What are the important features of TestNG?

A TestNG incldes:

1. TestNG uses more **object-oriented and java features.**

2. TestNG supports **multiple types of Before/After annotations** to create test cases.

3. It provides different types of assertions that helps in checking actual and expected results.

4. It provides **priority and grouping features** by which test cases can be grouped and prioritized easily.

Unit 2

Technical metrics for software:
Software Quality, A framework for Technical software metrics, Metrics for the analysis model,
Metric for the design model, Metric for source code, Metric for testing, Metric for maintenance.
Software Reuse:
Management issues, The reuse process, Domain engineering, Building reusable components, Classifying and retrieving the components, Economics of software reuse.
Selenium testing framework:
Selenium Webdriver Script: JAVA Code Example[A] , selenium and automation test tools, TestNG in Eclipse for Selenium WebDriver[A], Annotations, Framework [A], Examples in Selenium, Use of JXL API in selenium.
15L

Various metrics related to the product development process are :

- **Metrics for analysis model:** Addresses aspects of system functionality, system size
- **Metrics for design model:** Include architectural design metrics, component-level design metrics.
- **Metrics for source code:** These assess source code complexity, maintainability
- **Metrics for testing:** Assist in designing efficient and effective test cases.
- **Metrics for maintenance:** These assess the stability of the software product

Metrics for the Analysis Model

1. Function Point (FP) Metric

is used to measure the :

- ✓ Functionality delivered by the system,
- ✓ Estimate the effort,
- ✓ Predict the number of errors, and
- ✓ Estimate the number of components in the system.

2. Lines of Code (LOC)

- Used for size estimation.
- LOC can be defined as the number of delivered lines of code, excluding comments and blank lines.
- LOC is used to predict program complexity, development effort, programmer performance.

3. Metrics for Specification Quality :

Characteristics include :

Specificity, completeness, correctness, understandability,

Verifiability, internal and external consistency, &achievability,

Traceability, modifiability, precision, and reusability.

Each of these characteristics can be represented by using one or more metrics. For example, if there are n_r requirements in a specification, then n_r can be calculated by the following equation.

$$n_r = n_f + n_{nf}$$

Where

n_f = number of functional requirements

n_{nf} = number of non-functional requirements.

Completeness of the functional requirements can be calculated by the following equation.

$$Q_C = n_u / [n_i * n_s]$$

Where

n_u = number of unique function requirements

n_i = number of inputs defined by the specification

n_s = number of specified state.

Metrics for design model:

The complexity of a system increases with increase in :

structural complexity, data complexity, and system complexity,

which in turn increases the integration and testing effort.

Design Structural Quality Index (DSQI),

which is derived from the information obtained from data and

architectural design.

To calculate DSQI, the following values must be determined.

Number of components in program architecture (S_1)

Number of components whose correct function is determined by the Source of input data (S_2)

Number of components whose correct function· depends on previous processing (S_3)

Number of database items (S_4)

Number of different database items (S_5)

Number of database segments (S_6)

Number of components having single entry and exit (S_7).

Module independence (D_2):

$D_2 = 1-(S_2/S_1)$

Modules not dependent on prior processing (D_3):

$D_3 = 1-(S_3/S_1)$

Database size (D_4):

$D_4 = 1-(S_5/S_4)$

Database compartmentalization (D_5):

$D_5 = 1-(S_6/S_4)$

Module entrance/exit characteristic (D_6):

$D_6 = 1-(S_7/S_1)$

Metrics for Coding :

n_l = number of distinct operators in a program

n_2 = number of distinct operands in a program

N_1 = total number of operators

N_2= total number of operands.

By using these measures, we can calculate overall program length, program volume, program difficulty, development effort.

Program length (N) can be calculated by using the following equation.

$$N = n_1 \log_2 n_l + n_2 \log_2 n_2.$$

Program volume (V) can be calculated as:

$$V = N \log_2 (n_1 + n_2).$$

Program volume depends on the :

Programming language used and represents the volume of information

(in bits) required to specify a program.

Volume ratio (L) can be calculated by using the following equation.

$$L = (2/n_1)^* (n_2/N_2).$$

Program difficulty level (D) can be calculated by using the following equation:

$$D = (n_1/2)^*(N_2/n_2).$$

Effort (E)can be calculated by using the following equations.

$$E = D * V.$$

Metrics for Software Testing

Function point can be used to estimate testing effort in term of :

- ❑ Errors discovered,
- ❑ Number of test cases required

cyclomatic complexity can be used as a metric in the basis-path testing to determine the number of test cases needed.

Using program volume (V) and program level (PL), effort (e)can be calculated by the following equations.

$$e = V / PL$$

Where

$$PL = 1/ [(n_1/2) * (N_2/n_2)]$$

For a particular module (z), the percentage of overall testing effort allocated can be calculated by the following equation.

$$\text{Percentage of testing effort } (z) = e(z)/\sum e(i)$$

Where, e(z) is calculated for module z with the help of equation (1). Summation in the denominator is the sum of Halstead effort (e) in all the modules of the system.

Metrics for Software Maintenance

For the maintenance activities, IEEE has proposed Software Maturity Index (SMI), which provides indications relating to the stability of software product. For calculating SMI, following parameters are considered.

- Number of modules in current release (M_T)
- Number of modules that have been changed in the current release (F_e)
- Number of modules that have been added in the current release (F_a)
- Number of modules that have been deleted from the current release (F_d)

$$SMI = [M_T – (F_a + F_e + F_d)]/M_T.$$

What is Software Testing Metrics?

A Metric is a quantitative measure of the degree to which a system, system component, or process possesses a given attribute.

Software testing metrics:

Quantitative approach to measure the quality and effectiveness of the software development and testing process.

Metrics can be defined

as "STANDARDS OF MEASUREMENT".Software

Metrics are used to measure the quality of the

project.

i.e. a Metric is a unit used for describing an

attribute.

Metric is a scale for measurement.

Examples of metric:

1. **Cyclomatic complexity:** quantitative measure of the number of linearly independent paths through a program's source code
2. **Function point analysis :** **functionality** delivered to its users
3. **Cohesion** : relationship within the module
4. **Coupling:** relationships between modules
5. **Number of Classes & Interfaces**

Importance of Metrics

- Metrics is used to improve the **quality of products and services.**
- Different metrics helps the teams to monitor the efficiency of the process and control them
- It provides the scope of improvement for current process.

Types of Metrics:

Using different types of metrics we measure the quality of the software.

A manual testing metrics comprises of two other metrics –

Base Metrics and Calculated Metrics.

- **Base Metrics:** It comprises the raw data captured during the testing process.

Examples of Base Metrics are:

– No. of test cases

– No. of test cases executed

Calculated Metrics:

It is obtained by converting the data that is gathered in Base Metrics into useful information.

examples of Calculated Metrics are:

– Test coverage
– Test efficiency

Types of Test Metrics

Process Metrics:

Used to quantify characteristics of the SW process. –

Usually related to events or things that occur.

–It defines areas of improvement in the application development and maintenance

Allows making changes while there is still a chance to have an impact on the project

Examples:

defects found in test :Like Defect density

requirements changes,

days to complete task.

Product Metrics:

It deals with the **quality** of the software product.

This metrics include the following −

- Mean Time to Failure
- Customer Problems
- Customer Satisfaction

Examples: code, design docs, test plan,

LOC, Defect detection rate

Project Metrics:

It can be used to **measure the efficiency** of a project team or any testing tools being used by the team members. It defines the ability to perform project execution

Like, **Productivity, Schedule,** estimates of SW development time based on past projects.

- Metrics provide:

1. **Quantitative measures**

    - Operational: cost, failure rate, change effort.
    - Intrinsic: size, complexity.
    - Complexity is calculated from source code
    - Complexity correlates with defects, maintenance costs.

2. Qualitative measures

- Good complexity metrics have three properties
  - **Descriptive:** objectively measure something
  - **Predictive:** correlate with something interesting
  - **Prescriptive:** guide risk reduction

Implementation:

Consider lines of code [LOC]:

Descriptive: yes, measures software size

Predictive, Prescriptive: no

Consider cyclomatic complexity

Descriptive: yes, measures decision logic

Predictive: yes, predicts errors and maintenance

Prescriptive: yes, guides testing and improvement

Software Reuse

✧systems are designed by composing existing components that have been used in other systems

✧Software engineering has been more focused on original development but it is now recognized that to achieve better software, **more quickly and at lower cost,** we need a design process that is based on systematic software reuse

✧There has been a major switch to **reuse-based development** over the past years

Reuse-based software engineering

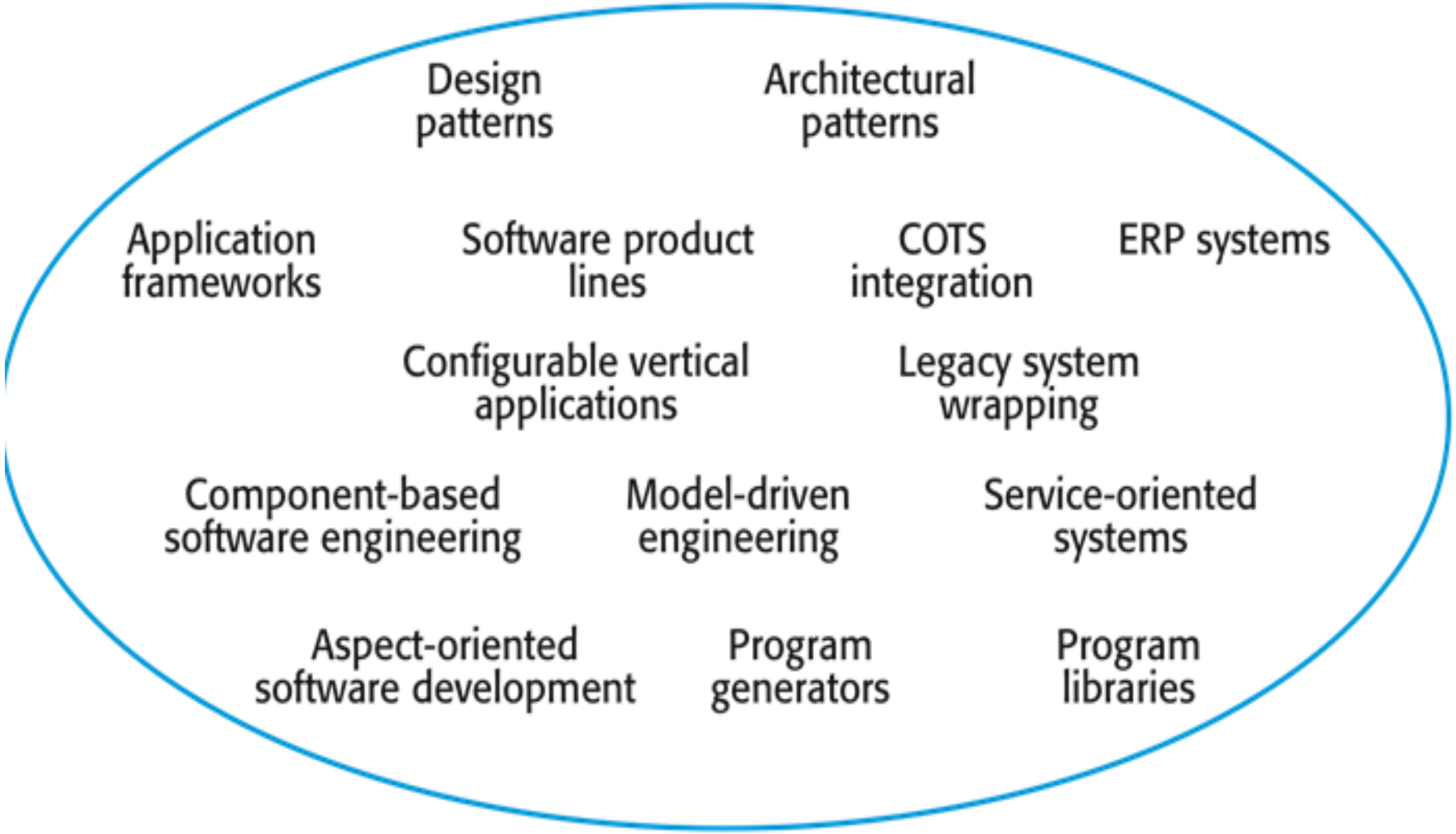
- Application system reuse
  - The complete system may be reused either without change into systems or by developing common architectures
- Component reuse
  - Components of an application from sub-systems to single objects may be reused.
- Object and function reuse
  - Software components that implement a single well-defined object or function may be reused

COTS product reuse

✧A commercial-off-the-shelf (COTS) product is a software system that can be adapted for different customers **without changing the source code of the system**

✧COTS systems have generic features and so can be **used/reused in different environments**

✧COTS products are adapted by using built-in configuration mechanisms that allow the functionality of the system to be tailored to specific customer needs



Approaches that support software reuse