Avoiding Overkill in Manual Regression Testing

Lisa Shepard
Lshepard@webmd.net

Abstract

Software QA Analysts take great pride in thoroughly testing every aspect of a product's functionality. As complexity and variability of a product increases, so does the need for more thorough testing to ensure excellent quality and few bugs. At WebMD, we have found that one of our biggest impediments to thorough testing is over-documentation of all the tests that have already been run. To combat this, we encourage our QA Professionals to think critically about what manual tests to document. More specifically, we ask them to think strategically about what manual tests NOT to document.

For most QA Analysts, it's a natural instinct to document everything that was tested: for posterity, for future regression testing, to prove their worth to their manager, or a variety of other well-intentioned reasons. This over-documentation not only wastes the individual test writer's time, but it creates a stockpile of indigestible artifacts that ends up having a negative effect on overall quality.

This paper gives concrete examples of when we should and shouldn't document a manual regression test. This information will help QA professionals, developers and managers who erroneously equate the quantity of tests with the quality of testing and inadvertently sabotage their organization's testing efforts in the process. This provides details about what *should* be included in a regression test. Just as importantly, it highlights those details that *should not* be documented.

Using what is learned in the paper, QA professionals will be better equipped to write manual regression tests that are robust, readable, and low-maintenance. In doing so, these QA professionals and their development teams will be in a position to see huge benefits of spending more time testing and less time documenting excessive details.

Biography

Lisa Shepard is a Lead QA Analyst at WebMD Health Services in Portland, OR, where she has been diligently squashing bugs and improving quality since 2003. Although Lisa began her official software career in the late 1990s through college internships and a job at a startup company, she has been using her "QA brain" since early childhood. Throughout Lisa's nine years at WebMD, her test writing style has changed dramatically, which can be seen by looking at some of her historical (and embarrassing) test plans from the past. Among other projects, Lisa is on a mission to help QA Analysts learn how to avoid some of the same pitfalls and mistakes that she has made that led to bloated test plans!

Lisa has a B.S. in Computer Information Systems from the University of Idaho, as well as a B.A. in Spanish. When she is not sitting in her chair at work, you will likely find Lisa wakeboarding, coaching others on health & wellness, or enjoying a peaceful evening on the back deck with her family and friends. Lisa lives in West Linn, Oregon with her husband and 4-year-old daughter (who, incidentally, is also showing signs of a QA brain!).

1 Introduction

When developing software, it is essential to run a multitude of tests to ensure the code is functioning properly and to find and fix as many defects as possible. There are often misguided intentions, however, about what to do with all of those functional tests *after* initial development is over. In some extremes, all the test cases are thrown away and no one ever tests that functionality again. In other situations, all of the test cases get saved under the guise of "regression tests." In this situation, the massive set of manual regression tests soon becomes too big to actually run on a regular basis, and just as if there were no tests recorded, this functionality never gets tested again.

As you can see, it can be just as harmful to over-document tests as it is to under-document them. In this paper, QA professionals will learn when it is appropriate to write a manual regression test and when it is better to leave the test undocumented. We will also learn what to include in a test case (and what to omit) to ensure that the test cases are effective but not overly fragile.

Please note that the target audience for this methodology is organizations that don't deal with matters of life and death. Although every company would desire zero bugs in the production code, there are certain organizations that devote extra resources and budgets to their QA effort because the cost of a bug (in lost lives or money) is too expensive to allow a single bug into production (airplane instrumentation, some medical equipment companies, etc.). Although software bugs in production are costly for every company, most corporations put a limit on the time and resources allotted for functionality and regression testing, and recognize that limiting the resources opens up the possibility of bugs slipping through the cracks. It goes without saying that these corporations want high quality code in their production environment, but the bugs don't result in loss of life or limb.

This paper will provide concrete examples of how to effectively document manual regression tests. After reading it and practicing some of the ideas set forth, QA professionals will be better equipped to write manual regression tests that are robust, readable, and low-maintenance.

2 Is there such a thing as too many tests?

In the beginning of my career at WebMD, I was responsible for in-depth testing of some new functionality. With every test that I ran, I documented the test and the resulting behavior. I found great security in documenting every piece of information I was learning. I used the tests to guide my current testing, to document the expected behavior, and to keep track of what I would need to test when doing regression testing with each release. After a couple of months and thoroughly testing a few new areas of functionality, I was already up to about 200 discrete tests. Within a year, I had a suite of about 1500 tests within about 20 sub-categories of functionality. Not only did I feel like I had done great work for WebMD, but managers encouraged their QA resources to do the same thing… to prove that we had fantastic test coverage.

2.1 What's the harm in writing a lot of tests?

As you can imagine, I didn't regression test every single one of those tests with every release cycle. There's no way I would have had time for that. But if there was ever a question about expected functionality, I could refer to them. After a little bit, a few of them became inaccurate due to changing functionality, but I could spot when that happened, right?

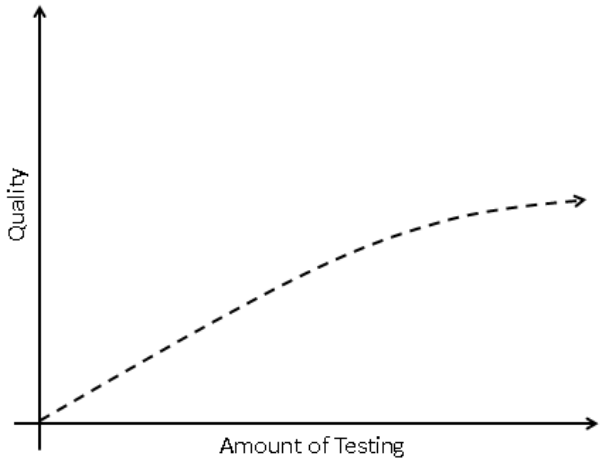
Fast forward about two years. I had been transferred to another project and someone else had been forced to muddle through my awesome collection of tests. Of course, they had no idea which ones could be trusted for accuracy. They didn't know the search terms or names that I used to find tests when I needed to refer to them. In fact, even though there was a lot of useful information in the test suite, the

entire collection had been moved to a “deprecated” folder. And the new QA team in charge of this functionality started all over again, creating its own suite of tests. The cycle had started anew!

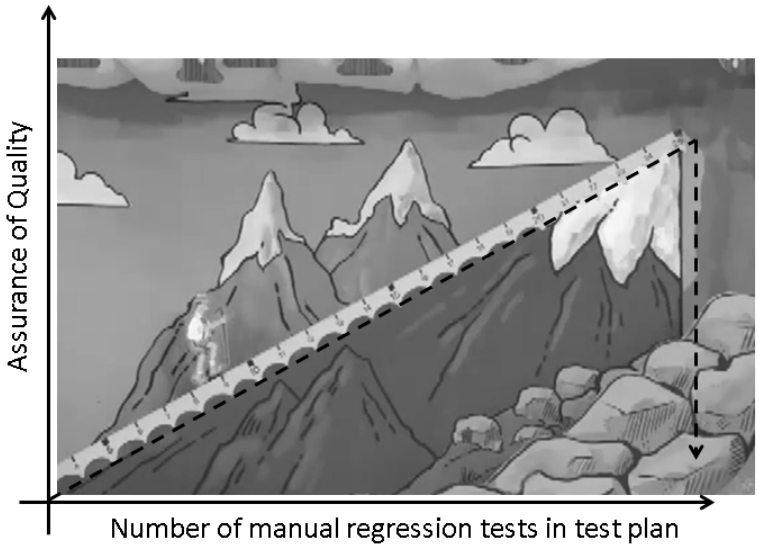
So in the end, not only did I inadvertently waste hundreds of company hours writing the test cases, but at the point that I crossed over the line into “too many tests to digest”, I rendered the entire suite of tests useless. I have watched countless QA teams do this on numerous projects, doing more damage than if they hadn’t written anything in the first place.

2.2 The relationship between tests and quality

We all know that testing is essential to ensure a high quality product. In general, the more you test, the better the quality will be, although this *does* taper off at a certain point and provide less of a return. Depending on the product being developed, each organization may have a different definition of what is *enough* testing, but the correlation between testing and quality can be described with the following diagram.



Notice that the horizontal axis is labeled “Amount of Testing,” not “Number of Tests”. In fact, when we start looking at the number of documented manual regression tests, I think of the graph as a mountain cliff. As we add manual tests to the regression testing plan, we increase the assurance of quality that we get with each round of regression testing. But as soon we’ve reached a critical point in the number of documented tests, the test plan is no longer manageable, maintainable or digestible. The entire test plan starts to get ignored and the assurance of quality doesn’t just decrease… it drops off a cliff.



When it comes to documenting manual test cases, our goal is to find a point where we have adequately documented the cases we plan to use for manual regression testing, but to document no more than that.

3 Why do we document tests in our regression plans?

There are many reasons why QA professionals document tests. Although there are one or two valid ones, most of the reasons for documenting regression tests are detrimental to overall quality. The QA professionals have good intentions, but those good intentions wind up resulting in bloated, unmanageable test plans. Here are some of those reasons and how they apply to the documentation of manual regression tests.

3.1 To document acceptance criteria

During the development and testing of new functionality, we create test plans to ensure we are delivering all of the features and functionality we are supposed to deliver. Whether by looking at product specifications or collaborating with a business owner on stories, we track tests for the golden path, edge cases, small units of functionality, and various combinations of settings.

As important as all of these tests are, it doesn't necessarily mean they all are worthy of being elevated to the status of "regression test." In a perfect world, where time is limitless, every single test that was run during initial development would be rerun release after release. But that world doesn't exist and time is limited, so we have to be smart about what gets included in the manual regression test suite.

Even though a test case was recorded during initial verification of product functionality, many of those tests should NOT be kept around for the long term. For this reason, it's important to be smart about how much time is spent on these tests that will eventually get thrown away.

3.2 To document functionality

Quite frequently, functionality isn't thoroughly recorded in a requirements specification or product document. As we create new functionality or fix bugs, QA's perfectionist tendencies want to ensure we have every single detail tracked. The last thing we want is for something to slip through that same crack in the future. Documenting the test gives us a false sense of security that we have patched up the crack.

Remember, if we write so many tests that it causes the entire suite to get thrown into the garbage after we leave the project, then we haven't done anything to increase the overall quality of the product.

3.3 To prove our value

We are rarely willing to admit that we are documenting tests simply to prove what a good job we did, however, there are organizations that erroneously equate a high number of test cases with a high quality of testing. If you are working in an environment where you think you need to write a lot of tests to show that you are actually doing work, stop what you're doing and address this with your manager.

3.4 To aide in creation of automated tests

Sometimes, it is necessary to document tests that you plan on automating in the near future. I would contend these tests need to have very little detail in them and they shouldn't be around for long. If these tests are frequently getting automated, then there won't be a glut of unnecessary tests clogging up the test plan. If the "Tests to be Automated" repository is always growing, however, then the mountain of artifacts is not worth the technical debt of creating and sorting through them.

3.5 To guide in regression testing

To many people, it might seem ridiculous to state that one reason to write manual regression tests is to help with manual regression testing. The reality is, however, that this is one of the few valid reasons to document a regression test. If you have no intention of running a regression test with every release, there is no reason to keep it around.

4 So how do we document just the right amount?

How do we achieve the right balance of documented manual test cases? Is there some magic formula that tells us what to document and what not to document?

The simplest way to find that balance is by following one rule:
All existing regression tests MUST BE TESTED during every major round of regression testing

When teams are forced to follow this rule, a few things happen:

4.1 We think twice about spending excessive time on writing functional tests

If there's a good chance that most of my individual functionality tests are going to be deleted or combined into a more conclusive regression test at the end of a sprint or the end of a release, then I am going to be much more strategic about how much time I spend on documenting those tests. I can document what is necessary to make sure I test all the cases I need to, but I don't need to ensure that it is in "archive" worthy format. I can spend more time testing and less time writing about what I just tested.

4.2 We can trust the information in the manual regression tests

Have you ever run across a test in a test repository and found that it didn't match up with current functionality? What normally happens in those cases? Rarely, the discrepancy highlights a bug that has just been introduced and the regression test has done its job of flushing out unexpected regressions in behavior due to new development.

The majority of the time, however, we find that the behavior had changed one or more releases ago, but the test just wasn't updated because nobody knew it existed. After the confusion is settled, no bug is entered and the test still remains unmodified. The cycle then repeats itself a couple of releases later when someone else finds the same "bug."

Once we are in a pattern where every test case is run with every release, we start to find fewer and fewer false bugs. When we do find a scenario where behavior behaves differently than the test case, it's likely that the unexpected behavior was introduced with the most recent development. At this point, we can decide if the test case needs to change to reflect new behavior or if the code needs to be changed to fix an unintended bug.

4.3 We continue to review our test plans with every release

As years go by, the number of test cases added to our test plans will continue to grow. Even when the test cases are written concisely and strategically, this may still result in the plan growing to a size that can't be manually regression tested with every release. At any point we find our departments unable to make it through all of the regression tests, we have two options: Delete some tests or allocate more time to regression testing.

In many cases, the appropriate response is to groom and delete the existing test cases. When this is the case, it is important to recall the visual of the hiker climbing up the cliff. By deleting those "excess" test cases, we are NOT decreasing quality. We are keeping that hiker from falling off the Cliff of Quality Assurance. We are actually helping the overall QA effort by deleting tests that no longer contribute to the quality of the product.

If someone makes a conscious decision to not run a set of tests because time is short, they are breaking the "Run Every Test" rule and need to remedy it by either deleting the test or by testing more.

5 What should be included in a manual regression test?

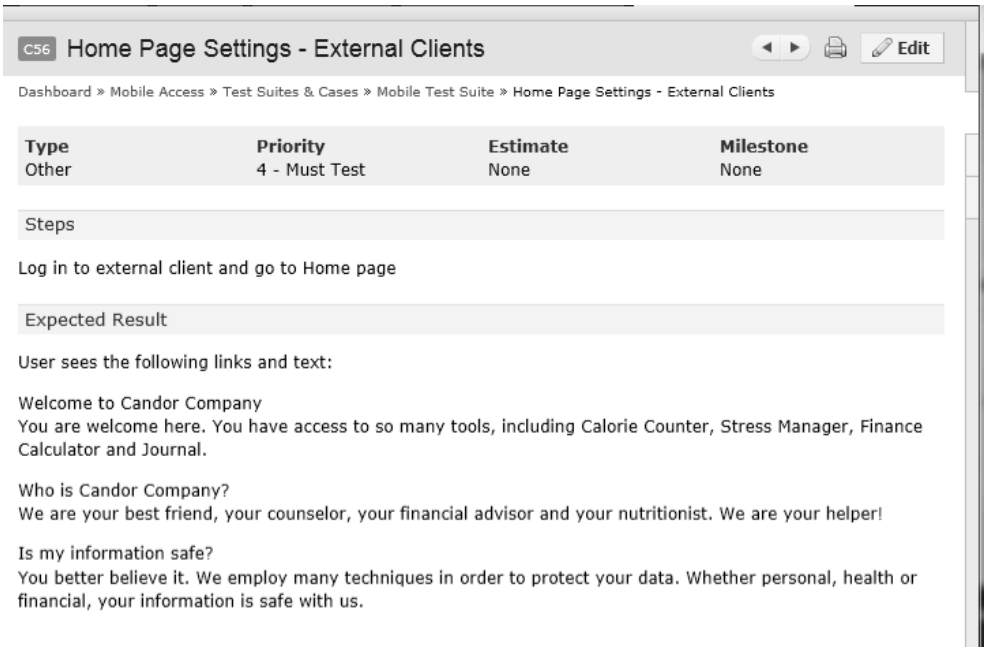
Up to this point, we've discussed the need to decrease the number of manual test cases in our regression test repository and purposefully maintain a low number throughout future development. In addition to limiting the number of tests, we also need to purposefully limit the *amount of detail* in the tests themselves.

Depending on the test, the format for a manual regression test can vary greatly. For that reason, I won't try to give you a standardized template for what a test case should look like. Instead, it's better to look at some guidelines and write your tests using these tips.

5.1 Include the intent of the content… not the exact text of it.

Often, a tester needs to verify the existence and intent of content but not necessarily the exact text. To keep your tests more readable and less fragile, record the main goal of the test and leave out the exact textual details. Not only will this make the test less likely to break, but it will also make the criteria for passing the test much more obvious to the tester.

In the two figures below, you will see two tests that are tough to read and are fragile due to the fact that there is exact text in the content:



C56 Home Page Settings - External Clients

Edit

Dashboard » Mobile Access » Test Suites & Cases » Mobile Test Suite » Home Page Settings - External Clients

Type	Priority	Estimate	Milestone
Other	4 - Must Test	None	None

Steps

Log in to external client and go to Home page

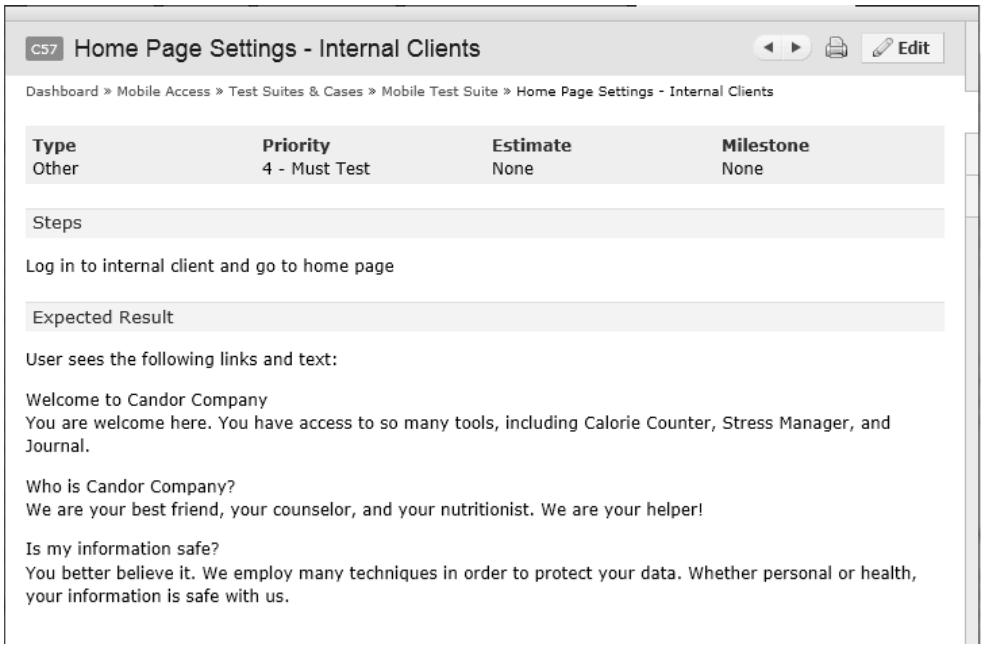
Expected Result

User sees the following links and text:

Welcome to Candor Company
You are welcome here. You have access to so many tools, including Calorie Counter, Stress Manager, Finance Calculator and Journal.

Who is Candor Company?
We are your best friend, your counselor, your financial advisor and your nutritionist. We are your helper!

Is my information safe?
You better believe it. We employ many techniques in order to protect your data. Whether personal, health or financial, your information is safe with us.



C57 Home Page Settings - Internal Clients

Edit

Dashboard » Mobile Access » Test Suites & Cases » Mobile Test Suite » Home Page Settings - Internal Clients

Type	Priority	Estimate	Milestone
Other	4 - Must Test	None	None

Steps

Log in to internal client and go to home page

Expected Result

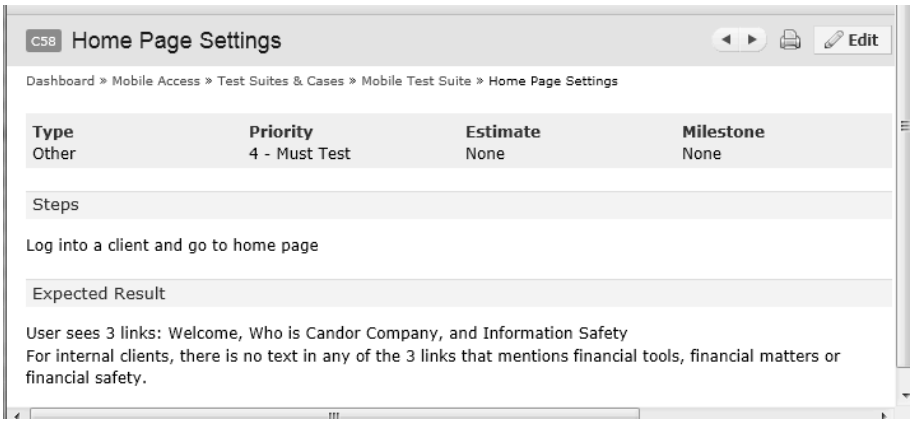
User sees the following links and text:

Welcome to Candor Company
You are welcome here. You have access to so many tools, including Calorie Counter, Stress Manager, and Journal.

Who is Candor Company?
We are your best friend, your counselor, and your nutritionist. We are your helper!

Is my information safe?
You better believe it. We employ many techniques in order to protect your data. Whether personal or health, your information is safe with us.

Using these guidelines, there shouldn't be precise text in these tests since it can frequently change in subtle ways, and we can indicate the intent of the tests much more clearly and effectively with the following test:



Of course, if the exact text is essential for legal reasons or specific client demands, then by all means, create a manual regression test that verifies that exact text. But usually, the main intent of a test is that the general concept or instruction is communicated. Write your tests in a way that ensures the user is testing for the general concept and don't make the tester sift through a paragraph of exact text to try to decipher the important elements.

5.2 Avoid writing extensive setup instructions or leading steps

In his blog, software tester and exploratory testing aficionado David Gilbert contended that "the amount of formatting and verbosity applied to a test plan is inversely proportional to the amount of actual good testing represented by such plan." I would have to agree. For most tests, I believe it is smart to assume that the person running the test has at least some knowledge of the product area they are testing. If they don't, they will ask questions that will enable them to start testing. It takes a lot less time to give a mini training session to your new testers than it takes to continually create and maintain specific instructions for every user flow. It simply isn't efficient to write your tests in a question-proof manner when most of the time, the people running the tests don't need that much detail to know what to do.

The figure below shows a test in a verbose format. This test is representative of many of the tests that are being written in organizations today.

Step Name	Description	Expected Result
Step 1 Monthly Journal link	1) Go to Child Health 2) Select [The Monthly Journal] on the homepage	The Monthly Journal page displays.
Step 2 Enter Journal link	1) Click [Enter Journal] on the left-hand side of the page	The monthly timeline page displays.
Step 3 Select Month	1) Select month 1 on the monthly timeline	The Create a Month 1 Journal page displays
Step 4 Enter Memories	1) Enter information to complete the [For your Baby Book Memories] section: babytest seems to smile the most when... testing babytest's favorite song seems to be... testing songs When I first held babytest, I felt... like testing The best baby gift I got was a... a new test gift babytest looks most like... daddytest The most memorable thing this month was when baby... tested A top news headline this month is: testing is fun	Information is entered with no errors
Step 5 Minor Health Problems	In the [Minor Health Problems] section, select Yes to a few of the conditions (e.g. colic, cradle cap, diaper rash...)	Radio buttons work with no errors
Step 6 Safety Precautions	In the [Safety Precautions] section, select No to a few of the precautions (e.g. use car seat in back, baby sleeps on back or side)	Radio buttons work with no errors
Step 7 Additional Things	1) In the [List up to five additional things that you would like to remember about Month 1] section, enter the a few things in text boxes. (e.g. Crying, Not sleeping...)	Text is entered with no errors
Step 8 Edit baby Journal	The Monthly Journal > Month 1 > edit 1) Click at [edit] button to make changes to baby	New information is saved

To make matters worse, in the company where I found this test, it was combined with six other tests that looked very similar.

I was able to combine all seven tests into one succinct test that does a better job of detailing what we are truly interested in regression testing. You can see this test in the figure below. By writing this test more concisely, it is easier to read, less fragile and helps the tester verify the behavior that truly needs to be verified.

	Step Name	Description	Expected Result
	Creating Child Account	Able to create child account from Family Members or from Living Healthy --> Child Health --> Create a Child's Account	
	Capabilities in Child Account	From within a child account, user can: - create a calendar event with reminder - enter a journal entry - add/remove/edit Child Health Record entries - enter Child specific tracker values - Click hyperlinks for content - print Health Record Reports and Emergency Card	
	CDC chart	CDC chart is displayed in graph section of Child Weight and Child Height trackers.	

If more details are necessary, then it is acceptable to include details. But it's important to make a conscious effort to decrease the verbosity of tests and keep them readable and maintainable.

5.3 Avoid excessive use of copy and paste

Copy and Paste has its place in our typing lives. It's a great invention and helpful when used in moderation. But Copy/Paste is also one of the major sources of confusing verbosity and inaccuracy in documented test cases.

Just as copying code and duplicating it in another part of your code base is a bad idea, the same is true for writing tests. It leads to many inaccuracies when test writers move too fast and don't make the small changes in each of the copied tests. Future updates to the tests also get made in one location but aren't propagated to all copied locations, leading to further inaccuracy. As for running the tests, consumers start to skim through the repeated details, assuming everything is the same as the previous test, and they miss necessary details.

If you find yourself using Copy and Paste frequently while writing your tests, it's a good sign that there is a better way to write the test. Consolidate all the repetitive details and only highlight where you expect to see differences in behavior. In the end, you have a test where things don't need to be repeated over and over again. After all, if you don't want to type something over and over again, what makes you think someone wants to wade through the muck to try to find the important differences between tests?

5.4 Only include details if a difference in behavior would result in a bug

How many times has a test resulted in behavior that ALMOST matches the test case, but doesn't EXACTLY match? What do you do? In many cases, the tester knows enough about the recent product development and is able to decipher whether the discrepancy is a bug or not. He often finds himself saying, "Well, it's not behaving *exactly* like what is documented, but it's following the intent of the test so I believe the behavior is correct." When this happens, then it is a sign that unimportant details have crept into the test case and muddied the waters. When writing or maintaining tests, only add details in the test case that would actually cause someone to log a bug against the test. This level of detail may vary from organization to organization, but your goal should be to make the tests as robust and low-maintenance as they can be without compromising their ability to verify the desired behavior.

6 How do I find time to clean up my existing test plans?

In order to be able to run every single manual test in every major round of regression testing, most QA organizations will need to do some serious clean up their existing and bloated test plans. Even if teams start writing good tests now, in order to be able to follow the new rule of manually running every test with every regression testing session, it is necessary to clean up the technical debt from years past.

Don't worry, there is no need to fret. This cleanup won't require months of re-writing of tests. The most important thing to remember is that your job during cleanup is not to rewrite every single test that has ever been documented for the product area. Your job is to glance through the hundreds of tests and pluck out the nuggets of wisdom worthy of being propagated to regression test status. Test plan by test plan, grab what you need and then delete the rest.

Throughout the cleanup of one my product areas, there were multiple test plans that had 100-200 tests in each of them. You'd think that grooming these would have been an insurmountable task requiring a large amount of resources for the effort, but it wasn't. By following these steps, I pared the test plan down to what was really necessary.

6.1 Get rid of the minor details

On the utest.com testing blog, one participant stated that "discovering the unexpected is more important than confirming the known." I agree. So the first step I took was to get rid of all the tests that just served the purpose of confirming a bunch of known details. I read every test and deciphered what the intent of the test was (the "nugget of wisdom"). For most of the documented tests, the small pieces of functionality weren't significant enough (or likely enough to change) to warrant their own regression tests. So I deleted them.

One thing to remember: As you delete test cases, you are not deleting *quality*. Most of these tests are inaccurate and haven't been referenced in years. You are increasing quality by grooming the test plan to something that is manageable and digestible.

6.2 Remove duplicate text

Throughout the same test review, I found dozens of situations where multiple tests were almost exactly the same, but had one or two minor differences. I made a note of those minor differences and deleted the multiple instances of the duplicate text.

6.3 Rewrite what is left of the test case

I then rewrote the test in a way that retained the important pieces of what I had found… and then referred to the various options/scenarios that should be tested if there was more than one pertinent scenario. Using the tips for section 5, this rewriting of test cases took a surprisingly short amount of time.

Remember, these are regression tests for existing functionality, not acceptance tests for new functionality. It isn't essential to cover every limit, edge case and use case. You simply need enough detail in the test to check that nothing has broken code that was previously working as tested.

6.4 Ease into delete mode

After years of writing test after test, you may be uncomfortable hitting that delete button. I understand. It takes time to retrain your mind after so many years of equating quality with the number of tests. So in the beginning, you can start out with a safety net. Create a folder or repository named "Deprecated". Then move the tests you plan to delete into this folder so you can refer to it in the future if you need to. After a few months of ignoring this folder and never needing it, you'll feel confident in trashing it for good.

Overall, the cleanup of some of the most bloated, unreadable, and inaccurate test plans took a matter of hours. The return on investment for these few hours of effort has been more than worth it release after release.

7 How can I break my old habits?

I recognize that many QA professionals have been over-documenting and over-complicating their test plans for many years. Even while attempting to adopt the new guidelines in this paper, old habits will continue to take you back down the path of excessive regression test writing. There are a few things that can help you continue down the proper path

7.1 Run a report at the end of a release

One of the most concrete ways to check on how well you're maintaining the size and complexity of your manual test plans is by running a report of how many manual tests didn't get executed during a release. In order for this process to work, the number really should be zero. If it isn't, you have two choices: Allocate more time to regression testing or Delete some of the tests that weren't important enough to run.

7.2 Do frequent peer review

Peer review has become a useful tool in many areas of software development. Whether it's two developers coding in tandem or a developer reviewing code changes before a development effort is considered done, there is great value in having multiple eyes look at code changes. At WebMD, we have begun to implement the same technique in the area of regression testing.

During a regularly scheduled meeting, the QA Analysts for a given product go over the stories that were completed since the last meeting. In these meetings, we aren't demoing the behavior itself (since that is done to the business and product owners during sprint demos). And this isn't a time for us to review every single functional, acceptance, and exploratory test we ran during our testing efforts. Instead, we briefly discuss the areas we tested (to ensure we didn't miss something) and then show the regression test(s) that were created or edited.

These short meetings help to keep us on track throughout the entire release. It prevents us from forgetting to enter any test cases… but it also keeps our desire to document in check. With peer review, we are able to discuss our test case writing and change course as we go, instead of when it's too late at the end of the release.

7.3 Keep on keeping on

In order to help QA professionals continue to make good decisions about what test cases to write and what to include in them, I have created some "bumper stickers" to serve as reminders. I encourage you to find the bumper sticker(s) that resonates with you. Print out a couple for yourself and post them on your office wall. Keep the list handy to refer to with your fellow QA professionals. Who knows? You may want to come up with your own to remind you of the ways in which you plan to shift the paradigm of manual test plan documentation.

Without further ado, here are the bumper stickers!

8 Conclusion

For the most part, QA professionals have good intentions when documenting tests. Yet those good intentions can often result in overgrown and useless test plans that no one can digest or maintain.

By being more purposeful and intentional about what we choose to document, we are much more likely to truly assure the quality of our products. Many professionals have been over-documenting manual test cases for years and the habits are hard to change. But with practice and peer review of test plans, we can decrease the technical debt in our overgrown plans and start to write tests that are maintainable for years to come.

Not every tip and instruction in this paper will be possible to implement in every organization. After studying it, however, each development organization will be equipped to analyze their manual regression test needs and make wise decisions based on those needs.

When this happens, we can reach a point where our QA professionals are doing more testing than talking about testing, and the quality of our products will greatly improve. Good luck as you change the way your organization writes and maintains manual tests. May you start reaping the rewards of those changes!

References

UTest Software Testing Blog. “16 Great Quotes For Software Testers” Mike Brown, http://blog.utest.com/16-great-quotes-for-software-testers/2011/01/.

Test Explorations. “The Theorem of the Well Formed Test Plan” David Gilbert, http://testexplorer.com/blog/?p=96