SQA For Executor Module

From TBTAF Project

A00354517 - Oscar Padilla López
A00749401 - Juan Francisco Calvillo Villegas
A01201034 - Eduardo Cruz Hernández
A00354552 - Rafael Ramírez Barba

Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Guadalajara
Pruebas de Software y Aseguramiento de la Calidad
Noviembre de 2015

1. Purpose

The purpose of this document is to define the Software Quality Assurance Plan (SQAP) for the Executor module of TBTAF project, and define the challenges, assumptions, tactics for addressing QA, and the integration tactics to be used in order to make the module work correctly with the other modules created independently.

1.1 Scope

This plan establishes the SQA activities performed throughout the life cycle of the TBTAF Executor module. The system level testing is out of scope of this document, which would include the whole TBTAF software quality activities. Instead, the focus of this document is in the Executor module and the integration tactics will be described to the extent that they are related to this module.

1.2 System Overview

TBTAF is the acronym of Tag Based Test Automation Framework. Its purpose is that the test specification and code stop being two separate artifacts. The code is filled with certain tags that allow the framework to perform a smart execution of the automatic tests, avoiding that way the use of the brute force approach.

It is broken down into several modules, like an interpreter, which translate certain scripts into actual commands to either enable a TBTAF service, execute tests, get information about an execution job, etc. Module discoverer is in charge of identifying the tags added to the code and inform the orchestrator which is the module that can be considered as the brain of the framework. It coordinates the execution and communication between the other modules.

This document is focused on the Executor. The TBTAF Executor module receives the set of tests to be executed and then collects the corresponding results.

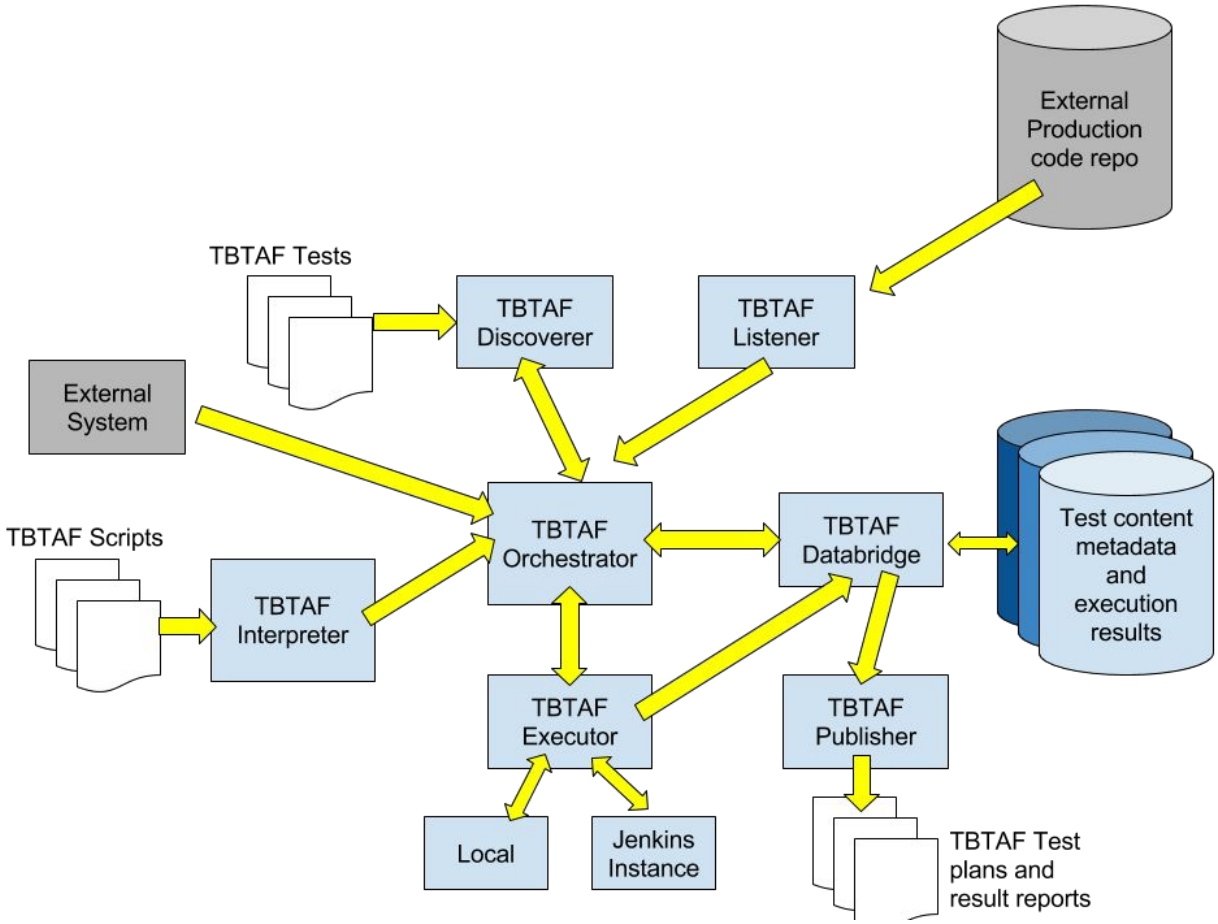

Figure 1.1. TBTAF architecture diagram

1.3 Reference Documents

TBTAF wiki at: https://github.com/S41nz/TBTAF/wiki

2. Management

This section describes each major element of the organization that influences the quality of the software.

2.1 Organization

There are 3 groups affecting the QA organization:

1. **Program manager**.
    a. Reviews and approves the Executor module SQA Plan.
    b. Identifies if an audit performed by an external quality team is required.
    c. Identifies the quality factors to be implemented in the software.
    d. Specifies the technology to be used for the development of the tool.
    e. Reinforces any quality standards to be followed.

2. **Project manager**.
    a. Detects and follows-up on any quality issues raised by SQA.
    b. In charge of the validation of the developed product. Notifies if the software developed does not meet the requirements provided.
    c. Supports the development team in the creation of QA related tasks.
    d. Creates part of the code for integration testing.
    e. Creates a first version of the software specification.

3. **Development team**.
    a. In charge of the verification of the developed product.
    b. Creates the code for unit testing.
    c. Creates part of the code for integration testing.
    d. Fixes any quality issue detected in unit or integration testing.
    e. In charge of keeping the specification up to date.

3. SQA Tasks

3.1 Task: Review Software Products

The software product to be evaluated in the Executor module is the main API class: TBTAFExecutor which includes the use of an internal class called ExecutionTBTestSuite and the public results class TBTAFExecutionStatus.

Guidelines for testing:

1. **Unit test**. Each class must be unit tested. Unit testing frameworks will not be used because of the amount of time available is not enough for the development team to learn them, ad-hoc scripts will be used instead. The scripts must cover each relevant method of each class with valid and invalid inputs (if the method receives any input).
2. **Code review**. Each class must be reviewed for at least two developers before being committed to the GitHub repository.
3. **Integration testing**. These tests will be conducted in conjunction with the other module development teams using the tests provided by the project manager.

3.2 Task: Evaluate System Requirements Analysis Process

The Executor module requirements were described by project management specifying the API expected to be created in order to be used by other modules. The requirements were analyzed in the process of the detailed software design and changes were proposed in short meetings with project management.

The development team was assigned to update the public documentation of the API in order to match to the modifications accepted.

3.3 Task: Evaluate System Design Process

The Executor module high-level design was described by project management specifying the API expected in order to be used by other modules.

The development team was in charge of the module detailed design. The system design was discussed with the project manager and completed based on his feedback. Further evaluation can be scheduled based on the results of the integration phase with other modules.

3.4 Task: Evaluare Software Requirements Analysis Process

We had a session in which Project Management told us the requirements of our module. They also told us how the module would fit in with the rest of the project.
After a first iteration in the Development cycle, we received feedback from PM and worked on a few Action items derived from that feedback.

3.5 Task: Evaluate Software Design Process

General class hierarchy was explained by Project Management. After feedback gotten from a review with PM, one of the Executor module's methods was revisited and modified to return a new type of object. This change was documented in the wiki.

3.6 Task: Evaluate Software Implementation and Unit Testing Process

We created a testing file that would test every bit of functionality in the Executor module as if the execution was held by the Orchestrator module. All functions proved to be correct and, features like parallelism executed correctly.

3.7 Task: Evaluate Unit Integration And Testing, Ci Qualification Testing, Ci/hwci Integration And Testing, And System Qualification Testing Processes

The wiki is constantly updated so that everyone's got visibility on the overall design and progress of the project. A session is scheduled to integrate all modules and test them as a whole. The test environment will be a standard laptop. If discrepancies arise during the integration testing, the involved modules will be updated and their corresponding entries in the wiki updated accordingly.

3.8 Task: Non-Deliverable Software Notification

The system does not depend on any Non-Deliverable software. It is fully developed in Python, which can be obtained from the official website. Installing the Python tools in the target machine should suffice to be able to execute and use the System.

3.9 Task: Evaluate Storage and Handling Process

All the code is stored in an online Repository, github.com. All documentation is stored on Wiki pages on the same repository. Code is "uploaded" using a git console to connect to the repository, pull the existing code, and push the new code back into the Repository.
There is no physical documentation that needs special storage, so there is no risk on this point.

3.10 Task: Evaluate Deviations and Waivers

Deviations and waivers may occur to the Executor module in case that additional functionality is required during the Execution phase. The Wiki should always be updated with information about deviations and waivers that may occur.

3.11 Task: Evaluate Configuration Management Process

Information about the configuration of the Executor can be found in the Wiki. Should any configuration change be required, it will be updated there and then modified in the code. The naming convention that is being followed by the Executor module, including classes, methods, and parameters is the same as the specifications in the Wiki.
There are no configuration files except those required on each folder to specify that the folder is a package (__init__.py).
The version control of the source code is managed by the github and git tools. The version control of the Wiki is managed by github alone. It is recommended that only the owners of the Executor module perform changes to the related source code and Wiki files.
The Executor module can be found on its own directory with the same name (executor) on the tbtaf root directory. Common files used by the Executor module can be found under the common directory, also on the tbtaf root directory. Enums used by the Executor can be found under the enums directory, which is inside of the commons directory.

3.12 Task: Evaluate Software Development Library Control Process

All libraries required for the Executor module are tested locally and updated via git, which handles the versioning. ***If an older version is needed to be retrieved, it can be done via git commands*** There are no physical files in the project.

3.13 Task: Verify Project Reviews and Audits

3.13.1 Task: Verify Technical Reviews

1. System Requirements Review (SRR) – Audit team will check:
    (1) If the requirements are written in a list that clearly separates one of each other.
    (2) They will review that each requirement is clearly stated and has no ambiguity.
    (3) Each requirement comes from a user story or business need.
    (4) Further reviews on the design phase to validate that the artifacts generated on that part of the project actually resolves the requirements.

2. Software Preliminary Design Review (PDR) – Audit team will check if:
    (1) The way the Executor will interact with the Orchestrator is correct.
    (2) The defined platform is the most appropriate one to run that module.
    (3) The configuration management of the module is correct for the entire project.
3. Software Critical Design Review (CDR) – Audit team will asses:
    (1) The classes design and interaction defined to solve the requirement of the Executor actually cover the needs of that module.
    (2) The algorithms that the Executor will use solve specific problems mentioned in the system requirements.
4. Software Test Readiness Review (TRR) – Audit team will validate:
    (1) That the QA tactics specified on a previous section are completed, totally documented and ready to be put in practice.
5. Formal Qualification Review (FQR) – SQA team will:
    (1) Run a determined number of tests, based on the tactics mentioned in the previous item, using different types of inputs and check the module's behavior to validate if it acts in concordance to the system requirements.
6. Production Readiness Review (PRR) – Audit team will:
    (1) Ensure that the tests ran were complete enough in terms of code and system requirements to ensure that all of them were covered.
    (2) The interaction of the Executor with the other modules is correct.
    (3) That the defects found in the testing phase have not a big impact so the system can be installed in production environments.

3.13.2 Task: Verify Management Reviews

SQA will provide the following information to management:
a. Compliance. Not applicable.
b. Problem areas.
- SQA will list in a document those areas that require special attention after each technical review.
c. Risks.
- After each technical review and based on the identification of the problem areas, SQA team will list a series of risks that may impact the life cycle of the project.

3.13.3 Task: Conduct Process Audits

-Validate that the system requirements are clear enough and the SW development team is clear about them.
-Unit and integrated testing of the module Executor was held.

3.13.4 Task: Conduct Configuration Audits

3.13.4.1 Functional Configuration Audit. SQA will:
-Do a mapping between the requirements, design documents and the code related to that design to analyze if they correspond each other. If there is code extra or features not covered by any document design or code programs, the audit is not successful.

3.13.4.2 Physical Configuration Audit. SQA, in case the previous task is successful, will
-Use the same mapping created on the previous section, adding the user documentation part to it, to ensure that all the documents correspond to the requirements stated.

3.14 Task: Verify Software Quality Assurance

-Auditor will check if requirements are stated as specified in section 3.22.1.a.
-Validate that the unit and integration tests have passed.

3.15 Responsibilities

The ultimate responsibility for the quality of the TBTAF Executor software and associated documentation produced by Executor Team rests with the TBTAF Executor Software Project Manager. The SQA Manager shall implement the SQA procedures defined in this plan.
SQA derives its authority from the Project Manager through the Executor Team Manager. SQA shall monitor project staff activities and review products for compliance to applicable standards, procedures, and reference (e). The results of SQA monitoring and analysis along with SQA recommendations for corrective action shall be reported to the Software Project Manager, and, as required, to the Executor Team Manager. All documents and software approved by the Software Project Manager for release to Test Cases Executor shall have been reviewed and approved by SQA.

3.16 Schedule

For this project, the revisions will be made:
-After the requirements documentation has been done.
-After the design and coding phase has been done.
-After the unit testing has been completed.
-After the integration testing has been completed.

4. Documentation

The public documentation is available at GitHub repository at: https://github.com/S41nz/TBTAF/wiki.

Because of the time constraints detailed design documentation has not been developed. This and any other documentation will be produced by direct request of the program manager.

5. Standards, Practices, Conventions And Metrics.

The standards and conventions followed in the code are those for Python programming, or *pythonic way* as it is known among the Python developers. The standards being followed are those for Version 2.7 of Python. This includes:

- Classes starting with capital letters.
- Passing self as parameter on the methods.
- Using libraries and packages by including the __init__.py file
- Creating getters and setters for Class attributes
- Exception handling

5.1 Metrics

The following measures will be made for the Executor module:

- Number of Executor features planned for the full project
- Number of developed Executor features as per the available time.

6. Test

The following Unit Test classes were created:

- **utest_executor.py** for TBTAFExecutor class
- **utest_executionTBTestSuite.py** for ExecutionTBTestSuite class
- **utest_tbtafExecutionStatus.py** for TBTAFExecutionStatus class

The tests for Integration Testing will be defined by Program Management and Project Management.

7. SQA Problem Reporting And Resolution

7.1 Process Audit Report
7.1.1 Submittal and Disposition of Process Audit Report
7.1.2 Escalation Procedure for Resolution of Non-Concurrence on Process Audit Report
7.2 Recording Problems in Software Code or Documentation.
7.3 Software Tool Evaluation Report
7.4 Facilities Evaluation Report

8. Tools, Techniques, And Methodologies

Tools

Python interpreter. It will be in charge of the module execution.
Notepad++. This is a common text editor tool. It will be useful on program outputs visualization and comparison. Also to check and modify the source code.
Git Hub. This is the repository being used to store all the TBTAF modules.
Git Client. This is to connect to Git Hub to allow the Executor developers to upload the source code to the master repository.

Techniques

Code reviews. This will be used to find any error related to the good practices of the language Python and identify any possible logic error.
Unit testing. This will test all the methods of the classes related to the Executor, to find any possible bug on their code.
Integrated testing. Its scope is the Executor only. It will be used to run the executor using some dummy test cases to analyze its behavior and check how all the classes that compose the module interact among them.

9. Code Control.

Executor Module uses git as Version Control Software, and github.com as Repository of the Code. The whole TBTAF Project uses the same Version Control Software and Repository.

Only the team members assigned to the Executor Module of the TBTAF Project should push code into the Repository.

Code changes to common files need to be reviewed by the Project Manager to make sure that the change does not affect any other Modules of the Project.

User access to the code is regulated by:

- github.com authentication process after proper registration.
- Users are able to access the TBTAF Repository only after the administraor has granted access to it.

Appendix A

Bugs found during Unit Testing

- There was a typo in the abort method in ExecutionTBTestSuite
    - The TBTestSuite class was referred to as TbTestSuite
    - There was the same issue in other methods like resume.
- There were missing *self* statements in abortTestCases
- Changes in TBTestSuite
    - Variable testCases changed to be an instance variable instead of static
    - EndTime variable is not compared against if the Test case hasn't been executed
- Typo in pauseExecution
- Typo in resumeExecution
- Error in class ExecutionTBTestSuite
    - If the object of type TBTestSuite asociated has no Test cases, getRunStatus crashed since it'd try to divide by zero.
- Change in class ExecutionTBTestSuite
    - Statuses pausing and aborting were added becuase processes Pausing and Aborting were labeled as Executing
    - Attributes that should be optional in the Constructor are now actually optional
    - The execute method now only runs successfully if the suite state is either non-started or aborted.
    - Resume can only run successfully if the current status is paused.
    - An Exception is thrown if the getRunStatus and the suite as no cases.
    - Unused code was removed.
- Changes in the Executor class
    - Calls to time.time() were changed to datetime.datetime.now() so it would be compatible with other modules.
- Changes in the TBTestSuiteResult class
    - Start and End dates shall not be set since both are calculated on the fly using the suite's cases.