UNIT III – PERFORMANCE AND REGRESSION

Factors – Methodology – Tools – Challenges. Regression Testing: Types – Methods. Internationalization Testing – Ad-hoc Testing: Buddy and Pair Testing – Exploratory Testing– Iterative Testing – Agile and Extreme Testing. Usability and Accessibility Testing.

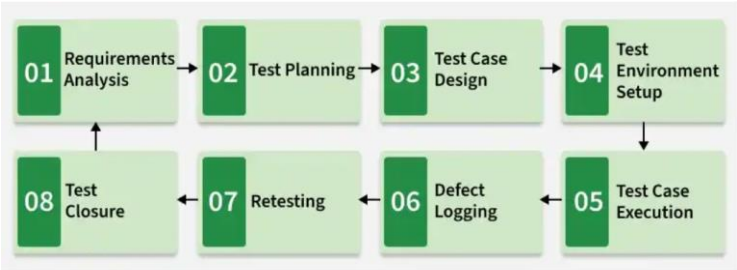
Regression Testing

Regression testing is a software testing technique used to verify that **modifications such as bug fixes, new features, or code changes do not affect the existing functionality** of the application. It ensures that previously working features continue to function correctly after changes are made.

- Executes **existing test cases** after code changes.
- Confirms that **current features still work correctly**.
- Identifies **unexpected defects** caused by recent modifications.
- Enhances the **quality, reliability, and stability** of the software.

Architecture of Regression Testing

Regression testing follows a systematic process to ensure that new changes do not affect the existing features of the software.

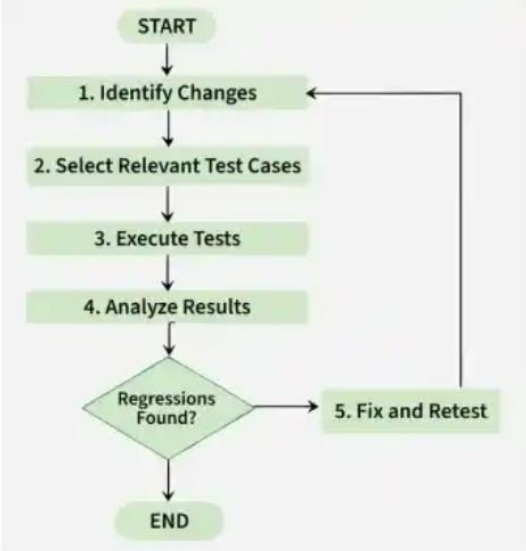


Architecture of Regression Testing

1. **Requirement Change:** Identify new features, bug fixes, or modifications in the application.
2. **Impact Analysis:** Determine which modules or functions are affected by the changes.
3. **Test Suite Repository:** Maintain a collection of existing manual and automated test cases.
4. **Test Case Selection:** Choose and prioritize the test cases that need to be executed.
5. **Test Execution:** Run the selected test cases manually or using automation tools.
6. **Defect Management:** Record, track, and resolve defects found during testing.
7. **Automation Framework (Optional):** Use automation tools to execute repetitive regression tests quickly.
8. **Reporting and Feedback:** Generate test reports and provide feedback on the stability and quality of the software after testing.

Workflow of Regression Testing

Regression testing follows a step-by-step process to ensure that recent code changes do not affect the existing functionality of the software.



Workflow of Regression Testing

1. **Identify Changes:** Review the latest code changes, bug fixes, or new features to determine the affected modules.
2. **Select Test Cases:** Choose the existing test cases related to the modified areas and add new test cases if needed.
3. **Execute Test Cases:** Run the selected test cases manually or using automation tools.
4. **Analyze Results:** Compare the actual results with the expected results and identify any defects or regressions.
5. **Fix and Retest:** Correct the detected defects and rerun the test cases to confirm that the issues are resolved without introducing new errors.

Regression Testing Tools

Regression testing is commonly automated because the same test cases are executed repeatedly after every code change. Automation reduces testing time, increases accuracy, and supports continuous integration and frequent software releases.

- **Selenium:** An open-source automation tool used to test web applications across different browsers and programming languages.
- **Ranorex Studio:** A testing tool for web, desktop, and mobile applications that supports both script-based and codeless automation.
- **testRigor:** An AI-based automation tool that enables testers to create test cases using simple natural language.
- **Sahi Pro:** An easy-to-use automation tool for cross-browser testing with support for continuous integration (CI).
- **Testlio:** A cloud-based testing platform that provides on-demand regression testing services using real testers and automation.

Need for Regression Testing

Regression testing should be carried out whenever changes are made to the software to ensure that existing features continue to work correctly.

- **After adding new features:** To verify that newly added functionality does not affect existing modules.
- **After fixing bugs:** To ensure that the bug fix has not introduced new defects.
- **After code modifications:** When the code is refactored or optimized without changing its functionality.
- **After system or environment updates:** When changes are made to the database, server, configuration, or third-party integrations.

Advantages of Regression Testing

- **Maintains software quality** by ensuring that existing features continue to work correctly after code changes.
- **Saves time and effort** by reusing existing test cases, making regression testing ideal for automation.
- **Improves code quality** by helping developers identify and fix issues introduced by recent changes before software release.

Disadvantages of Regression Testing

- **Can be time-consuming and expensive** if regression tests are performed manually instead of using automation.
- **Requires testing all affected areas** after code changes, which can increase the testing effort and execution time.

Example of Software Regression Testing

Step 1: Initial Software Development (Version 1)

A company develops an online marketplace application where users can **buy and sell used cars**.

Main feature:

- Sell used cars

Testing performed:

- Functional testing
- Regression test cases for the "Sell Used Cars" feature

Result:

- The application works correctly and is released.

Step 2: Customer Requests a New Feature

After Version 1 is released, the customer requests a new feature:

> Users should also be able to **purchase car insurance** while buying a used car.

Now the developers start working on **Version 2**.

Step 3: Developers Add the New Feature

The development team adds:

- Insurance purchase option
- Insurance payment process
- Insurance confirmation

Now the application supports:

- Selling used cars
- Selling insurance for those cars

Step 4: Test the New Feature

The testing team first checks whether the new insurance feature works correctly.

Example test cases:

- Buy insurance successfully
- Insurance premium calculation
- Payment completion
- Insurance confirmation message

These are **functional tests** for the new feature.

Step 5: Perform Regression Testing

Now testers run the **old test cases** from Version 1 again.

Why?

Because adding insurance might accidentally affect the existing car-selling feature.

Regression test cases include:

- User login
- Search used cars
- View car details
- Add car to cart
- Purchase used car
- Payment process
- Order confirmation

Step 6: Verify Existing Features

The testers check:

- ❖ Can users still sell used cars?
- ❖ Is the payment working?
- ❖ Are order confirmations generated?
- ❖ Is the search feature working?

If all these old features still work after adding insurance, the regression test **passes**.

Step 7: Execute Both Sets of Tests

Before releasing Version 2, the testing team executes:

Regression Tests (Old Features)

- Sell used cars
- Payment
- Search
- Order confirmation

New Feature Tests

- Buy insurance
- Insurance payment
- Insurance confirmation

Both must pass before release.

Step 8: Release Version 2

If:

- Old features work correctly
- New insurance feature works correctly

Then Version 2 is released to customers.

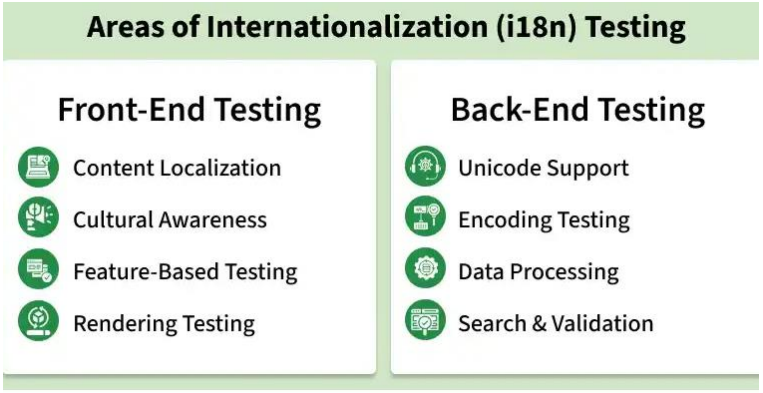
Internationalization Testing :

Internationalization Testing (i18n Testing) is a software testing process that ensures an application is capable of supporting different languages, countries, and cultural preferences without modifying its source code. It confirms that the application is designed to be easily localized for users in various regions around the world.

- ❖ Confirms that the application supports multiple languages, character encoding, and regional settings.
- ❖ Verifies that dates, times, currencies, numbers, and other locale-specific formats are displayed correctly based on the selected region.
- ❖ Ensures the application functions properly across different countries, languages, and cultural environments, making it ready for global users.

Types of Internationalization Testing

1. **Language Testing**
    - Checks if the application supports multiple languages and displays translated text correctly.
2. **Character Encoding Testing**
    - Ensures the application correctly displays special characters and symbols from different languages.
3. **Locale Testing**
    - Verifies that the application follows the regional settings of different countries, such as language, date, and measurement units.
4. **Date and Time Format Testing**
    - Checks if dates and times are displayed in the correct format for different countries.
    - Example: DD/MM/YYYY (India) and MM/DD/YYYY (USA).
5. **Currency and Number Format Testing**
    - Ensures currencies and numbers are displayed according to the country's standards.
    - Example: ₹1,000.50 and $1,000.50.
6. **User Interface (UI) Testing**
    - Checks that the application layout remains neat and readable even when translated text is longer or shorter.
7. **Input Method Testing**
    - Verifies that users can enter text using different keyboards and languages.
8. **Time Zone Testing**
    - Ensures the application displays the correct date and time for users in different time zones.



Areas of Internationalization (i18n) Testing

Front-End Internationalization Testing

Front-end internationalization testing checks the application's user interface (UI).

- **Content Localization:** Ensures labels, buttons, menus, dialog boxes, and other UI elements are displayed correctly in different languages.

- **Locale and Cultural Testing**: Checks that dates, times, currencies, phone numbers, and postal codes follow local formats.
- **Feature Testing:** Ensures region-specific features are available only to users in supported locations.
- **Language Display Testing:** Verifies that language files load correctly and translated text is displayed properly without cutting off or alignment problems.

Back-End Internationalization Testing

Back-end internationalization testing checks the server and database.

- **Unicode Testing:** Ensures the database stores and retrieves multilingual data correctly.
- **Encoding Testing:** Verifies that characters are transferred without errors or corruption.
- **Data Processing Testing:** Checks correct handling of different languages, currencies, and regional formats.
- **Search and Form Testing:** Ensures searches and form submissions work properly with international characters and multiple languages.

Internationalization Testing Process

1. **Requirement Analysis:** Identify the languages, regions, and cultural settings the application should support.
2. **Test Planning:** Plan the testing scope, test cases, tools, and testing environment.
3. **Environment Setup:** Configure different languages, locales, and regional settings for testing.
4. **Test Case Design:** Prepare test cases to verify language support, date/time, currency, number formats, and character handling.
5. **Test Execution:** Run the test cases in different locales and check whether the application works correctly.
6. **Defect Reporting:** Record and report issues related to language, formatting, encoding, or regional settings.
7. **Retesting:** Test again after fixing defects to ensure the issues are resolved.
8. **Final Verification:** Confirm that the application is ready to support users from different countries and languages.

Benefits of Internationalization Testing

- **Global Reach:** Makes the application available to users around the world.
- **Better User Experience:** Provides a consistent experience in different languages and regions.
- **Faster Global Release:** Helps launch the application in multiple countries with less cost and effort.
- **Improved Software Quality:** Makes the application more reliable, maintainable, and scalable.
- **Single Codebase:** Supports multiple languages using one common codebase.

- **Lower Maintenance Cost:** Reduces the cost of updates and long-term maintenance.
- **Standards Compliance:** Ensures the application meets international standards and regional requirements.

Tools Used for Internationalization Testing

- **Selenium:** Automates testing of multilingual web applications on different browsers.
- **Appium:** Tests multilingual mobile applications on Android and iOS.
- **BrowserStack:** Tests applications on real devices with different languages and regional settings.
- **LambdaTest:** Supports cross-browser and cross-locale testing.
- **Globalyzer:** Finds internationalization issues in source code and helps prepare applications for global use.
- **Pseudo Localization Tool:** Simulates translated text to identify UI and layout problems before actual translation.