Troubleshooting Risk-Based Testing

Solutions to Four Common Problems[1]

by James Bach[2]

Risk-based testing is a skill. It's not easy to know the ways that a product might fail, determine how important the failures would be if they occurred, and then develop and execute tests to discover whether the product indeed fails in those ways. This process is even more difficult when practiced in the typical development environment, with its pressure of impending deadlines, absence of complete information, and presence of different opinions about what could go wrong in the product. But it's a skill worth learning. Paying attention to risk focuses testing on that which is most often the core mission of testing: finding important problems fast.

In my consulting practice, I've noticed certain common problems testers and test managers experience when pursuing a risk-based approach. Let's look at four of these issues and see how to overcome them.

Problem #1: "Everyone thinks risk-based testing is just a management thing."

Often when I hear someone talk about risk-based testing, they seem to be talking only about *risk-based test management*—allocating resources to test activities according to particular product risks. The principle concern of risk-based test management is which test activities are most useful, and how much of them should we do. For instance, you may use risk as the basis of a decision to have four testers work on performance and scalability testing of a website, while only one does functional testing. That's cool. But what happens after the testers are given their tasks? Isn't there a value to risk-based test *design*?

Solution #1: Use risk to drive test design, too

When I use the term risk-based testing, I mean both risk-based test management *and* risk-based test design.

Test design picks up where management leaves off. Risk-based test design is the process of designing tests that reveal information about particular product risks. The principle concern of risk-based test design is whether the product will indeed fail in a way that we fear it might. The outcome is a set of particular tests to determine just that.

One way to approach this task is by creating a risk catalog (also known as a bug taxonomy). In its simplest form, this is a list of the types of bugs that may occur in a product. I say "bug" pointedly, in the sense of anything that threatens the value of the product. That may include faults, failures, or threat conditions. When I write a risk

[1] I thank Bret Pettichord, Brian Marick, Tracy Balkovic, and Rebecca Traeger for making comments that led to important improvements in this article. Here's a risk management strategy for you: ask talented people to review your work.
[2] Contact me at: james@satisfice.com, http://www.satisfice.com

catalog, it's an outline of statements of the form "[something] may [go wrong]." If I have more information, such as what causes the problem or what impact the problem will have if it occurs, I might also include that. I have done risk catalogs where I took a whole page to describe each risk. Generally speaking, though, I prefer to keep the statements to a single sentence or sentence fragment.

Cem Kaner published a rather sweeping risk catalog as an appendix in his *book Testing Computer Software*. But that catalog predates the Web era. Now one of Cem's students at Florida Tech, Giri Vijayaraghavan, has put together a useful catalog of risks for testing e-commerce shopping carts[3]. Giri's list includes items such as:

- Not enough memory on the system on which the shopping cart database resides
- Disk failures/hard drive crashes, and other irreversible media corruption of the shopping cart database may cause complete loss of data
- Corruption of shopping cart database backup

Everything in Giri's list relates to something you might do to test shopping cart functionality. The relationship is not direct. This is not a list of tests. But for every item on his list we can ask ourselves the question "What kind of test could I perform that would find problems related to that?" The list is detailed enough that it isn't much of a leap to think of specific tests.

I worry that if I depend on someone else's risk catalog, I might not make enough use of my own wits. So, if I was doing risk-based test design of shopping cart functionality, I would not initially use Giri's taxonomy. Instead I'd familiarize myself with the feature set, take a few hours, and think through the risks myself. I'd make my own list. Then I'd pull out Giri's list and check it against mine, to see if there was something important that I missed. By doing this, I get all the value of Giri's list, I can take more responsibility for the risk list, and I may possibly see some risks that Giri didn't.>>>>

Once you have a risk catalog, you can decide which items are worth testing. Then for each item, design tests that seek to answer these questions: Can this problem occur in my product? If so, how bad a problem would it be?

Problem #2: "My risk lists are muddled"

There are many ways identify and analyze risks. I like to talk about the reasoning processes of risk analysis and how they help bring good ideas to light. But what happens when you do this with other people? Then it's no longer just a reasoning process, but a social process, as well. Different minds frame the situation differently. Ideas overlap and conflict.

My colleague Bret Pettichord experienced this on a recent project. "We started our risk list in a meeting with the testing team." he recalls, "We had a half-dozen people and very quickly generated a long list of risks. What was hard was figuring out how to manage the list. We had this long list of things to worry about, but for many of the items, we weren't quite sure what to do."

[3] http://www.testingeducation.org/articles/bugs_in_your_shopping_cart_a_taxonomy_paper_isqc_sep_2002_paper.pdf

As he put it, “Many of our items were vague. One was ‘adjacent interference.’ It meant something to the person who added it to the list, but when we came back to it later, we couldn’t remember. What did it mean? We also had lots of different types of things showing up on our risk list. Our list of risks included risks, as well as types of tests (such as load testing), product features, and quality criteria (such as scalability or security). We needed a more precise understanding of what a risk was and how it related to some of the other concerns that were coming up.”

Bret was using the word “risk” broadly, here. Since testing is about finding problems in products, risk-based testing is about *product* risks. Bret’s team listed a lot of things other than product risks. Now, that’s okay for a start. Getting a mish-mash of ideas is normal in a brainstorm. Actually it’s a good thing. A core purpose of a brainstorm is to minimize the chance that an important idea goes unrecorded, even if a hundred weird ideas also come out. The problem is, you can’t take the raw output of brainstorming and expect to plug it right into a test plan. “We were used to lists of tests or bugs that gave you a list of things to do,” Bret said, “so that when you were done, everything would be checked off. We were having trouble figuring out how to check off the things on our list.” His challenge was to transform the list into something actionable.

Solution #2a: Categorize by type

One thing I find helpful is to make different lists for the different kinds of things that come up in a risk brainstorm. Basically, I want everything in a single list to relate to a single kind of activity. That way, I can pick up one list and work with it without getting scattered and confused.

I use lists like these:

Product Risks List: problems that may occur in the product (e.g. "The web server may be too slow under a normal load."). A product risk motivates a test or a test technique. Any specific product requirement can be reworded as a risk during the risk analysis. For any requirement, there is the risk associated with not meeting that requirement. If a particular requirement has little risk associated with it, then either don’t list it as a risk, or group it together with other risks such that together they are worth listing.

What you do with it: For each item on this list, you do one of the following:

- create tests to evaluate the risk
- investigate the risk further (without necessarily testing it)
- accept the risk (not testing for it at all)
- delegate the risk to someone else (perhaps letting the developers mitigate the risk by redesigning the product).

As the project progresses, you use the product risk list as a basis for reporting. By the end of a well-run project, your testing will have revealed enough information to give management confidence that the status of each risk is known.

Risk Factors List: conditions that tend to create increase product risk (e.g. The developers have not worked with this web server before.). You don’t test *for* a risk factor; you test in the *presence* of a risk factor. For instance, “feature X is especially complex”

does not suggest a particular problem that you will try to find in the product, it merely suggests that problems of any kind are more likely to occur in complex code. Another kind of risk factor is a "threat"— a condition or input that stresses the product in some way, and potentially exploits a vulnerability. To decide if a risk idea is a risk factor or a product risk, ask yourself what would happen if you reported it as a defect. If the response would obviously be "not a bug" then it might be a risk factor, but it probably isn't a product risk.

What you do with it: Since a risk factor is something that leads to a product risk, you can use the items on this list to help you get ideas for what the product risks might be and their potential severity. They might also help you think of risk mitigation strategies (though that's probably beyond the scope of testing), testability requests that you'll want to take up with the programmers, or specific tests to perform. For example, "corrupted client platform" might be a risk factor that comes up in a risk brainstorm. I'd put this on the risk factors list, and then ask what kinds of product failures are more likely to occur with a corrupted or misconfigured client platform? How would we detect such a condition? How can we create that condition?

Risk Areas List: groupings of similar product risks (e.g. performance). A risk area implies that there is at least one product risk, and probably more, that belongs to it. During a risk analysis, a general category of quality criteria, such as compatibility or performance, can be thought of as a risk area. In other words, the risks of not meeting compatibility standards might lead us to include "compatibility" (or "incompatibility," if you prefer) as a risk area.

What you do with it: The items on this list may be used as headings in a risk-based test plan, with specific risks grouped underneath. Risk areas help you summarize the risks and the associated test strategy. For any risk area, you ought to be able to imagine a variety of specific ways the product may fail. If you can't, then either it's not actually a risk area, or you need to gather more information.

Issues and Questions List: a catch-all that includes things that need to be investigated or answered so that the project can go forward (e.g. Do we have the budget to purchase a load-testing tool?).

What you do with it: Resolve the issues, escalate them, or ignore them. This is standard project management stuff.

Product Components or Features List: these are parts of the product that are perceived to be associated with some elevated risk (e.g. web server). A particular component may have a whole complex of risks associated with it. Try to get those into the open.

What you do with it: You can use this list in risk based test management to allocate your attention across the product. I like to summarize the product into a component list that fits on one or two pages, and designate each area as requiring "higher,", "normal," or "lower" resources. When a product component comes up in a risk brainstorm, you can discuss whether it's merely a "normal" risk (usually, I assume that everything needs testing), or a higherrisk compared to other components.

Test Ideas List: suggestions for testing (e.g. load testing). Test ideas are worth capturing for later, when you are figuring out how to test in ways that address the risks. When test ideas pop up in a risk analysis, and they seem important, ask what are the specific risks that motivate them.

What you do with it: The items on this list eventually find their way into the test strategy or particular test cases.

Project Risks List: risks that affect the success of the project, but not necessarily the operation of the product (e.g. The requirements documentation has not been delivered.). When risks come up in a product risk analysis, that seem to be *project* risk factors, put them on that list, instead.

What you do with it: It's probably not your job, as a testing organization, to run the project or be the quality police. Nevertheless, when project risks come up in a brainstorm about product risks, record them. Later, revisit the items on that list, and decide which ones threaten the testing part of the project (those become an issue list for the test manager), and which ones threaten the rest of the project (those you confide to the project manager or other people).

Risk mitigations: these are things you want to do in order to reduce risk (e.g. limit access to the production server). They may be beyond the scope of the test project.

What you do with it: Unless you have control over the project, you probably will pass these ideas along to the project manager.

This is not an exhaustive list of categories; I'm not sure there is one. If you find that some other category of information is offered in your risk brainstorm, and you can't shoehorn it easily into these categories, then make a new list. For any new list you create, answer the question "What do I do with this?" Otherwise, don't make the list.

You might make these lists during the brainstorm, categorize the items after the brainstorm, or do a brainstorm that focuses only one kind of item. Of course, you can also keep everything lumped together in one big ol' list, as long as you know what to do with each type of idea.

Solution 2b: Know your frame of reference.

Remember what a risk is. I like this definition: the danger that something bad will happen. For any product risk, I immediately want to know: "What is the something that can happen? How bad could it be? How likely is it? These questions require us to determine causes and effects. One reason risk lists get muddled is that people get lost in a labyrinth of causes and effects.

Take "incorrect input" for instance. Is it a cause or an effect? Is incorrect input something that might trigger a failure in the product, or is it the effect of a poorly designed user interface? Or does this incorrect input result from the output of another subsystem that must have already failed? Depending on your frame of reference, this may or may not be a product risk. There is no one right answer here. What the user experiences as a product is a long and convoluted chain of causation. We test all along that chain.

One way to keep your bearings is to think in terms of this generic chain: Victim <- Problem <- Vulnerability <- Threat:

- **Victim:** Someone that experiences the impact of a problem. Ultimately no bug can be important unless it victimizes a human.
- **Problem:** Something the product does that we wish it wouldn't do. (You can also call this "failure," but I can imagine problems that aren't failures, strictly speaking.)
- **Vulnerability:** Something about the product that causes or allows it to exhibit a problem, under certain conditions (also called a fault).
- **Threat:** Some condition or input external to the product that, were it to occur, would trigger a problem in a vulnerable product.

In terms of this chain, we can say that someone may be hurt or annoyed because of something that might go wrong while operating the product, due to some vulnerability in the product that is exploited by some threat. This is essentially a mini-story about risk. Whatever risk idea comes to your mind, find its place in the story, then try to flesh out the other parts of the story. To do that, you need to set your frame of reference: first decide what product or subsystem thereof you're talking about—what is the thing that has the vulnerability, faces the threat, exhibits the problem, and impacts the victim?

Problem #3: "No one on my project wants to talk about risk."

Risk is scary. It's emotional. If you're a designer who has just successfully argued for using a new and untried

Example

Let's say your team holds a risk brainstorm and the result is a long list of items that look like this:

1. performance and usability
2. very large transactions
3. we don't know enough about web services!!!!
4. automated regression testing not possible on hyperbolic tree component. Also we have no control over its source code.
5. web server failure

Take a moment and try to categorize these items.

Here's what I think:

1. performance and usability	Risk areas. The next step is to get more specific about each of them.
2. very large transactions	Could be risk driver, test idea, product risk or product feature. Who knows? This one needs more context.
3. we don't know enough about web services!!!!	Sounds like a risk factor that someone thinks is pretty important. It may also be an issue, a project risk, and a product feature. More information, please.
4. automated regression testing not possible on hyperbolic tree component. Also we have no control over its source code.	Two separate items stuck together: the first one looks like an issue or a project risk, the second one seems to be a project risk and possibly a product risk factor. Together, they also suggest that special attention may be needed to properly test hyperbolic trees (if we decide it's important to test at all, that is.)
5. web server failure	Could mean anything. This is just too vague unless the meaning is obvious from the context of the project.

Most of these items, I believe, would benefit from some expansion or context setting. In some cases, discussing them among the project team may lead to dozens of specific new items for our lists, and a much better idea of what to test for.

technology in your product, then you might not want to admit publicly that your idea carries risk. If you're a customer, you might not like to hear that the product you are about to use is full of risks. If you're a corporate lawyer, you might be nervous about the liability of risk lists floating around, ripe for subpoena. I've seen senior managers expect and assume that all risks are "managed," which really means eliminated.

It can be difficult to get people to talk frankly about risk. Or if they do, it can be difficult to get them talking in specific, rational terms. You may be accused of being "negative," implying that the very idea of looking for problems brings them upon the project.

Solution #3: Shift the focus from risk to choices.

You can't test everything equally. And even if you could, it would be too expensive. This is the rallying cry of the risk-based tester: *Let's make smart choices.*

If people on the project don't want you to talk about risk, fine. Start talking about choices. No one can escape choices. What should we test? How intensively? Why test more of this than of that? Without uttering the word risk, you can get people talking about it.

Another tactic is to do your own risk analysis in private, then announce your choices about what needs testing. Listen carefully to the reaction of management and developers. This is sort of a "stone soup" approach— by proposing to serve them a stone, you might get other people to contribute some soup. (If this reference is lost on you, you can read the fable online. See the StickyNotes for a link.) Proposing my own risk list rarely fails to get people to talk to me, if only to object to my choices. When they do, I get more information about risk.

Problem #4: "I *don't* know that it's *not* a risk. Does that make it a risk?"

Every risk you identify starts as little but a rumor. You don't know, prior to testing, what bugs are in the product. You don't even know what the probability of any particular bug is. You might know that certain problems are possible. That's about it. This makes life hard for us when we do a risk analysis, because uncertainty can feel like product risk.

But there's a big difference between something you know is a risk and something you fear might be a risk. It's the difference between knowing you are in danger of being stung by a hornet because you see a hornet buzzing around your bedroom, and *not* knowing that you *aren't* in danger of being stung, because you haven't yet searched every place a hornet might hide.

Why does this matter? It matters when we try to understand which risks are severe and which ones are only mild. When you're trying to compare the magnitude of risks that you know little about with that of risks you know a lot about, there's a temptation to inflate the importance of those risks you have less information about. That can lower the credibility of the whole risk analysis.

Solution #4: Distinguish informed risks from uninformed risks.

When I categorize product risks, I categorize them in two major dimensions: magnitude and level of information. The scale I most often use for information is *lots*, *some*, *little*, and *none.* (I like to keep it unpretentious.)

By distinguishing risks by level of information, we provide a vital bit of perspective for people reviewing the list. We are more confident in our assessment of informed risks then of uninformed risks. If we say that a risk is severe, but that we have little information about it, we're basically saying there's some cause for concern. By investigating that risk, including testing against it, we gain information. Our goal as testers is to turn all important risks into well-informed risks. It may be that doing so causes us to discover that a given risk isn't much of a risk, after all. Or maybe we confirm that a risk is severe, and that motivates management and the developers to take action to improve the product and lower that risk.

Revealed! The Great Secret of Risk-based Testing

So much for specific problems, now for a helpful idea that cuts across the whole spectrum of risk-based testing: *Bad risk analysis is a self-correcting problem.* This is true because if your testing isn't risk-based enough, then you probably will ship with important problems. Then you will say, "Whoa, next time let's put more focus on problems like that." That cycle of learning is a fundamental part of engineering everything from bridges (see *To Engineer is Human*, by Henry Petroski) to airplanes (read any NTSB aircraft accident report). That's why you should not be too hard on yourself when you're using risk-based testing on a new product line, where you don't have the benefit of experience. Some risks you will be able to anticipate. Others you won't. There is no way to know for certain what the risks are in advance. All anyone can expect of you is to be thoughtful and resourceful, make people aware of the limitations of your analysis, and learn from experience.

The failures that escape our testing net will condition how we test in the future—if we're paying attention. Just bear in mind what Mark Twain said about that: *"We should be careful to get out of an experience only the wisdom that is in it—and stop there; lest we be like the cat that sits down on a hot stove-lid. She will never sit down on a hot stove-lid again—and that is well; but also she will never sit down on a cold one anymore."*