“Regression execution takes too long!”

“The regression test pack is getting bigger and bigger!”

Excuses like these are heard all the time. It can be a real challenge to reduce regression testing, because it is a vital testing procedure that seeks out software errors by retesting the entire software system. The whole purpose is to ensure that no additional errors were introduced during the process of fixing problems or while introducing new features. Most companies spend around 40-60% of their test execution effort on regression testing. This is an important aspect to ensuring the customer’s experience will stay the same as previous release, or even better.

We all know about the pesticide paradox, which you execute the same set of repetitive tests over and over again, your software eventually builds up resistance resulting in nothing new being revealed by the tests. Therefore, we may ask ourselves why we should invest such a huge effort when the chances or probability for having defects found are getting closer and closer to none. Taking into account these excuses and how regularly we hear them, it seems clear that it is necessary to optimize the regression suite to maximize coverage and minimize redundancy with minimum time and resources.

An effective suite of well-designed regression test cases will not only provide a baseline assessment of the current version, they can also be used during the release of newer versions of the product. Obviously, testing only the section of code which has been changed is more cost-effective than retesting the entire suite; while certain tests should be run on the entire product, intensively testing code which has not been edited is a waste of time. The simple adage of “If it ain’t broke, don’t fix it” can be applied to code which a tester already knows to be functioning well.

The main purpose here is to find the redundant test cases that don’t provide great value to the overall testing effort. They are added into the regression pack to simply increase the code coverage or inflate the number of tests to please management.

How do we do it?

1. Regression test cases do not normally need to test bound, invalid data, etc.; normally they will be designed and focused based on the design of the test system. It is assumed that the above were tested prior to the regression test effort in previous release.
2. Ensure test cases are managed according to risks, with a risk index attached to each planned test case. Then we can check the risk against the new software release to ensure they are still critical as they were in the previous releases.

3. Using random test generators to create regression suites on the fly is becoming more common. In this practice, instead of maintaining tests, we generate test suites as we need them by choosing several specifications and generating a number of tests from each specification.

4. Test case prioritization is very commonly used to ensure test cases to be executed are reordered to maximizing the score function. There are many techniques which can predict the probability of each test case finding faults in regression testing.

5. Combination approach: any or all of the above methods can be used together.

Regression Strategy

A regression strategy is needed to pull everything together; therefore, it is necessary to put careful consideration into the chosen approach so that it can remain consistent and repeatable throughout the tests.

It is important to keep in mind the developers’ role when it comes to regression test suite optimization. Because they know what bugs were fixed, how they were fixed, and which new features are important, developers must act as a liaison with test engineers, who will have a fair idea of where the bugs may be.

Before moving on, we should classify all dimensions of regression:

- Regression at any test level: unit, integration, system, acceptance, etc.
- Regression for any test type: functionality, usability, performance, reliability, etc.
- Regression on any configuration supported by our system or product

Bearing this in mind and analyzing the above, the regression strategy should reply on at least 3 major factors:

- Risk assessment results and data
- Coverage information
- Past defects logging

Those 3 elements sum up most of the data that we need to optimize the regression pool.
Summary

Regression test execution demands us to be proactive along the testing lifecycle, keeping in mind that the test cases that we develop today for this release will be integrated into the regression pool of test cases for future releases.

In order to gain the full benefit of an effective regression test suite, we must automate as many regression test cases as is reasonably possible.

Obviously this is only the beginning, and we still have to investigate how these assumptions can fit in different types of product and system. It is also important to take into account whether or not there exists a clear methodology or guidelines for the industry in regards to establishing and maintaining a pool of regression test suites effectively and efficiently.