Automation testing project requirements

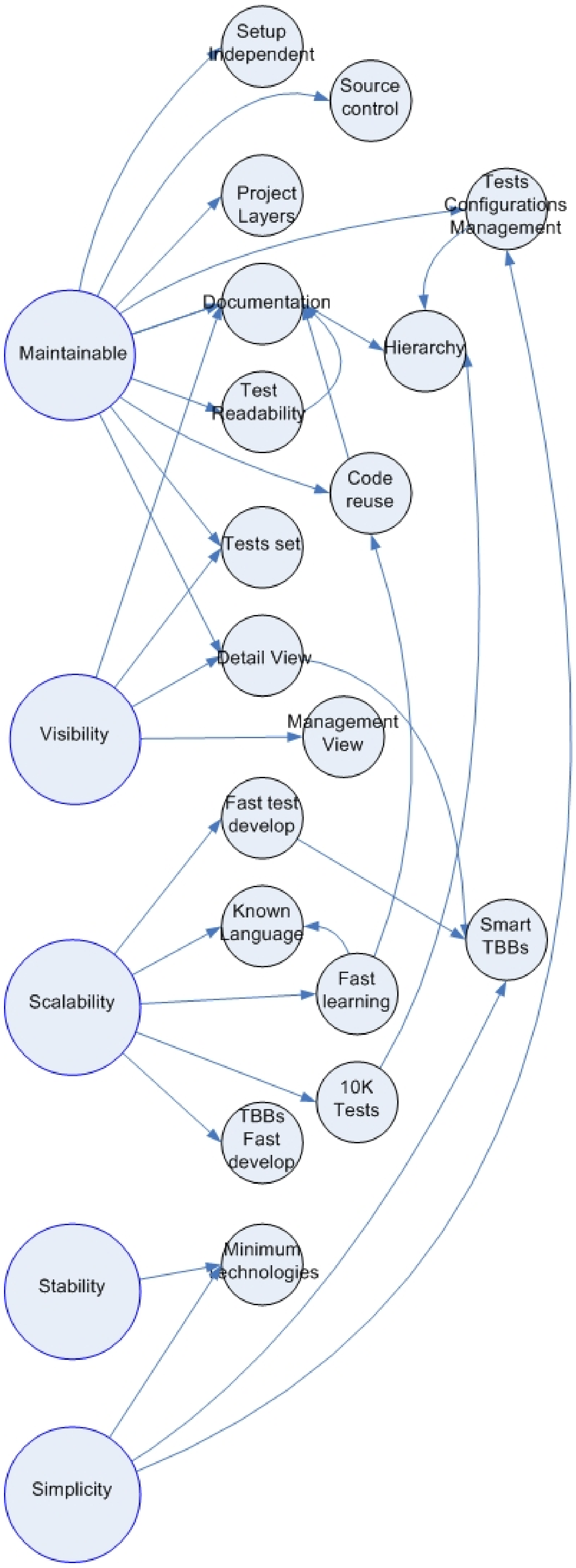
The writer is an independent consultant for automation testing in the fields of networking and J2EE application. Guy_Arieli@hotmail.com. Tel: +972-54-7899446.

In the process of building new automation testing project, understanding the requirements of the project is very important. In this article I would like to define and explain a set of requirements and design considerations, I find common to most of the automation projects.
There are new ideas like "Tests Configurations management" and "SUT independent" that you might find helpful in your project.
But in general this article target to be a baseline for your project requirements document.
I target this document to functional system testing but it's applicable to unit and integration testing as well.
You can take it and add your project specific requirements.
I hope it will help.

The main requirements (I call them "level 1 requirements") order by there importance are:

- Maintainability
- Visibility
- Scalability
- Stability
- Simplicity

From every "level 1 requirement" you can associate sub requirements. When I draw it, I got the following map (don't panic, it's little frightening at the beginning).



The direction of the connections implies on the direction of the request. For example 'Simplicity' requires 'minimal technologies', the line will be from 'Simplicity' to 'minimal technologies'.
You can change it to suit your project and your understandings. You can add requirements remove some and change add or remove the connections between them.

Following are explanations for the map bullets:

Maintainability

The maintainability is the most critical factor in automation projects. Most of the automation project (especially project involving GUI) fails because of maintenance problems. When the project grows there will be more and more tests re-factoring and test building blocks (TBB) re-factoring.

Tests Configurations Management (TCM)

The usual case for most of the automation project I run into is that every test has its own configuration. In the better cases the test is separated to setup phase, the test main phase and the teardown phase. I can see a few problems with it:

- It creates restrictions for tests execution order.
- When one test fails it can affect the other tests.
- You have more code and more code repetitions.
- You get longer tests execution time.

Test Configurations Management is framework capability that enables you to define a set of configurations. Every test is bind to a configuration. It's the framework responsibility to move from one configuration to the other.
The test can assume it's running under the bind configuration. The test is not allowed to change the configuration.
Using TCM sometimes requires to revisions in your tests plans, but it easiest your project maintenance and can shorten dramatically the tests development time.
The models and implementation of Tests Configurations Management is a subject for deferent article.

SUT Independent

SUT (System Under Test) Independent is a framework module that enables you to run tests on deferent physical SUTs. Assume you are testing a system that contains a few devices. You identify the devices by there IP address. Now you have a few identical setups, that the only deferent between them is the devices IPs. The requirement is that you will be able to run the tests on deferent setups without making changes to the tests themselves.
(You can't hard code the IPs to the tests code.)

Project layers

Your automation project should be build of 3 main modules the framework the TBBs and the tests.
We can say that this should be part of the project design and not part of the requirements. But I add it because it's so fundamental.
Keep those modules separated will help you maintain your project.
I will give a short explanation about each module.

- Framework – responsible for tests execution, test repository and grouping (suite), test flow handling, visibility, tests configurations management and SUT independent.
- TBB – the tests building blocks are the interfaces to your business components. If you test Microsoft Word (some thing I hope I will never do), one of the TBB method will create new document, one will type text to the document, one will save a document and so on. Using the TBB methods you will be able to build your tests.
- Tests – this one is obvious.

Documentation

All the project modules the Framework, the APIs and the Tests should be well documented. I find it very helpful to create documentation convention that will be used in the project (especially for the tests documentation).

Tests Hierarchy

Tests hierarchy issue is something between a requirement and a design recommendation.
The tests should be organized in a tree. There should be a single testing tree to your project.
The hierarchy of a test will set its place in the tree and in the used file system (like package in java).
The number of hierarchies should not be limited.
It will help when you hopefully will have thousands of tests. It also can be used in the Tests Configurations Management.

Test readability

The tests in your project should be readable, and not only to the one how wrote them. When opening a test it should be clear what is tested and how it's done.

Code reuse

As in every software project you should target that the code or tests parts in this case, will be reused. In most of the cases it's a matter of conventions.

Tests set (suite)

Test set is a logical grouping of tests. It's should be part of your automation framework capabilities.
Following are the feature requirements:

- Creation of unlimited variety of tests sets.
- Enable selection of tests.
- Control the order of the tests
- Control the tests running flow.

There should be a mechanism to release tests sets.

Visibility
The project should support the needed visibility. Usually (in functional system testing) I see a need for 2 layers of visibility:

Detailed view
A report that log every call to every API method in the test. In case of test failure it should give enough information to understand what failed.

Management view
Give the needed information to the management team. Usually the coverage and the status of the run.

It's important that when writing a test you will not be troubled by supporting the needed views. It should be part of the test building blocks (TBB) responsibilities.

Scalability
Scale is closely linked to maintenance. If you will not design your project to overcome scale problems you will end up with a lot of maintenance.

10K tests
Soon enough (hopefully), you will have a project with more then a thousand tests. It will grow to few thousands in a year. The tests will be written by growing number of QA and R&D engineers. It is something to keep in mind when designing and coding the project.

Known language
This one is a personal request; please don't invent a new scripting language. It is still a mystery to me, why so many automation projects and up in inventing new exciting scripting language. So please use existing language. Language that support object oriented is even better.

TBB fast development
One of the biggest pitfalls in automation projects is the development of interfaces to the system tested.
Following is a list of requirements of the TBBs:

- Using committed APIs – it should be built on committed APIs of the product you are testing. The commitment is needed to insure stability and proper use of the APIs.
- Easy to interface – it should be easy to interface to the needed API. The interface should be product version independent.
- Stable – the APIs that are uses should be in a stable stage.
- Simplicity – the interfacing should be simple.
- Early availability – in the R&D process, were new feature and other changes are entering the product, the used product API should be available in early as possible stage of the development.

Usually the first API to be chosen is the GUI. The problem with GUI is that it fails all the listed requirements. It's not committed, it's not easy to interface, it's usually not stable not simple and it's the last to be developed.
In most of the cases my recommendation is not to use the GUI for functional testing, but to design the product in such a way it will have a business logic layer, that is used by the GUI, and by the automation.
The TBBs will be build over this business logic layer. Then the GUI or the presentation layer can be tested manually (in a very quick and easy process).

Stability/Simplicity
Your automation project should earn the trust of the colleagues, QA engineers and of course the software engineers that develop the product you are testing. Stable automation will help you in this task.

Smart TBBs
Usually the tests in your automation project will be written by QA engineers. They should be expert in the product they are testing and they don't have to be (and usually aren't) expert programmers.
You can help them by doing the following:

- No return value – most of your TBBs should not return a value. The expected result should be passed to the methods as parameter.
- Success/Fail – the success/fail decision should be made inside the TBBs.
- Handling test flow – in case of step fail (the method fails), some kind of exception should be thrown. This exception usually will handled by the automation framework.
- Reporting – in any case (success/fail), the TBBs should file a report.
- Hide the complexity of the implementation.