**Practical no. 9**

**Abhishek Kumar Singh**

1. **Requirement Analysis**: Understand the functional and non-functional requirements of the software. This helps in defining the scope of testing.
2. **Identify Critical Features**: List down the features or functionalities of the software that are most critical to its intended purpose and business objectives.
3. **Boundary Value Analysis**: For each input field, identify the boundaries. Create test cases for:
   * Minimum value
   * Just above the minimum value
   * Maximum value
   * Just below the maximum value
   * Typical values
4. **Equivalence Partitioning**: Divide the input domain into equivalence classes and create test cases for each class. For instance, if an input field accepts values between 1 to 100, then test cases can be:
   * Any value from 1 to 100 (valid values)
   * Less than 1 (invalid values)
   * Greater than 100 (invalid values)
5. **Negative Testing**: Focus on invalid inputs. Test the software's behavior when it receives unexpected or incorrect data.
6. **Functional Testing**: Create test cases based on the functional requirements. Ensure that each requirement is tested individually and in combination with others.
7. **Integration Testing**: Test the interactions between different modules or components of the software. This includes checking data flow, interface compatibility, and system interactions.
8. **Performance Testing**: Validate the software's performance under load, stress, and normal conditions. Ensure that it meets the defined performance criteria.
9. **Security Testing**: Identify potential security vulnerabilities and create test cases to validate the software's security features. This includes testing for authentication, authorization, data encryption, etc.
10. **Usability Testing**: Evaluate the software's user interface, user experience, and overall usability. Ensure that it is intuitive, user-friendly, and meets the user's needs.
11. **Regression Testing**: After making changes or updates to the software, retest the existing functionalities to ensure that no new defects have been introduced.
12. **Error Handling and Recovery Testing**: Test the software's ability to handle errors gracefully and recover from failures. Create test cases for scenarios where unexpected errors occur.
13. **Compatibility Testing**: Validate the software's compatibility with different operating systems, devices, browsers, and other software components.
14. **Data Validations**: Test the software's data validation mechanisms. Ensure that it handles data correctly, prevents data corruption, and maintains data integrity.
15. **Localization and Internationalization Testing**: If the software is intended for a global audience, test its compatibility with different languages, cultures, and regional settings.
16. **Documentation Testing**: Validate the software's documentation, user manuals, help guides, and other instructional materials. Ensure that they are accurate, up-to-date, and provide clear guidance to users.