1.Most of the methods of Software Estimation is empirically defined-True or false. If false give correct answer.

Ans:True. Most methods of software estimation are empirically defined, meaning they are based on practical experience, historical data, and observations rather than theoretical or mathematical models. Software estimation involves predicting the effort, time, and resources required for software development, and empirical methods often leverage past project data, expert judgment, and heuristics to make predictions for new projects. While there are some theoretical models, such as COCOMO (Constructive Cost Model) and function point analysis, many estimation techniques rely on empirical data and practical insights gained from previous software development projects.

2.What does Maintainability Index mean?

Ans:The Maintainability Index is a software metric that provides a quantitative measure of how maintainable and understandable a software system is. It is used to assess the ease with which a software system can be maintained, enhanced, or fixed over time. The higher the Maintainability Index, the more maintainable the software is considered to be.

The Maintainability Index is typically calculated based on various factors and metrics, including:

1. **Cyclomatic Complexity (CC):** This measures the complexity of the control flow in the software. It counts the number of linearly independent paths through the code.
2. **Lines of Code (LOC):** The total number of lines of code in the software.
3. **Halstead Volume:** This is a software metric introduced by Maurice Halstead that combines various aspects of program length, vocabulary, and complexity.
4. **McCabe's Cyclomatic Complexity:** Similar to the CC, this metric measures the complexity of the code by counting the number of linearly independent paths.

3.Define Index of Variation.

Ans:

The term "Index of Variation" typically refers to a statistical measure that quantifies the degree of variation or dispersion in a set of values. There are different indices of variation used in various contexts, but one common example is the Coefficient of Variation (CV).

The Coefficient of Variation is a standardized measure of relative variability and is calculated as the ratio of the standard deviation to the mean of a set of values. It is expressed as a percentage:

$$CV=((\text{Standard Deviation})/\text{mean})\times 100$$

The Coefficient of Variation is useful when comparing the variability of datasets with different units or scales. A higher CV indicates greater relative variability, while a lower CV suggests more consistent or less variable data.

Other indices of variation may be specific to certain fields or types of data. It's essential to specify the context or the particular index of variation you are referring to for a more accurate definition.

4.What is the most common measurement of software?

Ans.Several measurements are commonly used in the field of software engineering to assess various aspects of software development, quality, and performance. Here are some of the most common measurements in software engineering:

1. **Lines of Code (LOC):** This measures the size of the software by counting the number of lines of code. While it's a simple and widely used metric, it doesn't necessarily reflect the quality or functionality of the software.
2. **Cyclomatic Complexity (CC):** CC is a measure of the complexity of a program based on the number of linearly independent paths through its source code. It helps assess the complexity of control flow in a program.
3. **Function Points (FP):** Function points are a unit of measurement to express the amount of business functionality an information system provides. They consider factors like inputs, outputs, inquiries, and files.
4. **Halstead Metrics:** Introduced by Maurice Halstead, these metrics include measures of program length, vocabulary, volume, difficulty, and effort. They provide insights into software complexity and effort required for development.
5. **Effort Estimation:** Metrics related to estimating the effort required for software development, often measured in person-hours or person-months.
6. **Defect Density:** This metric calculates the number of defects (bugs) per unit of code, such as defects per KLOC (thousand lines of code).
7. **Maintainability Index:** A composite metric that combines various factors, including cyclomatic complexity, lines of code, and Halstead volume, to assess how maintainable a software system is.
8. **Code Churn:** The frequency and extent of changes made to the source code over time, which can provide insights into the stability and evolution of a software project.
9. **Code Review Metrics:** Metrics related to the code review process, such as the number of issues identified, resolution time, and review coverage.

5.What is Coupling between objects?

Ans:Coupling between objects refers to the degree of dependence or connection between different objects in a software system. In object-oriented programming (OOP), objects are instances of classes, and coupling describes how much one class or object relies on another. There are different types of coupling, and the goal is often to minimize or control coupling for better software design and maintainability. Here are some common types of coupling between objects:

1. **Low Coupling (Loose Coupling):** This is the ideal scenario where objects are relatively independent of each other. Changes in one object have minimal impact on other objects. Loose coupling is desirable because it promotes modularity, flexibility, and ease of maintenance.
2. **High Coupling (Tight Coupling):** In a tightly coupled system, objects are highly dependent on each other. Changes in one object may have a significant impact on other objects. Tight coupling can lead to difficulties in maintenance and make the system less flexible.
3. **Data Coupling:** Objects are loosely coupled if they communicate through well-defined interfaces and exchange only necessary data. Data coupling is a form of low

coupling where objects share data but are not overly dependent on each other's internal details.

4. **Control Coupling:** This occurs when one object controls the behavior of another object, typically by passing control information or instructions. Minimizing control coupling helps maintain independence between objects.
5. **Temporal Coupling:** Objects are temporally coupled if they depend on the timing or sequence of operations. Reducing temporal coupling involves ensuring that objects do not rely on specific timing or order of execution.
6. **Content Coupling:** This is a form of tight coupling where one object relies on the internal implementation details of another. It is generally advisable to minimize content coupling to enhance encapsulation and maintainability.

6.What do you understand by the term "Manage by numbers"?

Ans:"Manage by numbers" is a management approach that emphasizes the use of quantitative data and metrics to make decisions, evaluate performance, and guide organizational strategies. In this context, "numbers" refer to key performance indicators (KPIs), metrics, and other quantitative measures that provide insights into various aspects of an organization's operations.

Here are some key aspects of the "manage by numbers" approach:

1. **Data-Driven Decision Making:** The approach involves making decisions based on data and empirical evidence rather than relying solely on intuition or subjective judgments. Managers use quantitative metrics to analyze trends, identify patterns, and inform strategic choices.
2. **Performance Measurement:** It involves systematically measuring and monitoring performance through numerical metrics. These metrics can include financial indicators, operational metrics, customer satisfaction scores, employee productivity, and more.
3. **Goal Setting and Monitoring:** "Manage by numbers" often includes setting specific, measurable, achievable, relevant, and time-bound (SMART) goals. Progress toward these goals is regularly monitored using numerical metrics.
4. **Continuous Improvement:** The approach emphasizes the continuous monitoring and improvement of processes and outcomes. By analyzing numerical data, organizations can identify areas for improvement and implement changes to enhance efficiency and effectiveness.
5. **Accountability:** Quantitative metrics provide a basis for holding individuals, teams, and departments accountable for their performance. Clear expectations and measurable goals help in assessing whether objectives are met.
6. **Benchmarking:** Organizations often compare their performance metrics with industry benchmarks or best practices. Benchmarking allows them to understand where they stand in comparison to peers and identify areas where they can improve.
7. **Risk Management:** The use of quantitative data enables organizations to assess and manage risks more effectively. By identifying potential risks and their impact, managers can make informed decisions to mitigate or address them.

While the "manage by numbers" approach offers many benefits, it's important to note that not all aspects of organizational performance can be captured by quantitative metrics alone. Qualitative factors, such as organizational culture, innovation, and employee satisfaction, also play crucial roles and may require complementary management approaches.
The success of the "manage by numbers" approach relies on selecting relevant and meaningful metrics, interpreting data accurately, and using insights to inform strategic decisions and actions.

7.Assume you have three modules-A, B, and C - which have sines of 10 KLOC, 24 KLOC, and 50 KLOC, respectively. Now find the Deviation and Variance from the data.
Ans:

To calculate the deviation and variance, we need to know the average size of the modules. Let's assume that n represents the number of modules, x_i represents the size of each module in KLOC, and $\bar{x}$ represents the average size.

1. **Calculate the Average Size ($\bar{x}$):**
   $\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$
   For the given data:
   $\bar{x} = \frac{10+24+50}{3}$
2. **Calculate the Deviation (d_i):**
   The deviation of each module from the average is calculated as:
   $d_i = x_i - \bar{x}$
   Apply this formula for each module:
   - $d_A = 10 - \bar{x}$
   - $d_B = 24 - \bar{x}$
   - $d_C = 50 - \bar{x}$
3. **Calculate the Variance (Var):**
   The variance is the average of the squared deviations:
   $\mathrm{Var} = \frac{\sum_{i=1}^{n} d_i^2}{n}$
   Apply this formula for the given data:
   $\mathrm{Var} = \frac{d_A^2+d_B^2+d_C^2}{3}$

Let's calculate these values:

$$\bar{x} = \frac{10+24+50}{3} = \frac{84}{3} = 28$$

$$d_A = 10 - 28 = -18$$
$$d_B = 24 - 28 = -4$$
$$d_C = 50 - 28 = 22$$

$$\text{Var} = \frac{(-18)^2+(-4)^2+(22)^2}{3}$$

Now, calculate the variance.

$$\text{Var} = \frac{324+16+484}{3} = \frac{824}{3}$$

So, the average size ($\bar{x}$) is 28 KLOC, the deviations are -18 KLOC, -4 KLOC, and 22 KLOC for modules A, B, and C, respectively, and the variance (Var) is $\frac{824}{3}$.

8.Briefly describe how Agresti-Card-Glass System Complexity Metric works.

Ans:The Agresti-Card-Glass System Complexity Metric is a software complexity metric designed to assess the complexity of a software system based on its structure and interactions. It takes into account various factors, including the number of decision nodes, decision density, and cyclomatic complexity. The metric was proposed by researchers Louis Agresti, Stuart Card, and Robert L. Glass.

Here's a brief overview of how the Agresti-Card-Glass System Complexity Metric works:

1. **Decision Nodes:**
   - Decision nodes refer to points in the code where the control flow can branch based on a decision (e.g., if statements, switch statements).
   - The metric counts the number of decision nodes in the software system.
2. **Decision Density:**
   - Decision density is a measure of how densely decisions are distributed throughout the code.
   - It is calculated by dividing the number of decision nodes by the number of lines of code (LOC).

$$Decision\ Density = \frac{Number\ of\ Decision\ Nodes}{Number\ of\ Lines\ of\ Code\ (LOC)}$$

3. **Cyclomatic Complexity:**
    - Cyclomatic complexity (CC) is a traditional metric that measures the number of linearly independent paths through the source code.
    - It is calculated using the formula $CC=E-N+2P$, where E is the number of edges in the flow graph, N is the number of nodes, and P is the number of connected components (regions).
4. **Integration of Metrics:**
    - The Agresti-Card-Glass metric integrates decision nodes, decision density, and cyclomatic complexity into a single metric for system complexity.
    - The exact formula for the metric might vary, but it typically involves combining these factors to provide a comprehensive assessment of the software's complexity.

$$\text{Agresti-Card-Glass Metric} = a \times \text{Decision Density} + b \times \text{Cyclomatic Complexity}$$

- The coefficients a and b are used to adjust the relative importance of decision density and cyclomatic complexity in the overall complexity metric.

The Agresti-Card-Glass System Complexity Metric provides a more nuanced view of software complexity by considering both decision density and cyclomatic complexity. It can help software developers and managers identify areas of code that might be more prone to errors, more challenging to maintain, or have a higher potential for bugs. The goal is to encourage the development of more modular and maintainable software systems.

9.What are the problems that may arise if you measure a system by counting the lines of codes that it has?

Ans:Measuring a software system solely by counting lines of code (LOC) has several limitations and can lead to inaccurate or misleading assessments of software complexity, quality, and productivity. Here are some problems that may arise when relying on LOC as a primary metric:

1. **Code Duplication:** LOC does not distinguish between original code and duplicated code. A system with a high number of lines of code may be inflated if there is extensive code duplication, leading to an overestimation of its complexity.
2. **Code Efficiency:** LOC doesn't provide information about code efficiency or performance. A shorter piece of code might be more efficient than a longer one, and vice versa. Efficiency is not reflected in the LOC metric.
3. **Language Variability:** Different programming languages have different syntax and structure, making LOC a poor measure for comparing systems written in different languages. For instance, a task that requires fewer lines in one language might take more lines in another.
4. **Code Comments and Whitespace:** LOC includes comments and whitespace, which are essential for code readability but don't contribute to the functional logic of the program. A heavily commented codebase may appear larger than it actually is.
5. **Algorithmic Complexity:** LOC does not capture the inherent complexity of algorithms or the overall design of the software. Two programs with similar functionality may have different LOC counts if they use different algorithms or design approaches.
6. **Maintenance Challenges:** A system with a high LOC count may be challenging to maintain and understand. However, simply reducing LOC without considering other factors may not necessarily improve maintainability or code quality.

7. **Productivity Issues:** Measuring productivity based on LOC can lead to counterproductive behavior, such as encouraging developers to write longer code to meet arbitrary targets. It may not accurately reflect the true value or effort invested in the development process.
8. **Focus on Quantity over Quality:** Relying solely on LOC can incentivize developers to prioritize quantity over code quality. Quality aspects like modularity, readability, and maintainability are crucial but are not captured by the LOC metric.
9. **Ignoring Non-Code Artifacts:** The LOC metric does not account for non-code artifacts such as documentation, test cases, and configuration files, which are integral to a software project but are not reflected in the LOC count.
10. **Incomplete Picture of Software Complexity:** Software complexity involves various dimensions, including data structures, dependencies, and interactions, which are not adequately represented by LOC alone.

To address these limitations, it's important to use a combination of metrics, including cyclomatic complexity, function points, and other software engineering metrics, to provide a more comprehensive and accurate assessment of software quality and complexity.

10.3 SD means what percentage of the population?

Ans:"3 SD" typically refers to a value that is three standard deviations away from the mean in a normal distribution. In a normal distribution (also known as a bell curve or Gaussian distribution), about 99.7% of the data falls within three standard deviations of the mean. This is known as the "68-95-99.7 rule,".

11.Give example of nominal scale.

Ans:Nominal scale is a type of categorical measurement scale that categorizes data into distinct, non-ordered categories or groups. The categories in a nominal scale have no inherent order or numerical significance.

12.What are Software Development Metaphors?

Ans:

1. **Building Construction:**
    - Like building a house, software development involves planning, designing, and assembling different parts to create a final product.
2. **Cooking Recipes:**
    - Software development is like following a recipe. Developers use specific steps and ingredients (code) to create a software "dish."
3. **Gardening:**
    - Similar to tending a garden, developers plant code "seeds," nurture them, and ensure a healthy environment for the software to grow.
4. **Sports Team:**
    - In software development, a team works together like a sports team, with specific roles, a game plan (project plan), and the goal of achieving success through collaboration.
5. **Travel and Navigation:**
    - Software development is compared to a journey where developers plan routes (project phases), encounter challenges, and adapt to changes along the way.
6. **Storytelling:**
    - Writing code is like telling a story. Each line of code contributes to the narrative, and the goal is to create a coherent and readable story.
7. **Chess Game:**

- Software development is strategic, much like a chess game. Developers make moves (coding decisions) to reach their goal, requiring foresight and planning.

13.Having a deep inheritance tree is not a good software development practice. True or false?

Ans:True. Having a deep inheritance tree, often referred to as deep class hierarchies or deep inheritance hierarchies, is generally considered a practice to avoid in software development.

14.What is GQM^1?

Ans:GQM^1 (Goal-Question-Metric) is a framework used in software engineering and other disciplines for defining and measuring project goals. It is an extension of the GQM model, which was introduced by Victor Basili and his colleagues to establish a systematic and measurable approach to software measurement.

In the GQM^1 framework:

- **Goal (G):** This is the overall objective or purpose that an organization or project aims to achieve. Goals are typically broad and strategic, providing a high-level direction for the project.
- **Question (Q):** Questions are derived from the goals and represent specific aspects that need to be understood or assessed to achieve the goals. Questions break down the goals into more detailed and measurable components.
- **Metric (M):** Metrics are the quantitative or qualitative measures used to answer the questions. Metrics provide the data and information needed to assess progress toward the goals.

The GQM^1 framework helps organizations define, structure, and implement measurement programs by establishing a clear hierarchy from broad goals to specific metrics. It promotes a systematic and goal-oriented approach to measurement, ensuring that the metrics collected align with the organization's overall objectives.

Here is an example of how GQM^1 works:

Goal (G): Improve software quality.

Questions (Q):

1. What is the defect density in the code?
2. How many defects are found during testing?
3. How satisfied are users with the software's performance?

Metrics (M):

1. Defect Density (measured as defects per KLOC - thousand lines of code).
2. Defect Count during Testing.
3. User Satisfaction Score (measured through surveys or feedback).

In this example, the goal is to improve software quality. The questions break down aspects of quality that are measurable, and the corresponding metrics provide the quantitative or qualitative data needed to answer those questions.

15.Why is Software Measurement a challenging task?

Ans:Software measurement is a challenging task due to several factors, including the nature of software itself, the complexity of development processes, and the dynamic nature of the software industry. Here are some reasons why software measurement can be challenging:

1. **Intangibility of Software:**

- Software is intangible, making it difficult to measure directly. Unlike physical products, the characteristics and quality of software are not easily observable or quantifiable.

2. **Subjectivity in Quality:**
   - Software quality is subjective and depends on user expectations, requirements, and context. Measuring aspects like usability, maintainability, and user satisfaction involves interpreting qualitative data, which can be challenging.
3. **Diversity of Software Types:**
   - Software comes in various types, including desktop applications, web applications, mobile apps, embedded systems, and more. Each type has unique characteristics and requirements, making it challenging to apply a one-size-fits-all measurement approach.
4. **Changing Requirements:**
   - Software requirements often change during the development process. This dynamic nature can make it challenging to establish stable and consistent metrics, as the goals and objectives may evolve over time.
5. **Interdependencies:**
   - Software is composed of interdependent components, and changes to one part can affect others. Measuring the impact of changes and understanding these interdependencies requires sophisticated measurement techniques.
6. **Human Factor:**
   - Software development involves human creativity, collaboration, and decision-making. Human factors, such as the skill and experience of developers, can influence the quality and productivity of software development, making it challenging to quantify.
7. **Evolution of Technology:**
   - Rapid advancements in technology lead to changes in software development practices, tools, and methodologies. Keeping measurement practices up-to-date with evolving technologies is a continuous challenge.
8. **Complexity of Development Processes:**
   - Modern software development processes, such as Agile or DevOps, involve iterative and collaborative approaches. Measuring progress and success in these dynamic and complex environments requires adapting measurement practices accordingly.
9. **Multiple Stakeholders:**
   - Software projects involve various stakeholders with diverse interests and perspectives, including developers, managers, customers, and end-users. Balancing the needs and expectations of different stakeholders in measurement can be complex.
10. **Trade-Offs between Metrics:**
    - There are often trade-offs between different metrics. For example, optimizing one aspect of software (e.g., speed) may negatively impact another aspect (e.g., maintainability). Balancing conflicting metrics is a challenging aspect of software measurement.
11. **Data Quality and Availability:**
    - Obtaining reliable and consistent data for measurement can be challenging. Data quality issues, incomplete information, and the availability of historical data may hinder the accuracy of measurements.

16.What is CMMI?

Ans:

CMMI stands for Capability Maturity Model Integration. It is a set of industry-recognized best practices for improving the processes involved in developing, managing, and maintaining software systems.

17.A matrice program is denoted by GQM with bottom-up approach - true or false.

Ans:False

18.how many types of measurement models are there? State the names.

Ans:

Goal-Question-Metric (GQM) Model:

Balanced Scorecard Model:

19.What 'Death March"?

Ans: In the context of project management, a "Death March" refers to a project that is characterized by an excessive workload, unrealistic deadlines, and extremely challenging conditions.

20.How to measure Effort according to Algorithmic model?

Ans:

21.Define different measurement scale typing along with the ways for measuring of central tendency.

Ans:Measurement scales, also known as levels of measurement, categorize data into different types based on the nature of the values. There are four main types of measurement scales: nominal, ordinal, interval, and ratio. Each scale has specific characteristics, and the choice of central tendency measures depends on the scale of measurement. Here's a brief overview of each scale and the associated measures of central tendency:

1. **Nominal Scale:**
    - **Characteristics:** Nominal scales categorize data into distinct categories or labels. The categories have no inherent order or numerical value.
    - **Examples:** Categories like colors, names, or types.
    - **Measures of Central Tendency:** The most appropriate measure is the mode, which represents the most frequently occurring category.
2. **Ordinal Scale:**
    - **Characteristics:** Ordinal scales order data, but the intervals between the categories are not consistent or meaningful. The order indicates a relative ranking.
    - **Examples:** Rankings (e.g., 1st, 2nd, 3rd), Likert scales (e.g., strongly agree to strongly disagree).
    - **Measures of Central Tendency:** The median is the most appropriate measure for ordinal data. It represents the middle value when the data is ordered.
3. **Interval Scale:**
    - **Characteristics:** Interval scales have ordered categories with consistent and meaningful intervals between them. However, there is no true zero point.
    - **Examples:** Temperature measured in Celsius or Fahrenheit.
    - **Measures of Central Tendency:** Both the mean and median can be used for interval data. The mean is more commonly used but may be affected by extreme values.

4. **Ratio Scale:**
    - **Characteristics:** Ratio scales have ordered categories with consistent intervals, and they have a true zero point, meaning zero indicates the absence of the measured quantity.
    - **Examples:** Height, weight, income, age.
    - **Measures of Central Tendency:** The mean is the most appropriate measure for ratio data. The median can also be used, but the mean is often preferred.

Measures of Central Tendency:

- **Mean:** The average of all values. Suitable for interval and ratio scales.
- **Median:** The middle value when data is ordered. Suitable for ordinal, interval, and ratio scales.
- **Mode:** The most frequently occurring value. Suitable for nominal, ordinal, interval, and ratio scales.

Choosing the appropriate measure of central tendency depends on the nature of the data and the scale of measurement. It's important to consider both the type of data and the statistical properties of the measures when analyzing and summarizing data.

22.define a four level model for software reliability for using the metrics meta-model.

Ans:A four-level model for software reliability, utilizing the metrics meta-model, can be structured based on different aspects of the software development and maintenance process. The metrics meta-model is a framework for organizing and understanding metrics, considering aspects such as goals, questions, indicators, and metrics. Below is a conceptual outline of a four-level model for software reliability using the metrics meta-model:

Level 1: Goals

1. **Primary Goal:**
    - **Description:** Define the overarching goal for software reliability. This could be to ensure the system meets user expectations for reliability, minimizing the occurrence of failures and errors.
    - **Metrics:** Establish a high-level reliability metric, such as the overall failure rate or the percentage of error-free executions.

Level 2: Questions

2.1 **Stakeholder Questions:**

- **Description:** Identify questions that stakeholders, including end-users and project managers, may have regarding software reliability.
- **Metrics:** Develop specific metrics to address stakeholder questions, such as the mean time between failures (MTBF), user-reported issues, or system availability.

2.2 **Process Questions:**

- **Description:** Explore questions related to the software development and maintenance processes that impact reliability.
- **Metrics:** Metrics may include defect density, code churn, and the effectiveness of testing processes.

Level 3: Indicators

3.1 **Operational Indicators:**

- **Description:** Identify operational indicators that provide insights into the software's behavior during runtime.
- **Metrics:** Metrics could include system uptime, response time under stress, and the frequency of critical incidents in the production environment.

3.2 **Development Indicators:**

- **Description:** Establish indicators related to the development process that influence reliability.
- **Metrics:** Metrics may encompass code complexity, adherence to coding standards, and the effectiveness of code reviews in catching potential issues.

Level 4: Metrics

4.1 **Execution Metrics:**

- **Description:** Define specific metrics related to the execution of the software.
- **Metrics:** Metrics at this level could include failure rates, error rates, and mean time to failure (MTTF).

4.2 **Testing Metrics:**

- **Description:** Specify metrics related to the effectiveness of testing efforts in ensuring software reliability.
- **Metrics:** Metrics may involve code coverage, test pass rates, and the identification and resolution of critical defects during testing.

This four-level model aligns with the metrics meta-model by organizing metrics in a hierarchical manner, starting from high-level goals down to specific metrics that directly measure aspects of software reliability. It provides a structured approach to understanding, evaluating, and improving software reliability throughout the development life cycle. The specific metrics and indicators can be tailored based on the context of the software project and the goals of the stakeholders.