Top Ten Factors for a Successful Regression Test

Original Software

Overview

What is regression testing? Anybody who has spent a little time in software will have heard a developer say something along the lines of “Well the changes in my code have nothing to do with that side of the system so I can’t have caused the problem.” The skill of regression testing is in identifying all un-expected changes before the system is released, those deemed as errors can then be removed thus ensuring the system has not regressed.

‘The skill of regression testing is in identifying all un-expected changes throughout the system.’

Why regression test? It is simply to reduce the likelihood of errors in the software adversely affecting the users of that software. In that respect it is a risk mitigation technique, and one that is very important saving companies time, money and the risk of significant embarrassment.

So, what constitutes a successful regression test? The number of defects found, or maybe the number of test cases ran? Ultimately the QA department can only do so much, no system can be tested exhaustively. The only real measure of success, or failure, is the customer experience after the software is delivered. If all is as it should be and no defects have found their way through- then you have conducted a successful regression test.

Let’s have a look at some of the key factors that can ensure the best outcome.

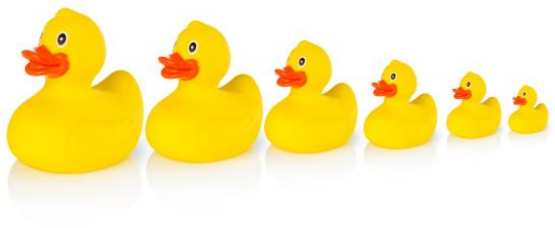
Top 10 factors to enable a successful regression testing strategy.

1.Time window.

There would be no point planning a full system regression that lasted many weeks if the release had to be made tomorrow. The type of development lifecycle being used will heavily influence the time window available to regression test a system. If an agile methodology is being used this would be a much smaller window than if the project is a longer waterfall effort. The efforts required coupled with the smaller timeframes in agile can be mitigated somewhat by the release train approach. Depending on the size and scope of change it may be necessary to use risk-based methods to attempt to regression test in smaller time frames. (see the risk factor below).

2. Pre-requisites.

The next factor to consider are the pre-requisites for a given test, this will mainly be the people needed to run the tests. Are specific skill sets required? Maybe only certain hardware can be used for the tests needed. Even if the tests are all automated somebody would still need to oversee the run and results. Getting the pre-requisites and the time windows aligned, is one of the key determining factors of success.



3. Environment.

Unless the test environments are ready then nothing can begin, also check that the specification of environment sensitive testing is covered e.g. localization/language checks. Elements to look for here are ensuring that the build and the data layer are accurately aligned. Many false positives will be given if there are missing new data fields or tables. This is where a smoke test is best adopted, this is a very lightweight run through of some key processes just to check that the environment, database and build are aligned and ready to test.
A good way of thinking about this is if you needed to release the build within a few hours, what would be the tests to run? These would be the ones that would make the best smoke tests.

4. Data.

Usable data for any new functionality may be needed, such as specific set up data or new types of process. Standard data for existing functionality and previously found defects must also be available for testing. This can sometimes be difficult to come by and new tests in the QA environment will fail due to existing data. This data should either be re-usable or renewable if required, obfuscation may also be an issue if the test data is copied from live. The process of creating data may need collaboration with the DBA team so this should also be taken into consideration.

'Ensure you can fully test end to end across the core applications.'

5. Landscape scope.

With regression testing integration is key. This may be integration of different components within an application but also more importantly across applications. Most systems require end to end processes that cross many platforms and databases. It is important to define the landscape to regression test before testing starts.Sometimes it may be necessary to use stubs or API calls to mimic certain applications responses if they are not yet available to use in the test process.

6. Prioritization of risk.

After smoke tests are completed a more thorough set of sanity tests should be run. These would verify all the important processes, i.e. those that would cause real harm if they went wrong in a live situation. After these it is acceptable to look into the following areas: -

- Most used processes
- Areas with recent changes
- Recent defects fixed
- Clean run and negative testing
- Historical areas of defect density

Test cases can be scored based on the above criteria and the severity/impact technique can give the appropriate risk score. If the test cases are prioritized in this way with a time estimate it should be easier to plan what can be used to give the best results.

7. Automation.

One area of testing that lends itself to automation is regression. Automation allows repeatability, and re-use, over the more mundane tasks required. An automation test can be run hundreds of times with different data and scenarios overnight ready for the team to review in the morning. One important aspect of regression automation is that it must mimic what the end user does, therefore utilizing the UI as opposed to API's or other unit testing techniques is more vital here.

More efficiency is experienced when the automation can be created quickly and by more of the team. For fast-paced software development environments this is especially important. Therefore, automation needs to be available to everyone not just specialized test resources.

'....automation needs to be available to everyone not just specialized test resources.'

8. Script maintenance.

With QA personnel changes, or in fast-paced development environments, it is essential that automation can be manipulated without any impediments. These impediments normally revolve around the technical aspects of changing code to align with the new applications to be tested. Ease of baselining scripts is very important to allow regression to continue into the next test phase. If waterfall this may be months away but if dev/ops or agile this may be much closer to hand.

The main reason automation efforts fail into disuse is the lack of people, and time, to correctly re-align the automation created to continue to justify the returns needed.

9. Full UI regression.

For true regression all aspects of the software need to be tested, this means that any automation needs to take in to account UI changes from previous builds. This full content analysis should be created and checked automatically to give the most effective results. This also allows for compatibility checks across both hardware platforms and operating systems.

10. Output Analysis.

QA need to make the decision as to whether something has passed or failed quickly and act accordingly. To allow this to happen it is important that the automation being used creates an output that is easy to understand and interpret. It is quite common for automation to be run overnight, so the results that can be reviewed in the morning. Initially a simple high-level pass/fail is needed. After this though it is important to see exactly where the error has occurred, and the steps to re-produce, to see if this is a real software defect or another issue.

Conclusions.

Regression testing is a central component in ensuring software quality. Done correctly the software delivered will be more reliable, and robust, increasing in quality more so after every release. Applying the ten key factors above will ensure that your software delivery will be as bug free as possible every time.

Just make sure when you set things up:

- Your process allows you to identify all changes including in the UI, not just the expected ones.
- You can quickly and easily baseline scripts to keep pace with the rate of change-you don't want to resort to writing elaborate code here. If your solution doesn't support this it can often fall into disuse.
- Output documents and reports are user friendly and will make sense to both IT and business users alike.

At Original Software our approach to regression testing takes into account over 20 years' experience of solving the issues and using the key tenants above

We are proud of our genuinely code-free automation, manual testing assistants, patented UI identification and baselining self-healing scripts. All of which allow us to offer the only true, out of the box automation regression testing solution on the market today.

About Original Software

With a world class record of innovation, Original Software offers a solution focused completely on the goal of effective application delivery through quality management. By embracing the full spectrum of application quality management across a wide range of applications and environments, the company partners with customers and helps make quality and efficiency a business imperative. Solutions include a quality management platform, manual testing, full test automation and test data management, all delivered with the control of business risk, cost, time and resources in mind.

More than 400 organizations operating in over 30 countries use Original Software solutions. Current users range from major multi-nationals to small software development shops, encompassing a wide range of industries, sectors and sizes. We are proud of our partnerships with the likes of Allianz, Bimbo Bakeries, Costco, CertainTeed, Delta Dental of WI, Euronet. IAT Insurance, O'Reilly Autoparts, Cayman National Bank, Topcon, and DSC Logistics.

Original Software