innovate achieve lead

BITS Pilani
Pilani Campus

SE ZG501
Software Quality Assurance and Testing
Module – Session 10

Method 2) Three Point Estimation

- Three-Point estimation is one of the techniques that could be used to estimate a task.
- The simplicity of the Three-point estimation makes it a very useful tool for a Project Manager that who wants to estimate.
- In three-point estimation, **three** values are produced initially for every task based on **prior experience** or **best-guesses** as follows:

When estimating a task, the Test Manager needs to provide three values, as specified above.

The three values identified, estimate what happens in an **optimal state**, what is the **most likely**, or what we think it would be the **worst case scenario.**

Example: "**Create the test specification**", for bank website

You can estimate as following

The **best case** to complete this task is **120** man-hours (around 15 days). In this case, you have a talented team, they can finish the task in smallest time.

The **most likely** case to complete this task is **170** man-hours (around 21 days). This is a normal case, you have enough resource and ability to complete the task

The **worst case** to complete this task is **200** man-hours (around 25 days). You need to perform much more work because your team members are not experienced.

Now, assign the value to each parameter as below

$$a = 120 \quad m = 170 \quad b = 200$$

The effort to complete the task can be calculated using **double-triangular distribution** formula as follows-

$$E = (a + 4m + b)/6$$

PERT

$$E = (120 + 4 * 170 + 200)/6$$

$$E = 166.6\ (man - hours)$$

- Parameter E is known as **Weighted Average**.
- It is the estimation of the task "Create the test specification".

In the above estimation, you just determine a **possible** value, and not a **certain** one., we must know about the **probability** that the estimation is correct. You can use the other formula:

Since estimation involves uncertainty, it is essential to understand the **probability** that the estimated effort is correct

$$SD = (b - a)/6$$

$$SD = (200 - 120)/6$$

$$SD = 13.33\ (man - hours)$$

To measure this uncertainty, we use **Standard Deviation (SD)**, which quantifies how much the actual effort may **deviate from the estimated value**..

- Now you can conclude the estimation for the task “Create the test specification”
- To complete the task “Create the test specification” of Bank website, you need **166.6 ± 13.33** Man-hour (153.33 to 179.99 man-hour)

Step 4) Validate the estimation

- Once you create an aggregate estimate for all the tasks mentioned in the WBS, you need to forward it to the **management board**, who will **review** and **approve** it.
- The member of management board could comprise of the CEO, Project Manager & other stakeholders.
- The management board will review and discuss your estimation plan with you. You may explain them your estimation **logically** and **reasonably** so that they can approve your estimation plan.

Test estimation best practices

Add some buffer time

Account Resource planning in estimation

Use the past experience as reference

Stick to your estimation

Test Data Management

The process of planning, creating, and maintaining the datasets used in testing activities, ensuring that they are the right data for the right test case, in the right format, and available at the right time.

Test data is the set of input values used during the testing process of an application (software, web, mobile application, API, etc).

These values represent what a user would enter the system in a real-world scenario.

Testers usually can write a test script to automatically and dynamically identify the right type of values to put into the system and see how it responds to those data.

Example : Test data for the testing of a login page

- **Username** column and a **Password** column.
- A test script or automation testing tool can open the Login page, identify the Username field, the Password field, then input the values
- Can have hundreds to thousands of such credential pairs representing unique test scenarios.

Username	Password
user_123	Pass123!
testuser@email	Secret@321
admin_user	AdminPass#
jane_doe	JaneDoePass

- You can have hundreds to thousands of such credential pairs representing unique test scenarios.
- But having a huge database does not immediately mean all of it is high-quality.

Criteria to evaluate test data quality

Relevance: it makes sense to have test data that accurately reflects the scenario being tested.

Imagine testing the response of the login page when users enter the wrong set of credentials, but the test data being used is actually the correct, stored-in-database credentials. This returns inaccurate results.

Availability: what's the point of having thousands of relevant data points yet you can't retrieve them for testing activities?

Usually QA teams have clearly defined role-based access for test data, so TDM activities are also about assigning the right level of access to the right personnel.

Updated: software constantly changes, bringing with it new complexities and dependencies.

The responsibility of QA teams is to be aware of those updates and make changes to the test data accordingly to ensure that results accurately reflect the current state of the software.

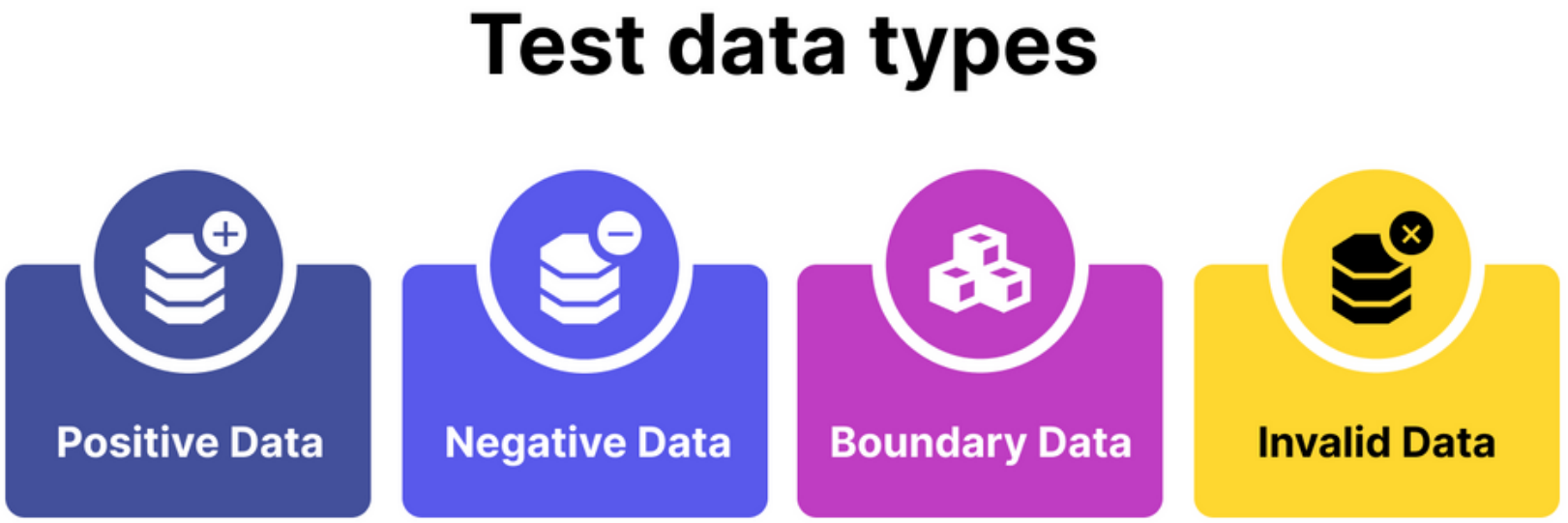
Compliance: aside from the technical aspects, we should never forget compliance requirements.

QA teams sometimes leverage directly production data for testing activities due to its instant availability, but production data is a tricky domain: it may contain confidential information protected by GDPR, HIPAA, PCI, or other data privacy focused policies.

PCI DSS, HIPAA, and GDPR are all security standards that protect sensitive information, but they differ in their scope, purpose, and enforcement.

- **Payment Card Industry Data Security Standard** is a set of security standards that protect cardholder data.
- **Health Insurance Portability and Accountability Act** is a federal law that protects patient health information (PHI)
- **General Data Protection Regulation** is a law that protects the personal data of EU citizens.

Types of Test Data



- **Positive Test Data:** this type of data consists of input values that are **valid and within the expected range** and is designed to test how the system behaves under expected and normal conditions. Examples: a set of valid username and password that allows users to login to their account page on an eCommerce site.

- **Negative Test Data:** in contrast with positive data, negative test data consists of input values that are invalid, unexpected, or outside the specified range. It is designed to test how the system behaves when users do something out of the “correct” pathintended.
- Examples: a set of username and password that is too long.

- **Boundary Test Data:** these are values at the edges or boundaries of acceptable input ranges chosen to assess how the system handles inputs at the upper and lower limits of the allowed range.

- **Invalid Test Data:** these are data that does not accurately represent the real-world scenarios or conditions that the software is expected to encounter. It does not conform to the expected format, structure, or rules within a given context.

🎯 Break-Time Brain Buster!

A login page consistently displays a "Login Successful" message, even when incorrect credentials are entered during testing.

Which of the following best explains the likely cause, based on test data quality principles?

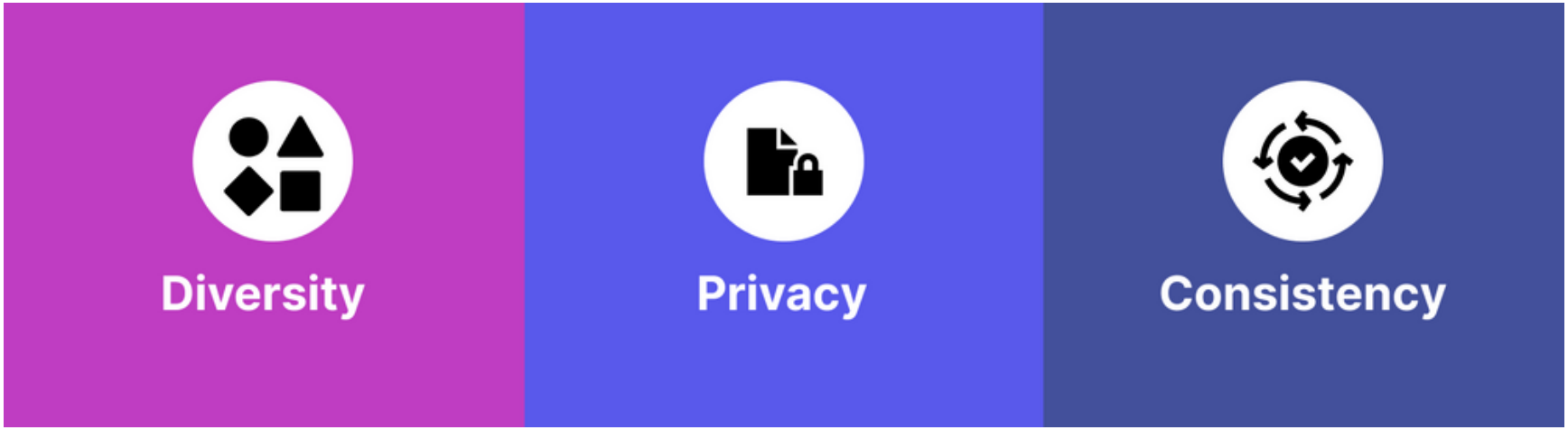
A. The system is malfunctioning and should be shut down.

B. The test data used for invalid credential scenarios contains actual valid credentials, resulting in inaccurate test outcomes.

C. The test data volume is insufficient for executing the test cases.

D. The automation script was not implemented for the login functionality.

Why Test Data Management?



Diversity: high test coverage means testing many different scenarios and having the right data for each one.

A simple registration page, for example, already requires so many datasets to cover all of the possible scenarios that can happen there :

- Valid credentials
- Empty username
- Empty password
- Incorrect username
- SQL injection attempt
- Special characters
- Too long username
- Too long password

Data Privacy: without good TDM practices, testers can risk using PII (personally identifiable information) to test, which is a breach in security.

- There are so many things you can do in TDM to prevent this from happening, such as data anonymization, which is essentially a process to replace real, sensitive data with similar but fictitious data.
- If teams decide to use real data, they can mask (i.e. encrypt) specific sensitive data fields, and use only the most necessary.
- Several teams employ Dynamic Data Masking (DDM) to dynamically mask data fields based on user roles and permissions.

Data Consistency: QA teams also need to ensure that their test data is uniform across the entire systems, adhering to the same format and standards, and even the relationships among the datasets must be continuously maintained over time when the complexity of the system grows.

Test Data Management Techniques

1. Data Masking
2. Data Subsetting
3. Synthetic Data Generation

Data Masking

- Data masking is the technique used to protect sensitive information in non-production environments by replacing, encrypting, or otherwise "masking" confidential data while retaining the original data's format and functionality.
- Data masking creates a sanitized version of the data for testing and development purposes without exposing sensitive information.

Data Masking Technique	Definition + Examples
Substitution	**Definition**: Replace actual sensitive data with fictional or anonymized values. You can leverage Generative AI for this approach; however, note that creating entirely new data is resource-intensive. **Example**: Replace actual names with randomly generated names (e.g., John Doe).
Shuffling	**Definition**: Randomly shuffle the order of data records to break associations between sensitive information and other data elements. This approach is faster and easier to achieve compared to the Substitution. **Example**: Shuffle the order of employee records, disconnecting salary information from individuals.

Encryption	**Definition**: Use encryption algorithms to transform sensitive data into unreadable ciphertext. Only authorized users with decryption keys can access the original data. This is a highly secured approach to take. **Example**: Encrypt credit card numbers, rendering them unreadable without proper decryption.
Tokenization	**Definition**: Replace sensitive data with randomly generated tokens. Tokens map to the original data, allowing reversible access by authorized users. **Example**: Replace social security numbers with unique tokens (e.g., Token123).

Character Masking	**Definition**: Mask specific characters within sensitive data, revealing only a portion of the information. **Example**: Mask all but the last four digits of a social security number (e.g., XXX-XX-1234).
Dynamic Data Masking	**Definition**: Dynamically control and limit the exposure of confidential data in real-time during query execution. In other words, sensitive data is masked at the moment of retrieval, just before being presented to the user (usually the masking logic is based on user roles). **Example**: Mask salary information in query results for users without financial access rights.

Randomization	**Definition**: Introduce randomness to the values of sensitive data for creating diverse test datasets. **Example**: Randomly adjust salary values within a specified percentage range for a group of employees.

Data Sub setting

Data sub setting is a technique to create a smaller yet representative subset of a production database for use in testing and development environments.

Data Subsetting : Benefits

- Reduce data volume, especially in organizations with large datasets. For testing purposes, smaller data volume minimizes resource requirements and therefore reduces maintenance needs.
- Preserve data integrity, as sub setting a dataset does not change the relationship between rows, columns, and any entities within it
- Easily include/exclude data based on specific criteria relevant to the team's testing needs, giving them a higher level of control. At the same time, this translates into improved efficiency in terms of data storage, transmission, and processing.

Synthetic Data Generation

- Synthetic data generation is the process of creating artificial datasets that simulate real-world data without containing any sensitive or confidential information.
- This approach is usually reserved only for when obtaining real data is challenging (i.e. financial, medical, legal data) or risky data (i.e. employee personal information).

- generating entirely new sets of data for testing purposes is a more practical approach.
- These synthetic datasets aim to simulate the original dataset as closely as possible, and that means capturing its statistical properties, patterns, and relationships.

Tools : Test Data Management

- **Informatica**- Data provisioning, data sub setting, data masking and data profiling are all included.
- **Compuware**- It intends to make the extraction, masking and delivery of test data simpler.
- **Delphix**- It can interact with many databases and systems and allows you to build and deliver masked or fake information copies for testing.
- **Microfocus Data Express-** Sensitive data is hidden and portions of production data are created.
- **IBM InfoSphere Optim-** It allows to produce, subset and conceal data for testing while preserving the security and privacy of the data.

Software Configuration Management

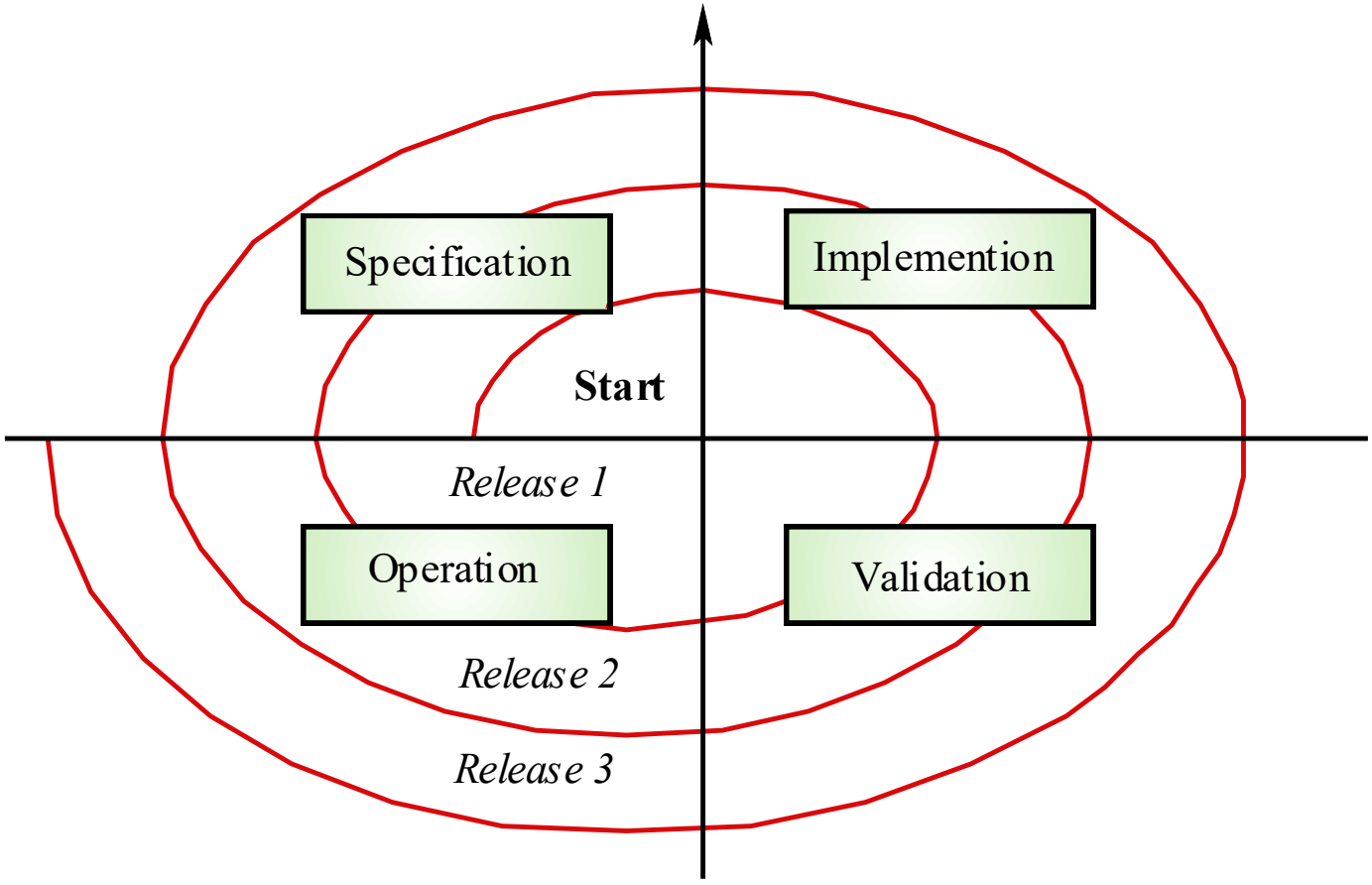
Maintenance is Inevitable

- System requirements may change during development due to environmental factors.
- Systems are tightly coupled to their environment
- When a system is installed, it changes the environment and that can change the system requirements.
- The delivered system may not meet its requirements.
- Systems must be maintained to remain useful in their environment.

Types of Maintenance

- Corrective Maintenance (21%)
  - making changes to repair defects
- Adaptive Maintenance (25%)
  - making changes to adapt software to external environment changes (hardware, business rules, OS, etc.)
- Perfective Maintenance (50%)
  - extending system beyond its original functional requirements
- Preventative Maintenance (4%)
  - Modifying work products so that they are more easily corrected, adapted, or enhanced

Spiral Maintenance Model



Maintenance Costs

- Usually greater than the development costs (2 to 10 times as much in some cases)
- Affected by both technical and non-technical factors
- Increase as software is maintained and system corruption is introduced
- Aging software can have high support costs (e.g. old languages, compilers, etc.)

Maintenance Developer Tasks

- Understand system.
- Locate information in documentation.
- Keep system documentation up to date.
- Extend existing functions.
- Add new functions.
- Find sources of errors.
- Correct system errors.
- Answer operations questions.
- Restructure design and code.
- Delete obsolete design and code.
- Manage changes.

Maintenance can be tough

- Limited understanding of hardware and software (maintainer).
- Management priorities (maintenance may be low priority).
- Technical problems.
- Testing difficulties (finding problems).
- Morale problems (maintenance is boring).
- Compromise (decision making problems).

Maintenance Cost Factors

- Staff turnover
  - No turnover usually means lower maintenance costs
- Contractual responsibility
  - Developers may have no contractual obligation to maintain the delivered system and no incentive to design for future change
- Staff skills
  - Maintenance staff are often inexperienced and have limited domain knowledge
- Program age and structure
  - As programs age, their structure degrades, making them harder to understand and modify.