GUIDELINES FOR BUILDING A ROBUST TEST AUTOMATION FRAMEWORK

Contents

[1. Introduction 2](#_Toc365008720)

[2. Test Automation Framework Design Challenges 3](#_Toc365008721)

[3. PRE DESIGN Considerations 3](#_Toc365008722)

[4. FraMEWORK design Principles and GUIDElines 5](#_Toc365008723)

[5. Framework Robustness Factors 6](#_Toc365008724)

[6. Object Oriented Implementation 7](#_Toc365008725)

[7. Design Patterns 7](#_Toc365008726)

[8. Sample Modular Automation Framework ARCHITECTURE 7](#_Toc365008727)

[9. Standard coding Guidelines 8](#_Toc365008728)

[10. Framework Maintenance 9](#_Toc365008729)

[11. Version Controlling 9](#_Toc365008730)

[12. Continuous Integration 10](#_Toc365008731)

[11. Summary 11](#_Toc365008732)

1. Introduction

Test Automation plays a significant role in Software Testing Life Cycle. The need for Automation is envisioned to achieve the following

* Increased Test Coverage
* Faster Test Execution
* Increased Regression Cycles
* Unattended test execution.
* Cost Reduction.

The success of automation depends on how Robust is the Automation Framework. To accomplish the above, requires designing an effective Test Automation framework. Designing a Robust Automation framework is based upon numerous dependency factors like

* Nature of the application
* Testing type (Functional or Performance)
* Test Data Sources (dynamic data)
* Technology/platform (used for Framework Development).
* Target Users (Users who are going to use this automation framework).
* Automation architecture
* Automation Environment
* Target coverage (GUI level, API level...)
* Application dependencies (Third party Integrations)

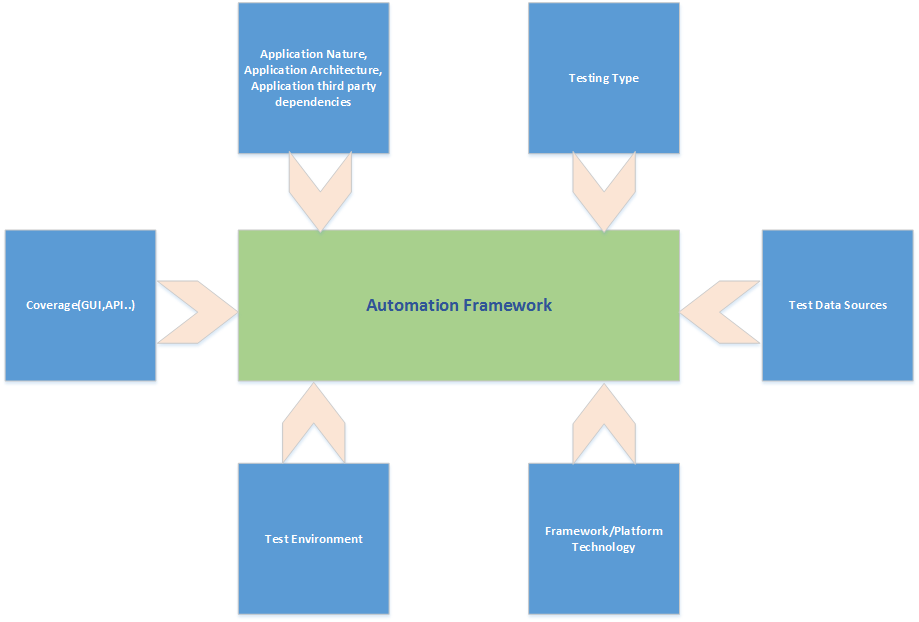


Figure: Design Dependencies

1. Test Automation Framework Design Challenges

Tool identification

Tool identification is one of the initial challenges in the framework design. Selection of appropriate tool would merely reduce the effort of enhancing the automation framework for supporting additional functionalities.

Scope

Scoping the Test Automation is another challenge. Framework design depends upon the areas that need to be covered as a part of automation.

Technology Selection

Technology selection is another challenge in implementing framework design in terms of maintainability, scalability and portability.

Areas of Automation

Simulating real time business scenarios which are processed by third party software is one of the Framework challenges

1. PRE DESIGN Considerations

Below are some of the initial considerations which will provide inputs in designing an effective Automation Framework and also addresses the above challenges

Nature of the application

The nature (Web, Native, API, Mobile...) of the application gives you a clear Scope of what all capabilities need to be built in a framework.

Testing type

The types of testing required (functional/non-functional) provides a choice to framework designer in selection of available automation tools or APIs that needs be integrated into framework

Test Data source

The incoming and outgoing data of the application/system helps in designing framework scalability in handling the test data volume. (Ex: if user needs to automate simulating 1000 incoming transactions into the system)

Technology/Platform

Selection of Technology/Platform do really matters in the design of Automation framework in terms of framework interaction with the Target application API’s or web services, framework performance and scalability

Target Users

Prior designing a framework, one has to consider the target users of this framework. If the users are business analysts, the framework should be designed in such a way, the framework provides an easy interface to run configure and maintain the automation tests.

Application Architecture

Understanding the application architecture is one of the important factors in designing the automation framework. This will help the framework designer in developing the framework components which might interact with Application at API level (ex: Before starting the test run if the application under test (AUT) requires preconfigured scripts /data that need to be initialized)

Test Environment

Gather the test environment details. If the application (AUT) supports multiple platforms (Windows, Linux…), one should think about the portability of framework to be able to run and execute the tests on different platforms which would be another input factor for the framework design

Target Coverage

Clearly identify the target coverage of the application. Should the framework target the application API’s or GUI interface? This would scope your automation framework design effort. However designing a framework supporting other testing types (GUI, Performance…) in the future, depends on how modular your framework is built.

1. FraMEWORK design Principles and GUIDElines

Below are some of the design principles that need to be considered as a part of your Framework design.

Application Independent

Framework should be independent of the Application under test. Any Data dependencies and configurations related to application should be decoupled from the framework

Automation Tool Independent

Framework should be independent of the Automation tools and should be capable of integrating multiple Automation tools/api.

Framework Third Party Dependencies

Care should be taken such that the identified automation tools and other third party tools integrated into framework have proper support and future updates.

Tool Impacts

Understand how the Automation tool interacts with the target application and identify any performance or security impact on the application (AUT). Ex: considering a tool injecting JavaScript into browser to simulate user actions might impact the performance of the application which is developed in JavaScript.

Test Harness Isolation

Framework harness should be isolated from the test scripts, test data and other configuration parameters. This would benefit execution of automation scripts on different versions/releases of the Application

Complexity Encapsulation

The complexity of the framework should be encapsulated from the target users. Encapsulating the framework implementation will increase the usability of the framework, reduces error prone and also reduces the automation script design time

Reusable components

Code that has been commonly used across the framework should be designed as reusable components rather than duplicating the same code. As your reusable components grow, it becomes easier to lay the framework for complex Automation projects.

Loosely Coupled Components

Loose Coupling allows you to make changes in one area of the framework without impacting other components. Loose coupling makes the framework more flexible, more adept at change, and easier to maintain.

Advantages

* Scalability
* Maintainability

1. Framework Robustness Factors

A good and robust framework should consider all possibilities and situations like unexpected error handling, application crashes and graceful exits.

Some benefits of the Robust Code

* Robust code is easier to change.
* Fewer Bugs.
* Easy and Better Integration. From an integration point view, is always going to have the same behavior
* Consistency.
* Better Maintainability

One should consider handling the following factors that would have an impact Framework robustness.

* Rigidity : The tendency for the Framework to be difficult to modify or enhance even in simple ways
* Fragility: Tendency of the framework to break in many places every time it is changed.
* Immobility: The inability to reuse framework for other projects

1. Object Oriented Implementation

Implementing the design using Object Oriented Programming gives a great benefit which in results in better design and fewer flaws

Advantages:

Improved Software Maintainability:

Since the Object Oriented design is Modular, any changes required in the framework can be easily updated without a need to make major changes.

Faster Development

Reuse enables faster development. Object-oriented programming languages come with rich libraries of objects, and code developed can be reused in future projects

Lower cost of development:

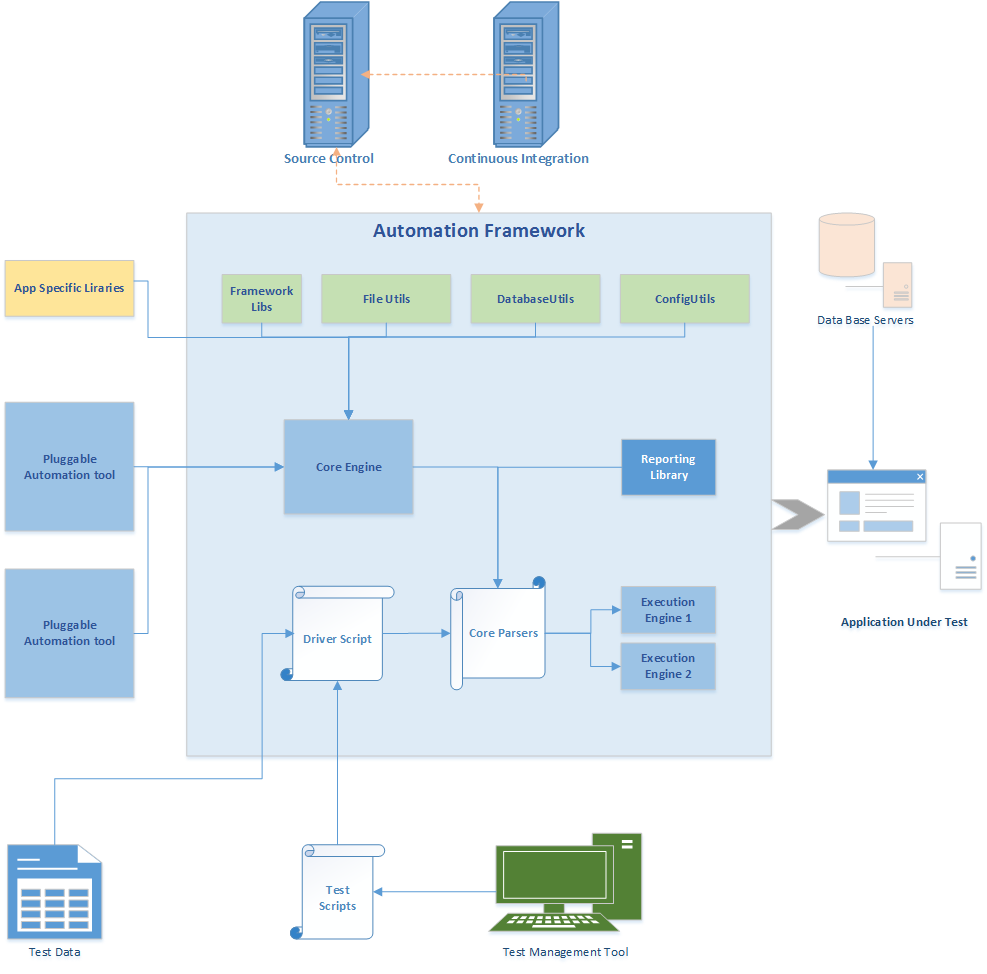
The reuse of software also lowers the cost of framework development. Typically, more effort is put into the object-oriented analysis and design, which lowers the overall cost of Framework development.

1. Design Patterns

Implementing the Object Oriented concepts with design pattern usage allows you to write faster code, encourage code reuse, increase readability and your code becomes easily adaptable to the changing requirements.

Follow the standard Design Pattern principles (ex: Singleton pattern, Factory pattern….) to achieve high maintainability and extendibility

1. Sample Modular Automation Framework ARCHITECTURE



1. Standard coding Guidelines

In addition to the above, following the standard coding guide lines would add some percentage of Robustness to your code and test scripts.

Since the white paper focuses more on Framework design, I’m not including all the standard coding guide lines.

However, some of the most commonly recommended guidelines are below

* Do not Swallowing exceptions.
* Encapsulate the code complexity.
* Maintain proper encapsulation of properties and methods.
* Usage of Interfaces for multiple implementations for same business functions
* Avoid repeating validation in test scripts
* Maintain reusable functions/methods
* Common Setup and Teardown for multiple tests
* Avoid using Absolute file paths
* Create independent test scripts/scenarios
* Avoid connection and object pooling my creating multiple database connections with in your test scripts.

Usage of static code analysis tools detects the code errors in early stages, warns on any unwanted code and if the code violates a defined set of coding standards. This would be a great benefit and eliminates manual code review to some extent.

1. Framework Maintenance

As the Automation framework becomes more structured, scripting increases the complexity in maintaining the framework and could have an impact on Framework performance.

Following factors needs to be considered from Test Script maintenance perspective

* Test Script dependencies:

Test Data should not be directly included with in test scripts and should be isolated. Having the test data isolated allows you to run different tests with different sets of data against different application environments (QA, Dev., UAT...) with minimal change in test scripts.

* Redundant Test Data:

Test Data used should not be duplicated across test scripts. Adding more data, results in complexity of maintaining the test data.

1. Version Controlling

Version Controlling allows you to take the Framework code backup for any accidental code deletions allows multiple sharing of the same code and ensures everyone is in sync with the latest code changes.

Maintaining different versions of code for different Framework release is always a best practice.

1. Continuous Integration

Integrating your Automation Framework with continuous integration process (CI) would add a value on your Framework maintenance

Some of the advantages of using CI are

* Automates the framework build deployment process.
* Since the framework build runs continuously, bug detection is done in early stages ensuring the quality of the build.
* Ensures code stability between different modules integrated
* Unattended Regression test execution.
* Distribution of test scripts on multiple machines/environments
* Automatic build result notifications

Object Repository Design

Object Repository should be configurable and should be decoupled from the test harness. The interface for storing the objects should be simple and easy to use. Typically maintaining the Object Repository in XML files is recommended. Usage of logical name for objects is recommended rather than directly referring to complex object names.

Reporting & Logging mechanism

Framework should have the capability of generating reports summarizing the Pass/Fail percentage. Reporting should provide flexibility to customize based on the need. Also, it’s a good practice to include screenshots and logs associated with the test scripts with in the report.

Logging, in Automation framework plays a critical role. Logs provide the piece of evidence on what went wrong in code and what actions are being performed by framework or tool. Not all log information might be relevant for all. A business analyst who uses framework might not be interested in knowing the object state and its value rather he/she would be interested in knowing the action that has been performed.

A good framework should provide logs at different levels

* Debug level logs
  + Used for debugging the code. Used by developers
* Info level logs
  + Used by automation testers, business analysts
* Trace level logs
  + For Audit level information
* Error level logs
  + Whenever an error occurs during execution

11. Summary

This white paper addresses the Automation framework design challenges, provides guidelines on Design considerations, Design principles and best practices which would help in building a Robust Automation Framework.

[[1]](#footnote-1)

About Alliance Global Services  
Alliance Global Services is a software development firm that partners with software, technology   
and information-intensive businesses on their mission critical work. Alliance architects and builds software applications, platforms and products that become primary drivers of innovation and revenue growth for its clients and their businesses. Alliance is recognized for driving quality and speed-to-market when business success depends on The Software InsideTM. Founded in 1994, Alliance is headquartered in suburban Philadelphia, Conshohocken, PA. For more information, visit [*www.allianceglobalservices.com*](http://www.allianceglobalservices.com/index.html)*.*

Corporate Headquarters  
Six Tower Bridge | 181 Washington Street, Suite 350 | Conshohocken, PA 19428  
Phone: 610.234.4301 | info@allianceglobalservices.com | www.allianceglobalservices.com

[[2]](#footnote-2)

1. <http://www.makinggoodsoftware.com/2010/03/02/robustness-the-forgotten-code-quality/> [↑](#footnote-ref-1)
2. http://www.objectmentor.com/resources/articles/Principles\_and\_Patterns.pdf‎

   http://www.saylor.org/site/wp-content/uploads/2013/02/CS101-2.1.2-AdvantagesDisadvantagesOfOOP-FINAL.pdf [↑](#footnote-ref-2)