WWW.TECHNODIVINE.COM

DESIGNING AN AUTOMATION FRAMEWORK

For Automation Geeks| By Bharat Kakkar

Return on Investment

- Dependencies – Framework should be OS Independent and Windows ROT (windows Running Object Table – Registered COM) independent, as far as possible. Because using such dependencies will increase the implementation and execution cost of the project.

E.g. Use of %COMMONPROGRAMFILES(x86)% environment variable should be avoided.

- Power of the Tool - Should use the feature of the tool effectively and efficiently
- Framework should be designed to deliver high Return of Investment by implementing
    - High re-usability and optimum utilization of available resources.
    - Low maintainability

- Avoid automating dynamic screens.
- Avoid business scenarios where complex hardware is involved

Note: Any framework should be developed if and only if one can foresee ROI on developing and executing automation.

Framework's Architecture

- Intelligent coding – It should take less efforts to automate the test e.g. end user should be enabled with deciding the test flow using data or keyword if the end user in not an automation expert.
- Dependencies - OS Independent
- Power of the Tool -Should use the feature of the tool effectively and efficiently
- Exception handling should be implemented.
- Techniques like caching, compiling all code into single library while execution should always be considered.
- Use object composition over class hierarch- avoid making the base class as giant class
- Ideally a framework should have following sub-components
    - Base Framework
        - Function Library (Script specific)
        - Shared Functions (common functions)
    - Abstract Layer
        - The Object Repository/Object definition
        - Driver Script
        - Keywords/Test Data (optional)
    - External Data
        - Input data for parameterization
        - Data for re-testing
        - Environment variables

Input Data

- For reading the input data Windows ROT should be avoided to make the execution faster, crash free and also to avoid dependencies on external programs.

E.g. Use XML data or Tools internal datasheet instead of Excel

- Test data should be segregated as per the modules or functional requirements within the datasheets.
- Input data should be considered for retesting as well as regression.

Reusability

- Shared components should be promoted.
- Duplication of work should be avoid at all levels

Scripting and Programing Level

- Separation between test setup, execution, validation and cleanup
- Coding standards should be followed to increase code readability
- Use of variables/objects should be limited for memory optimization and faster execution
- Conditional checking should be inserted to cover more scenarios and easy maintainability.
- Reporting- reporting /logging should be done at each step for easier debugging
- Calling functions and reusable actions wherever possible
- Keyboard inputs should be minimized
- Exception and error handling should be implemented wherever feasible
- Handling Application controls (objects)/ OS controls.
- Hard coding should be avoided (Parameterization should be done)
- Scripts shouldn't be interdependent, any changes made to the input data/database or the application configuration should be reset at the end of the script
- Common Scripts should share the reusable components to enable Suite creation for execution.
- Framework should be integrated with all possible layers of the application E.g. database integration
- Use of global variables should be limited, define the variables as per the scope required.
- Environment Libraries should be promoted
- Be cautious when re-factoring codes

Performance Aspects

- As far as possible do not use a hardcoded wait statement, instead use the synchronization capabilities of the tool.
- Objects in AUT should be well defined for easier recognition
- Object's Hierarchy should be followed, especially for web applications
- Loading and unloading a library/Action takes a lot of time and memory, hence should be one time or limited.
- Be cautious while using Regular Expressions
- Use blocks if available in the language you are working with.

Execution Aspects

- Capabilities to Run on
  - Multiple platforms (if feasible/if required)
  - Multiple OS should be supported
- Parallel execution should be considered
- Test Timeouts should be considered and handled
- Strategy to execute failed tests should be planned
- Priority based execution and user supplied variables should be enabled to handle e.g. may be done using environment variables.

Result

- For creating repots Windows ROT should be avoided to make the execution faster, crash free and also to avoid dependencies on external programs.

E.g. Use XML data or Tools internal datasheet instead of Excel

- Reporting should be done by logging each execution step to enable easy debug path.
- If feasible snapshots of the failures should be kept.
- Wherever possible Summarized report should be there on test and suite level.

Maintainability

- Modification to the application should be easily handled.
- Use of shared framework components should be promoted.
- Use relative paths wherever possible.