RBCS
TIME TESTED.
TESTING IMPROVED.
www.RBCS-US.com

Risk-based Mobile Testing

Rex Black, President, RBCS, Inc.

Risk-based testing is a long-standing best practice, but can you apply it to mobile testing? Absolutely! In this short article, we'll review what risk-based testing is, and then explain how it can be applied to testing mobile apps. Risk-based testing will help you focus on what to test, how much, and in what order on your mobile apps, which, given the timescales of mobile app testing, is more critical than ever before.

For any realistic-sized system, testing cannot reduce the risk of failure in production to zero, due to the impossibility of exhaustive testing. While testing does reduce the risk of failure in production, most approaches to testing reduce risk in a suboptimal and opaque fashion.

Risk based testing allows you to select test conditions, allocate effort for each condition, and prioritize the conditions in such a way as to maximize the amount of risk reduction obtained for any given amount of testing. Further, risk based testing allows reporting of test results in terms of which risks have been mitigated and which risks have not.

Risk-based testing starts with a process of analyzing risk to the quality of the system. First, you work with your fellow project team members to identify what could go wrong with the system. These are the quality risks, or, to use another common name, the product risks. In risk-based testing, these quality risks are potential test conditions. To determine which of the risks are test conditions, you and your colleagues assess the level of each risk. Important risks will be tested. The effort of testing associated with each risk depends on its level of risk. The order in which a risk is tested depends on its level of risk, too.

To clarify terms a bit, we can informally define risk as a possible negative outcome. The two key elements are possibility and negativity. A risk is neither impossible nor certain. If a risk becomes an outcome, that outcome is undesirable.

Risks are of different levels, as we know from real life. The easiest way to assess the level of risk is to use two factors: likelihood and impact. Likelihood has to do with the odds of a risk becoming an outcome. Impact has to do with the financial, reputational, safety, mission, and business consequences if the risk does become an outcome.

For example, people buy life insurance for premature death. As the saying goes, insurance is a bet that you want to lose. For all insurance, it's likely that you will pay more than you ever collect, and you're happy if that's the case.

Considering life insurance, premature death is unlikely, unless you engage in highly self-destructive lifestyle behaviors. (Of course, in that case the life insurance companies won't insure you.) So, premature death has a low likelihood. However, the impact can be very high. For example, suppose you are a primary breadwinner for your family, you have three kids, all under 18, and you die. Unless you have life insurance—or you had the good sense of being born with inherited wealth—that will be a devastating event for your family.

It can work the other way, too. For example, in many places in the world, going outside in the summer involves the risk of mosquito bites. The likelihood is very high. Usually—barring unusual disease outbreaks—the impact is very low. So, this is a risk managed through clothing, nets, and skin lotions, rather than insurance.

Testing software prior to release reduces the likelihood of undetected, serious bugs escaping into production; i.e., it reduces risk to overall system quality. Anything that could go wrong and reduce product quality, that's a quality risk. In addition to quality risks, there is another kind of risk, called project risks. Project risks are bad things that could happen that would affect your ability to carry out the project successfully.

Here are some examples of quality risks:

- System responds too slowly to user input during log in
- System calculates an incorrect total on the user's monthly bill
- System crashes when users enter long names, addresses, or other information during account creation

Here are some examples of project risks:

- Key project participant quits prior to the end of the project
- Equipment needed for testing not delivered in time
- Project sponsors cancel project funding during project

Let's summarize the risk-based software testing process. We analyze the quality risks, identifying the risks, and then assessing their level based on likelihood and impact. Based on the level of risk, we will determine what to test, how much, and in what order. By doing so, we will minimize the residual level of risk for the system as a whole.

In the case of mobile app testing, the process must involve very lightweight techniques. In addition, the risk analysis process should feed into the estimation process, since risk analysis determines the overall effort needed for testing. Likewise, the estimation techniques also need to be lightweight.

Interconnecting risk analysis and estimation provides a clean way for the project team to collectively determine what to test, how much and in what order. With that done, you can design and create a set of tests that cover the identified risks. The coverage for each risk is based on the level of risk. You then run those tests in a sequence based on the level of risk, too.

The lifecycle influences the way risk-based testing works. In a sequential life cycle, quality risk analysis occurs at the beginning of the project and everything follows from that for the next three months, six months, or even a year. However, for most mobile app development, you won't have a one-year-long development effort. For example, if you are using Agile or some similar process, quality risk analysis happens at the beginning of each iteration, as part of the iteration planning process.

When thinking of quality risks, think broadly. Consider all features and attributes that can affect customer, user, and stakeholder satisfaction. Consider the project team, the IT organization, and the entire company. Quality, to use J.M. Juran's definition, is fitness for use, which is the presence of attributes that satisfy stakeholders and the absence of attributes that would dissatisfy them.

These dissatisfying attributes are, basically, bugs. These bugs can affect functional or non-functional behaviors. When you're doing risk analysis, don't just think about functional quality risks. Think about non-functional quality risks as well.

Based on your risk analysis, the higher the risk, the more you're going to test and the earlier you're going to test. The lower the risk, the less you're going to test and the later you're going to test. The sequencing of the tests is clear, as you simply group the tests by the level of risk associated with the risks they cover, then run them in order of the level of risk. However, sometimes people ask me, "So, Rex, based on a certain level of risk, what does that mean in terms of the number of tests I should create?"

Unfortunately, the mapping of level-of-risk to level-of-test-effort isn't a simple calculation. Fortunately, I have an article on my website about a selection of test design techniques based on the level of risk. It's called "Matching Test Techniques to the Extent of Testing." It's only four pages long, and will give you some insights for effort allocation.[1]

---

[1] You can find that short article here: http://rbcs-us.com/resources/articles/matching-test-techniques-to-the-extent-of-testing/

Now, I said that prioritization of testing based on risk is clear, and that's true. It's very easy to determine the theoretical order in which you'll run the tests. However, you can't test software until you receive it, so risk-based test prioritize often requires risk-based development prioritization. This is true because most of the time you'll work in an iterative or incremental type of life cycle. Features are delivered to you for testing as they are completed. If features are built out of risk order, you can't run your tests in risk order.

You may be familiar with the phrase *forcing function* with respect to usability. It refers to the way an app guides someone toward the right way to use it. You've' encountered the forcing function if you ever think, as you're using an app, "Yes, it's just so obvious. In order to access that feature, you do this, then you do this, then you do this." Of course, you might not even notice when the app is easy to use, just like a fish probably doesn't notice the water as long as she's in it.

However, like a fish out of water, you're quiet likely to you'll notice the absence of the forcing function, because the absence will be experienced as frustration. You know that feeling, right? The "what is it doing?" and "what am I supposed to do next here?" feelings.

You can think of risk as a forcing function. Risk should guide you in your decisions about what to test, in what order, and how much. If you are doing safety critical or mission critical applications, you might use a formal risk analysis technique such as Failure Mode and Effect Analysis. However, it's more likely you'll use a lightweight technique, such as Pragmatic Risk Analysis and Management, a technique I've been using for 20-plus years. Such lightweight techniques don't create a lot of the documentation. You can capture their outputs on a spreadsheet or even a paper-based task-board.[2]

As I said before, you must consider functional and non-functional risks. You also need to consider physical aspects, the ways in which the mobile app interacts with the physical world via the device's sensors.

One of the mistakes testers sometimes make when they start doing risk-based testing is sit down alone and say, "Right, I'll just take all the requirements, and then ask myself, for each requirement, what could go wrong. Viola! Those are the risks." While it is true that those are *some* of the risks but those are not *all* of the risks. Because the requirements are imperfect and incomplete, and your personal understanding of the project and product is imperfect and incomplete.

[2] You can learn the PRAM technique in my books *Pragmatic Software Testing* and *Foundations of Software Testing*.

You can address this by including business and technical stakeholders in the risk identification and assessment process. This is harnessing the wisdom of the crowd to reach a better decision than any one person, no matter how smart, could. The inclusion of a wide range of stakeholders is another reason why you need to use a lightweight quality risk analysis process. Busy stakeholders will not participate in a process seen as a high-overhead, low-value process encumbrance.

Let's focus on the intersection of physical and functional attributes, and thus the quality risks present at those intersections. On the functional side, there's what the software can do, its features, the problems it can solve, the entertainment it can provide, what the display looks like. It might have audio elements to it, such as the ability to play music. It might have visual elements, from simple visual cues to the ability to stream videos. There can be tactile element, both in terms of you providing input via the touch screen and the app providing output by vibrating.

These functional elements can be cross-referenced with the physical elements that interact with them. You can put together a matrix showing intersections between functional elements and the physical elements. For example, different buttons, icons, and graphics that produce and are produced by certain features, the way that the battery and power management change the way the functionality behaves, the way the app uses rotation sensors, accelerometers, location information, cameras, and any other sort of physical sensor the device has. Notice that these physical elements are not functions on their own, but they are used by the functions, and many of the functions won't work without them. Therefore, using a matrix to identify these physical/functional interactions can help you see risks.

When you assess likelihood and impact, if you have an app that's already out in production, you should look at production metrics. While this slide does not provide an exhaustive list, it gives you so ideas:

- How many times has your app been downloaded? Of course, this applies to a native app. With a mobile website, you might look at the number of registered users.
- What is the number of downloads versus active application users? Obviously, if you have a relatively high number of downloads, but a low number of application users, that could indicate problems that are leading to a low active user rate.
- You can look at new users versus lost users, with the lost users being the difference between the total downloads and the active application users. You want the new user numbers to be high relative to the lost users, and the lost users numbers as low as possible.

- Frequency of use is another metric to consider. How often people do certain kinds of things is important to consider. Obviously, the screens and features that are more frequently used have higher risk from an impact point of view. If they're broken, then that creates more problems for people.

- Consider depth of visit and duration. How much time do people spend on the app, and what do they do? What kind of screens do they visit? Of course, this depends on your app, too. For example, with Yelp, if Yelp is doing its job right, you should be able to get in and out of that app quickly. In a matter of a minute or maybe less, you should be able to find the restaurant that you want, or the nightclub that you want. So, it might indicate a problem if Yelp had a very high depth of visit number versus Facebook, for example. Facebook wants you to buy stuff from their advertisers, so, to expose you to lots of ads, they want you to stick around. They want long duration and deep depth of visit.

- If you have a high bounce rate—the "I tried it once, and I've never used it again" number—there's clearly something wrong with your app. Of course, what is wrong isn't obvious. Was it reliability, was it performance, was it usability, or was it absence of functionality? Clearly, something is wrong, because a high bounce rate means people hate your app.

So these are some metrics to consider. There are also metrics discussed in the Foundation syllabus, the Agile syllabus, and the Advanced Test Manager syllabus with respect to risk analysis that are worth taking a look at.

Figure 1 shows a template that you can use to capture your risk analysis information, if you use the PRAM technique, Pragmatic Risk Analysis and Management. You can find a list of 20-plus quality risk categories on the RBCS website. You should use that as the starting point for your risk identification process, working through that checklist with your stakeholders. As you work through the checklist, you will populate the left column of this spreadsheet. Remember that not all quality risk categories will apply to your app. Review them, select the categories that do, and then, in each risk category, identify the specific risks.[3]

Once you have identified the risks, you can populate the likelihood and the impact columns as you assess the level of risk for each risk item. I use a five-point scale for likelihood and impact. The scales run from very high to very low for both likelihood and impact.

[3] The list is found here: http://rbcs-us.com/site/assets/files/1386/generic-quality-risks-list.doc

For likelihood, usually, it's clear how to interpret the scale. How likelihood is it that we would have bugs related to this risk item in our product? How often have we seen similar bugs in the past?

In terms of impact, though, distinguishing very high impact from high impact from medium impact from low impact from very low impact, that varies quite a bit from company to company, and app to app. So, when you first start using risk-based testing, work with your stakeholders to create well-defined criteria for those different impact levels to make sure that everybody is speaking the same language.

By the way, notice I use a descending scale, so as the numbers go up, the level of risk goes down. If you don't like that, you can flip these around and have five be very high, and one be very low. Either way, you have numbers ranging from one to five in the Likelihood column and in the Impact column.

So, now you've assessed likelihood and impact, each rated on a five point scale from one to five. Using a formula in this spreadsheet, in the Risk Priority Number column, you can simply multiply likelihood and impact. That calculation gives you an aggregate measure of risk. I call that the risk priority number.

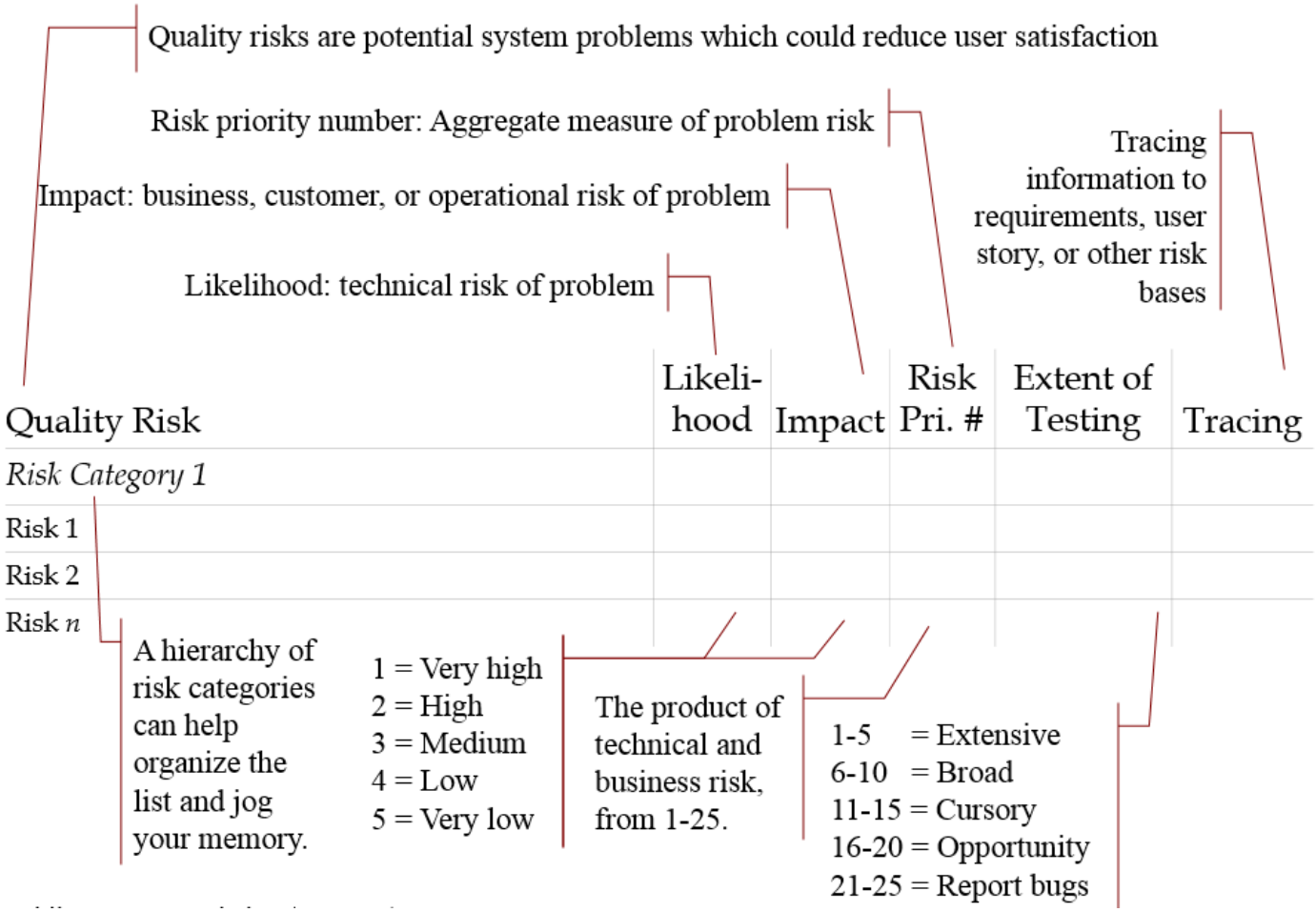

Figure 1: Quality risk analysis template

So, assuming a descending scale, consider a risk with a risk priority of number of one. Here's a risk that is very likely to happen. In addition, it's a risk with a very high impact. In other words, this risk is associated with a failure that could potentially pose an existential crisis to the company if it occurs in production.

Compare that to a risk with a risk priority number of 25. It's very unlikely to happen and has a very low impact. In other words, very few people would be affected by it, and even those people wouldn't care much.

At this point, it becomes clear how risk can guide our testing. Something with a risk priority number of 1, super scary and almost certain to happen, well, we better test the hell out of that, because, if we miss serious bugs in that area, we're in big trouble later. Something with a risk priority number of 25, utterly trivial and almost certain not to happen, well, maybe we get around to testing that, maybe not.

Given the risk priority number, we can use that number to sequence tests. Tests that cover a risk with a risk priority number closer to 1 get run earlier. Tests that cover a risk with a risk priority number closer to 25 get run later—if at all.

So that's test prioritization, but how about allocation of test effort? As you can see, I've given the extent of testing based on ranges of risk priority number. There are five levels of test effort allocation: extensive, broad, cursory, opportunity, and report bugs only. (These correspond to the levels of effort described in the article I mentioned earlier, Matching Test Techniques to the Extent of Testing, available on our website.)

The particular size of the ranges associated with each level of test effort allocation can vary, so consider what's shown here only an example. In this example, if the risk priority number for a particular risk is anywhere from 1 to 5, there will be extensive testing on that risk. If the risk priority number is 6 to 10, there will be broad testing. If the risk priority number is 11 to 15, there will be cursory testing. If the risk priority number is 16 to 20, you look for opportunities to test that risk as part of something else, but it doesn't get its own tests. Finally, if the risk priority number is 21 to 25, there will be no testing, but you will report bugs related to this risk if you see them.

Finally, at the rightmost side of the template is the Tracing column. If your risks are based on user stories, use cases, requirements, etc., then you capture that traceability information here. Now, you might say, "Wait a minute, I think you just contradicted yourself, Rex, because you said earlier that you don't want to just look at requirements when you're analyzing risks. You want to talk to stakeholders, too."

Yes, that is true. You start off by talking to people. You interview business and technical stakeholders to identify the risks, and assess their level of risk. Once you've done that, you go through the use cases, the user stories, what have you, and establish the relationship between the requirements that you have—in whatever form—and the risks that you've identified. If you come across a requirement that doesn't relate to any identified risk, you should add risks for that requirement because you're missing risks.

It's also possible that you will have risks that don't relate to a specific requirement. You might say that's a problem with the requirements. And you might be right about that. And maybe that's a problem that needs to be fixed. But it's outside the scope of testing. So, I'd suggest that, if you find risks that don't relate to any specific requirement, you report those risks to the people who own the requirements—say, the product owner in a Scrum lifecycle—and let them solve the problem.

As discussed, risk analysis as a way of deciding what to test, how much, and in what order. The beauty of the technique described in this article is that it's a proven method, used for around 30 years. It works on all types of applications, not just mobile applications, but it is particular apt for the tight timelines of mobile app development. It

works in various industries, from safety-critical to gaming. Give it a try on your next mobile app testing project. You'll find it a useful, lightweight tool for test prioritization.