Using risks to guide Exploratory Testing

> **Warning: Content Compatibility Notice**
>
> Please note that the articles in this documentation feature media from app version 2.0.5. Interface elements may differ in newer versions. Refer to the latest app version for the most accurate information.

What is a risk?

A risk can impact (negatively or positively) the value of what we aim to provide.

Usually, we talk about risks of events that can happen with a certain probability and can negatively impact some stakeholders (e.g., users) at a certain level.

> **Example**
>
> Using non-encrypted protocols can expose sensitive information, including credentials, to attackers.

Risks can be handled differently: we can prevent/avoid them, mitigate them, or accept them.

Testing is many times seen as a risk mitigation strategy. Still, it's also a strategy for preventing risks, depending on who and when is involved while testing and the decisions that are taken upon testing findings.

Using risks to drive testing

We use testing to obtain quality-related information. Risks and quality have a close relationship. Risks can affect the value, as seen by some stakeholders, at a given moment. In other words, risks can impact quality. If we implement a substantial feature without listening to users or having done some preliminary research, then there's a risk of that feature not addressing the needs of our users. If we use external libraries, there's a maintenance and dependency risk that can expose us to more problems.

We use risks to drive our experiments when we test, whether implicitly or explicitly.

Potential risks are:

- the product and the feature don't match user expectations and needs
- the feature claims are not met
- feature purpose is not clear and thus may not be used
- users get frustrated whenever using the new future
- users feel difficulties using the new future
- the feature is not consistent with other existing features within the product
- the new feature impacts other existing features somehow
- adds a considerable performance overhead
- the brand and UI guidelines are not respected
- it doesn't meet applicable law in all markets where the product may be subject to it
- it doesn't comply with regulatory requirements
- user data or the product itself can be accessed/used by non-authorized users
- the change is very hard to revert, operate, or monitor/observe
- the product or the feature will be a major success with a major spike in usage
- ...

In sum, to consider risk:

1. Think about who your users are, external and internal. Don't forget the unexpected users;
2. Think about what matters most to them;
3. Think about what could go wrong.

Contextualizing risks

There is no straightforward and ordered "checklist" of the risks we should tackle. That requires expertise from the tester and... context!

It's impossible to tackle all risks; it's impossible to test "everything." Therefore, whenever testing, we have to think about where we will invest our effort in so that we cover aspects that can give us valuable information about quality.

Contextualizing risks with time

Software and the overall development process is not a localized event; its a journey. It has a past, a present, and a future.

This means that software has a history that culminates in the present...

- features that exist and that are used
- features that exist and that are **not** used
- unintended behaviors (e.g., bugs, feature subtleties) that are used as features
- areas/features with technical and testing debt

It's important to understand...

1. **How was it used in the past?**

   > **Why?**
   >
   > It gives clues about:
   >
   > - features that users were using (and not using) and to what extent
   > - flows that users were performing and not performing

2. **How is it currently used?**

   > **Why?**
   >
   > It gives clues about:
   >
   > - features that users are using (and not using) and to what extent
   > - flows that users are performing and not performing
   > - if there was a change in user behavior and underlying needs
   > - if there was a change in the software that may have affected the current usage behavior

3. **How do we foresee it being used and evolving in the future?**

   > **Why?**
   >
   > It gives clues about:
   >
   > - concerns about performance and scaling
   > - how to monitor/track success
   > - how to quickly perform experiments with users
   > - existing features that may need to be rethought or that may be affected somehow, and thus may need tailored testing

Contextualizing risks with needs

Software is made to address needs. Needs can be of different types, though; they're not always "functional." What purpose are we trying to achieve, and for whom? Are there any existing references we should have in mind?

Generally, we can say that a need is met if a certain goal can be achieved effectively, efficiently, and satisfactorily.

Sometimes we focus our attention on effectiveness/correctness, leading us to look at written specifications, acceptance criteria, or claims. While correctness may be crucial for banking and financial products, it's not as relevant for a social application, where UI and UX matter most.

But needs exist not only for external users but also for internal stakeholders and even for the team supporting and developing the product. Are we using deprecated dependencies, or will dependencies have well-known security issues, for example? Can a component or a service provider easily be replaced by another one? Is our infrastructure properly tracked, is its setup scriptable, and are those scripts prone to error handling, for example?

Contextualizing risks with current practices

Is testing currently just implemented at the surface? Are areas/features covered with automated test scripts? What level and type of tests (e.g., unit, integration, system, functional, performance, security)?

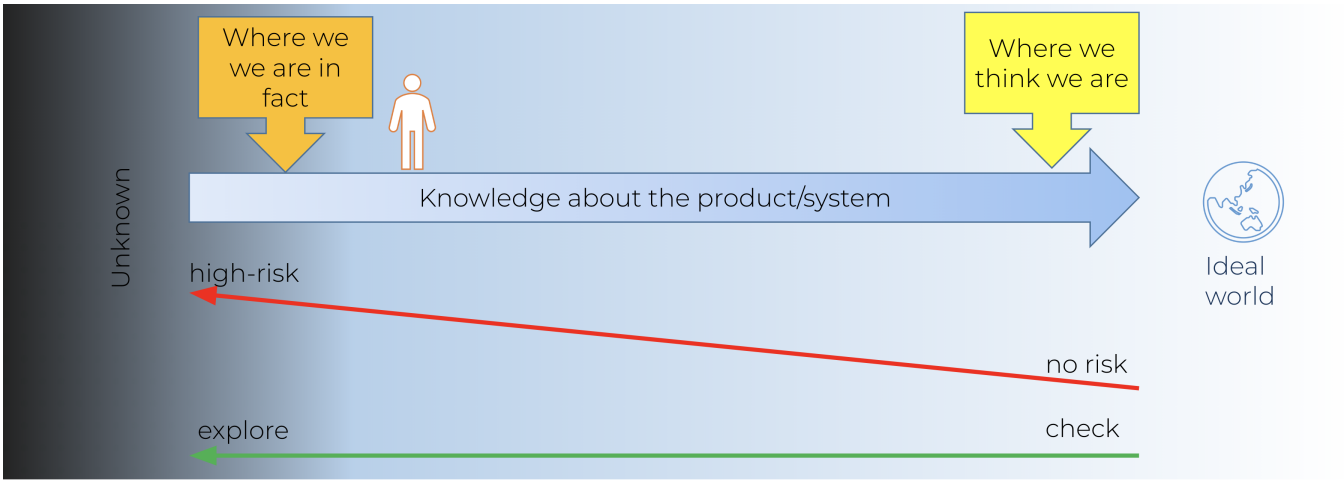
Knowing where we stand and depart from in testing tells us about potential risks.

What are we covering already? To what extent? Do we have quick feedback loops about it? Do we have a history of it?

Remember the unknowns

The risks that we are aware of, we can handle somehow, including with more or less testing depth.

We can group risks in a matrix or in a diagram, knowledge related, to give us a better understanding of the risks that exist and how we can approach them.



Adapted from Callum Akehurst-Ryan

- We can check the risks we know we know
- We can check & explore the risks we know we don't know
- We can pair with others to tackle unknown knowns and expose our biases or things we assume or skip without knowing
- We can keep learning and exploring to uncover unknown unknowns

From risks to test charters

Say we have selected a risk.. how can we turn it into a test charter?

Let's consider the following test charter template as a basis; remember that this a template; it's not strict, so you can adapt it freely to match your needs.

> **Charter template**
>
> Explore <area, feature, risk>
>
> with/using <resources, restrictions, heuristics, dependencies, tools>
>
> to discover <information>

Adapted from Maaret Pyhäjärvi, Elisabeth Hendrickson

Since we'll be performing a testing session, it will implicitly be limited in time, resources, and depth.

First, we have to consider the scope of what we aim to test broadly, i.e., the subject of our testing. Do we want to perform the testing around a specific feature? About a subset of an existing feature? Around a flow? The whole product? If the latter, you'll probably need to refine your scope and limit it further.

Second, what will we bring to the testing session? Are there any resources, tools, or heuristics that can help us? Can some restrictions be used to restrict the scope of our testing or amplify the probability of finding problems on the subject of our testing?

Third, what kind of information do we want to find? Do we want to find problems around the risk in general that we have identified? Won't that be too broad? Maybe we need to refine it.

Remember that we're talking about risks. As such, there's a probability of them occurring and the impact if they occur.

Our test charters should be around maximizing probability on one side but also consider situations with relatively low probability but still have a major impact.

Here are a few additional tips

- Internals
  - listen to your team, and pair with other team members, including developers, to expose risks that otherwise could escape
- Product Context & Market
  - listen to your users to understand what's important to them, what they most value, and the things that frustrate them the most; these will give ideas of potential risks
  - listen to your business and what's important to them, be on the same page, and don't ignore aspects that ultimately matter to your company and management
  - experiment with similar products, as you'll gain knowledge about what common and sometimes implicit expectations users may have whenever using your own product
- Success
  - try to understand what "success" means to different stakeholders
  - understand "where the money comes from" vs. "what are the common user/usage flows"
- Background Knowledge
  - learn more about quality attributes, to become aware of different aspects that people value differently; these are the different dimensions of quality that we can have in the back of our mind that inform us about what risks can target quality aspects
  - learn more about heuristics tailored for testing, as these can be used to provide some ideas for test charters but also ideas for many diverse experiments during the test session itself

Further resources

- TestSphere deck
- Risk Storming
- Would Heu-Risk It?