Introduction to Risk-Based Testing

Dr Stuart Reid

Introduction

Risk-based testing (RBT) has been around in various forms for over 20 years, and accepted as a mainstream approach for over half that time. It is the basis for the ISO/IEC/IEEE 29119 software testing standards, which mandate that all testing should be risk-based, and is also an integral part of tester certification schemes, such as ISTQB.

We all use risk on a day-to-day basis in our daily lives (e.g. 'should I walk the extra 50 metres to the pedestrian crossing or save time and cross here?') and similarly many businesses are based on the management of risk, perhaps most obviously those working in finance, such as banks and insurance companies. Despite this, and the fact that RBT is not a complex approach in theory, it is still rare to see RBT being applied as effectively as it could be. This article initially introduces the basic RBT concepts and risk management process before highlighting typical obstacles to effectively implementing RBT and suggesting means of addressing them in practice.

Why not Requirements-Based Testing?

Traditionally, testing has followed a requirements-based approach, where the testing simply covered what the customer asked for in the system.

But what happens in the typical situation when the customer doesn't state their requirements perfectly? For instance, if they do not document all their required features, or they do not specify all the relevant quality attributes, such as response times, security requirements and the level of usability they needed. The simple answer is that requirements-based testing does not cover these missing requirements – and so only provides partial test coverage of the system that the customer wants.

And what happens when the customer's requirements aren't all equally important (the normal state of affairs)? With requirements-based testing, every requirement is treated the same – so our autonomous car would have its pedestrian avoidance subsystem tested to the same rigour as the in-vehicle infotainment (IVI) system. Luckily, for safety-related systems we don't use requirements-based testing alone – the regulatory standards for such systems tell us that we must employ a risk-based approach with integrity levels. However, with any system, if we treat all the requirements as equally important, this will lead to inefficient use of the available testing resources.

So, if requirements-based testing is inefficient and leads to poor test coverage, then what is the alternative? Risk-based testing accepts that unstated requirements exist for all systems, and so missing requirements are treated as a known risk that needs to be handled. When missing or poor requirements are perceived to be a high-enough risk, then the tester will normally talk to users and the customer to elicit further details about their needs to mitigate this risk. Testers may also decide to use a specific test approach, such as exploratory testing, which is known to be effective when complete requirements are unavailable.

Why Risk-Based Testing?

Other than managing the problems associated with requirements-based testing, risk-based testing provides several other benefits, as described below.

More Efficient Testing

If we use risk-based testing, we identify those parts of the system that are higher risk and spend a higher proportion of our test effort in those areas. Similarly, we spend less of our test effort in the areas that are lower risk. This typically results in a more efficient use of testing resources – with less high risks becoming issues after the system is delivered. This may simply take the form of adjusting the amount of test effort used, but it can also include the use of specialist types of testing to address specific risk areas (e.g. if we have a user interface risk, we may decide to perform specialist usability testing to address the risk).

Test Prioritization

If we know which of our tests are associated with the highest risks, then we can schedule those tests to run earlier. This has two main benefits. First, if our testing is ever cut short, we know we have addressed the highest risk areas. Second, it means that if we do find a problem in a high-risk area, then we have more time left to address it.

Risk-Based Reporting and Release

By using a risk-based approach, then at any time, we can easily report on the current state of the system in terms of the outstanding risks (those risks that have not been tested and mitigated, where needed). This allows us to advise the project manager and customer that if they wish to release the system now, then all the risks we have not yet tested still exist (and so they accept those risks). Thus, any decision they make to release the system can be based on risks that they know about and have agreed exist.

RBT Process

Risk-based testing follows a typical risk management process, a simplified version of which is shown in Figure 1.

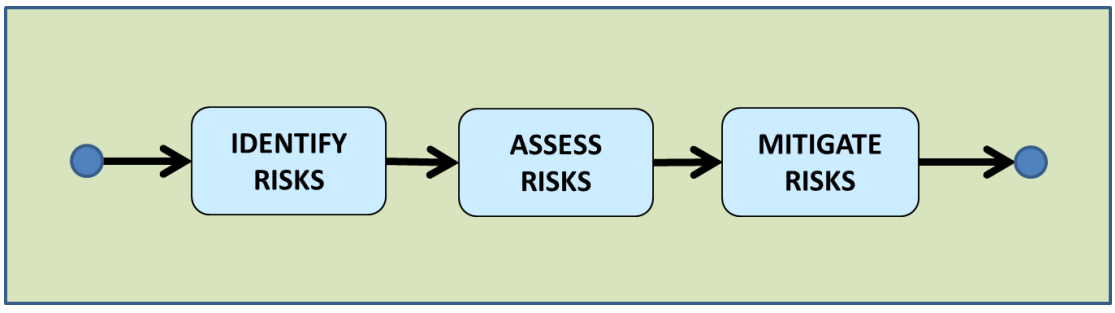

Figure 1: Simplified Risk Management Process

First, we need to identify any applicable risks. Next, we need to assess these risks so that we can derive a set of relative scores for these risks. Finally, we decide how we are going to handle the risks. We will consider how we implement each of these three stages in more detail next.

Risk Identification

Before we look at how we identify risks, we should first consider the types of risk we are seeking. There are two types that concern us: project and product risks.

Project risks are the same sort of risks that are used by project managers (often documented in the Project Risk Register). They are largely concerned with whether we will deliver on time and within budget. Project risks can also cover the availability of suitably-skilled staff and the likelihood of code being delivered from the developers on time. As a test manager, we need to consider project risks as they can have a huge influence on the type and amount of testing we include in our test plan. For instance, using a new testing technique may introduce the risks of lack of required implementation skills in the test team and increased test tool costs. We may also have experience of working with the development team in the past, and, know that they rarely deliver code that meets the specified quality criteria. Thus, we could identify this as a risk that could affect the testing (lower quality code means that testing finds more defects, and more fixing needs to be carried out, so increasing the risk of testing overrunning due to the extra re-testing that needs to be done).

Product risks are more specific to testing. These are risks associated with the deliverable product and so these risks affect the end users. At a high level, we may consider product risks in terms of functional and non-functional quality attributes. For instance, a functional product risk may be that the software that controls the car window fails to respond appropriately when the closing window encounters a child's head. Non-functional product risks could be that the user interface on the in-vehicle infotainment (IVI) system is difficult to read in bright sunlight (perhaps due to poorly-managed contrast and brightness), or that the response times when a driver selects a new IVI function are too slow.

When attempting to identify risks, ideally, we need to involve a good variety of stakeholders. This is because different stakeholders know about different risks and we need to try and find as many of the potential risks as possible (if we miss an important risk, it is highly-likely that we will fail to test the corresponding part of the system in sufficient depth, so increasing the likelihood of missed defects in that area).

There are several techniques we can use to identify risks; some of the more popular are shown below:

- Risk workshops
- Brainstorming
- Expert interviews
- Test team workshops
- Checklists
- External consultants
- Lessons learned

Risk workshops and brainstorming are good for involving stakeholders, but can be difficult to arrange and expensive (this somewhat depends on the stakeholders). An alternative is to interview stakeholders ('experts') one-at-a-time, which can be preferable if scheduling meetings with several stakeholders becomes problematic. Test team workshops to identify risks are far easier to arrange (as a test manager), but suffer from a lack of stakeholder variety. Risk checklists are nearly always worthwhile using, and examples can be found on the internet, but should also be tailored for, and managed within the organization to ensure they reflect organization-specific risks (these may come from lessons learned meetings).

An area that should always be considered as a source of product risks is the requirements for the product. Not delivering requested functionality is an obvious risk and one that the customer is unlikely to consider acceptable (so increasing the impact of this type of risk).

Risk Assessment

Risk exposure is assessed by considering a combination of the likelihood of an event occurring and the impact of that event. In an ideal world, we would be able to measure precisely the size of each risk. For instance, if the likelihood of an event occurring is 50% and the potential loss is $100,000 we would then assess that risk at $50,000 (so, risk score is normally calculated as impact multiplied by probability). In practice, when we perform risk-based testing, it isn't so simple.

We are rarely able to get the business to tell us what the impact would be if a particular risk became an issue. Take, for example, risks associated with the user interface of our IVI system. We may know from feedback on the previous version, that car drivers found the user interface difficult to use, and so we have identified this as a risk with the new IVI interface. But what is the business impact? This can be extremely difficult to measure, even in retrospect, but predicting it in advance of a vehicle going to market is practically impossible.

Calculating the probability of a risk becoming an issue could be considered an easier task, as we will be getting the information needed to determine this from the developers and architects (who we, as testers, are normally closer to), and perhaps from historical defect data. However, estimating the likelihood of a failure in a specific area is certainly not an exact science. We may talk to the architects and get their opinion on whether the part of the system associated with the risk is complex, or not. We may talk to the developers and ask how they rate the likelihood of failure, which may depend on whether they are using unfamiliar development techniques, and we could estimate the capabilities of the architects and developers (which also contribute to the probability of failure). However, we are never going to be able to come up with a precise probability of the system failing.

So, if we cannot accurately assess risk exposure when performing RBT, what do we do? First, we don't measure risk absolutely. Instead, we assess risks relative to each other, with the aim of determining which risks we need to test more rigorously and which we need to test less rigorously. Also, we make informed guesses. This sounds unscientific, but, in practice, the differences between risk exposures are normally so large that our informed guesses are good enough.

The most common approach to risk assessment for RBT is to use high, medium and low (for impact, likelihood and risk score). Figure 2 **Figure** *2*shows how this normally works. The business provides a low, medium or high impact for a given risk, while the developers provide a low, medium or high probability of failure. These are combined, as shown, and the resultant position gives us the relative risk score. It is important to emphasize that if actual numbers are assigned as risk scores (rather than simply reading off a 'high', 'medium' or 'low' risk score from the position on the graph) then these numbers are only useful for determining the relative position for different risks – the scores should NOT be used to directly determine how much testing to perform. For instance, if we had two risks, one 'Low-Low' with a nominal score of 1, and one 'High-High' with a nominal score of 9, we should not assign nine times as much effort to the second risk than the first risk. The scores, instead, should tell us that the second risk is relatively higher than the first risk, and so we should assign more testing to it. To convince yourself that this is correct, you might like to try applying similar

approaches to the same two risks, but using different scales for impact and likelihood – you will soon see that the result can only be useful as a relative risk score.

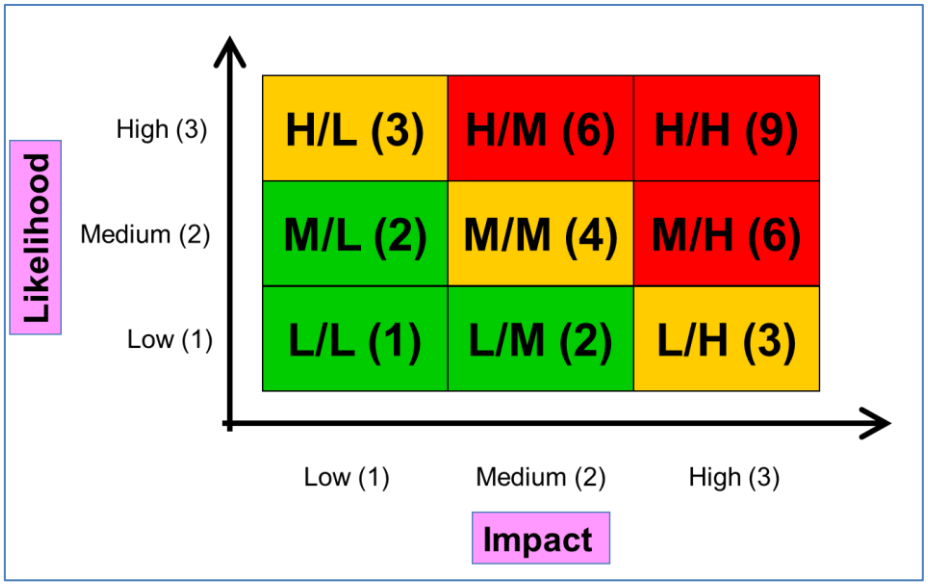

Figure 2: Risk Scoring using High, Medium and Low

So far, so simple. Here are some suggestions when you use this approach.

Make sure that the stakeholders giving you the scores for likelihood and impact use the full range (from low to high). You will find that some stakeholders (especially users) think that 'their' part of the system is the most important part and so always score every risk in their area as high impact. This is not especially useful, as we are trying to get relative scores, so we can discriminate between risks, and if most risks are scored high then we will find that most risks end up with the same high score.

Also, get the stakeholders to sign off on the scores. There are few things worse than a bug being found after release and the customer complaining that you should have found this bug because it was in a 'high risk' area, when you can clearly recollect that they agreed that that area was low risk– and so it was not tested as much as some of the other areas.

Finally, there is a third parameter that we often use in our calculation of the risk scores. This is frequency of use. Imagine we are performing a risk assessment and two system features are both assigned as equally high risk ('High-High'), but we know that the first feature is only used once per day, while the other feature is used every minute. This extra information should tell us that the second feature warrants a higher risk score, as if that feature fails, the impact will be far higher.

Risk Mitigation

The third step of our simplified risk management process (shown in Figure 1) is risk mitigation. Although it shows this final stage as 'mitigate risks', in practice we will not mitigate ("lessen or make less severe") all the risks. For instance, we may decide that for some risks the cost of testing is too high for the level of risk exposure. This is often the case for low scoring risks – and it is quite common to set a risk exposure threshold, and not test risks below this level. We may also decide that if the testing costs for a given risk are very high, then it is not cost-effective to test that risk. In

such a case, we normally try and handle the risk another way, and we may ask the developers to try and reduce the risk, for instance, by introducing redundancy to handle a failure, or we may recommend that the users live without that feature in the system.

When we mitigate risks by testing, we are using testing to reduce the risk by reducing the perceived probability of failure. We cannot get rid of a risk completely as testing cannot guarantee that the probability of failure is zero (we can never guarantee that no defects remain). However, if we test a feature and the test passes, we now know that for that set of test inputs in that test environment, the feature worked, and the risk did not become an issue. This should raise our confidence around this feature, and if we were to re-assess the risk we would expect the probability of failure to have decreased. This decrease may be enough to move the risk exposure below the threshold under which we have decided that testing is not worthwhile, otherwise we run more tests until our confidence is high enough (and our perceived probability of failure is low enough) to get the risk exposure below the threshold.

Risk mitigation by testing can take several forms and uses both the risks and their corresponding risk exposures.

At a high level, we often mitigate risks by deciding what is included in the test strategy. For instance, we may decide which test phases to use on the project, and we may decide which test types to use for different parts of the system (e.g. if we have risks associated with the user interface with high risk exposures we are far more likely to include usability testing as an explicit part of our test strategy). Risks can also be used to decide which test techniques and test completion criteria are selected, and the choice of test tools and test environments can also be influenced by the assessed risks.

At a lower level, we may decide (as individual testers) how we will distribute our tests across a single feature based both on the assessed risks and our knowledge of the system and its users. For instance, if we are testing a feature and we know that certain parts of it are used more often than others, it would be reasonable to spend more time testing the more frequently-used parts (all else being equal).

Good risk mitigation is highly-dependent on the skills and experience of the tester (and test manager). If we only know two test techniques then we have only four possible options (use neither, use technique A, use technique B, or use both techniques). However, it may be that neither technique is suitable for mitigating the perceived risk, in which case the effectiveness of our risk-based testing is going to be severely-limited. On the other hand, if we have knowledge of a wide range of testing techniques, we are far more likely to be able to choose an appropriate mitigation.

RBT should NOT be a One-Off Activity

On many projects, risk-based testing is performed as a limited one-off activity, used only at the start of a project to create the test plan, after which nothing changes. The simplified risk management process, shown in Figure 1 reinforces this idea, which is why we really need a better, more realistic model of our risk-management process.

As we mentioned previously, as soon as our tests start passing, our confidence increases, and the perceived probability of failure should decrease. Thus, as we run tests, the risk levels change, and so our testing should also change to reflect this (as tests pass, the perceived level of risk decreases until

the risk threshold is passed, and we then stop testing). It also works the other way - our test may fail. In this case the probability of failure (for these test inputs) is now 100% and we no longer have a risk, instead we have an issue that must be handled (risks must have a probability of failure that is less than 100%, otherwise they are an issue).

One of the principles of testing is that defects cluster together, so, as a tester, when we find a defect we should immediately start considering the likelihood that we have just discovered part of a defect cluster. Thus, the parts of the system near our newly-found defect will now have an increased likelihood of failure. This will increase the associated risk exposure – and this should mean that more testing in this area is now considered.

Risks do not only change because of the testing. Customers often change their requirements mid-project, which immediately changes the risk landscape. Also, external factors, such as the release of competing systems may change the business impact of certain risks (perhaps unique features are now more important) or it may be that the imminent release of a competing system means that delivery (and testing times) are shortened to allow release before the other system, so increasing project risks. Similarly, an unexpected winter 'flu epidemic could also change project risks associated with the availability of testers and test capabilities.

Risk-Based Testing Challenges

We have seen that risk-based testing can be a highly-effective and relatively simple approach to managing testing. However, there are still some pitfalls that should be avoided and useful modifications to the basic approach that should be considered:

RBT Challenge 1 – RBT should work at <u>all</u> test levels

Many test managers start using RBT when they are still testers and use it to prioritize and target their testing within a certain test phase, often system testing. This use of RBT by individual testers to manage their own testing is certainly a valid use of the approach, but its potential for use by test managers at the level of deciding test strategy for the complete project is even more powerful. At this level, RBT is used to decide and justify which test phases to use (or not), and to define test completion criteria for each of these phases, thus addressing higher level risks.

The use of RBT at even higher levels should not be ignored. It is also worthwhile identifying risks that apply across a whole programme of projects or across an organization and determining means of mitigating these risks via testing. Such an approach can lead to the mitigation of risks through the definition of an organizational test policy and an organizational test strategy that will define guidelines for testing across the whole organization. One obvious way of ensuring RBT is used at all levels is to mandate its use in the organizational test policy, or if there is no test policy, include requirements for the use of RBT in the organizational test strategy.

RBT Challenge 2 – RBT should address testing across the whole life cycle

As a test manager, when writing the Project Test Plan, there is a temptation to only include those testing activities that you control directly. The Project Test Plan, however, needs to address all the testing performed across the whole life cycle, whether it is carried out by testers or developers (or any other stakeholders). After all, if the test manager doesn't take responsibility for testing, who

else will? A common occurrence is that testing activities early in the life cycle, such as reviews of requirements and design specifications, which we know are extremely efficient in terms of defect detection and prevention, are not planned and executed effectively, if at all. Another typical situation is that the extent and quality of developer testing (typically limited to unit testing) is decided by the developers alone, which may lead to lower quality code being passed into later testing phases. As test managers, we must ensure that the planned testing is targeted at mitigating risks as early as possible in the life cycle, even when that testing is not actually performed by members of the testing team.

RBT Challenge 3 – RBT should not be a 'stand-alone' means of managing risk

Although RBT is a valid and valuable approach to managing testing, it becomes far more powerful when it is integrated with the risk management performed by the project manager and the developers. In this way, we can share a common understanding of the risks to the project (ideally documented in a shared risk register), how these risks interact (e.g. poor development leads to higher testing costs), and how they should be mitigated. This allows us to ensure risks are mitigated in the most efficient manner by those that are best placed to do it; often prevention by developers is far more efficient than later detection by testers.

If introducing RBT into an organization, be sure to take advantage of any other risk-based approaches to project management and development that are already there. Not only is it more efficient and effective to use an integrated approach to risk management, but getting buy-in to risk-based testing from many stakeholders is time-consuming, and if they are already managing risk in other project areas it is far easier than starting from scratch.

RBT Challenge 4 – RBT requires a 'professional' level of test maturity to work effectively

RBT will not work if those attempting to use it do not possess a high enough level of test maturity. To be able to select the most appropriate testing for a given risk, then the tester or test manager must know the range of testing options that are available to them, and how these options relate to the different risk types. This requires a practical level of familiarity with test case design techniques, understanding the effectiveness of each technique at detecting different types of defects (and so mitigating risks), and knowing in which test phases they are most effective. This knowledge is the bedrock of the professional software tester. In an industry where many testing practitioners find it difficult to name more than one test case design technique (let alone apply it), it is not surprising that many testers struggle to apply RBT effectively. This is one area where many testers could cost-effectively spend some time improving their skills.

RBT Challenge 5 – RBT should not only address risks in the deliverable product

Most new users of RBT tend to concentrate on the risks in the deliverable product (e.g. the risk that the anti-lock brake system mistakenly applies the brakes at high speed). However, for RBT to work effectively, we also need to consider the risks to the performance of testing on the project itself. These project risks include risks such as the late delivery of code from developers and the lack of testing resources available to the test manager. For RBT to work most effectively both product and

project risks (and their mitigation) need to be considered together as they can have an immediate effect on each other, and the optimal balance needs to be achieved. For instance, if there is a project risk that the available time for testing is shortened, it is not satisfactory to simply reduce the amount of testing, as this will typically result in fewer defects being detected and will nearly always result in a consequential product risk of a 'buggy' deliverable. In this case any mitigation needs to strike a balance between the two interacting risk types (project and product) to achieve an outcome that is acceptable to all stakeholders. The ability to successfully identify and implement such compromises is the sign of a true professional test manager.

Conclusions

RBT is known to be best practice for today's professional testers, but, despite the simple concept, many testers struggle to apply it in an effective manner.

This is often due to the scope of RBT being applied too narrowly, by it not encompassing the higher levels of testing, such as in the organizational test strategy, and by it not being used as the basis for test planning that covers the whole life cycle. RBT can also fail to fulfil expectations when it is not integrated into the wider risk management practices within an organization, or when RBT is only used to address risks in the deliverable software rather than also considering the risks to the testing itself.

These challenges can largely be addressed by the industry changing its perspective on RBT and widening its view. Probably the biggest challenge to the effective use of RBT is the lack of skills available to test practitioners, however, this should be an opportunity for testers to ensure that they acquire the full 'testing toolset', to allow them to effectively mitigate the risks with the right testing options.

