking downtime metrics helps organizations refine their processes to achieve higher system reliability.

**Question 10:**

**Depth in Tree (DIT)**

Depth in Tree (DIT) is an object-oriented metric that measures the number of levels of inheritance from the root (base) class to the current class. A deeper inheritance hierarchy can complicate maintenance and understanding, as developers must consider behaviors defined in multiple ancestor classes. While some inheritance can improve code reuse, excessively deep hierarchies often signal the need for rethinking the design structure.