Risk Based Testing -
Pragmatic Implementation In an Agile Group

- Geoffrey Morris, Manager QA & Testing, Avanade

October 4, 2016

What is “Risk Based Testing”?

- “Quality” Risks, not Project Risk.
- Test the riskiest stuff as early & deeply as possible.
- More risk == more testing. Less risk == less testing.
- Benefits to testers:
  - Critical for under-resourced Teams.
  - Easier to understand test strategy.
  - Much friendlier reporting, useful for all the roles in a group all the way out to stakeholders & sponsors.
- Benefits to developers:
  - Positive influence to design & development focus, time, & resource.
- Benefits to PMs, stakeholders:
  - Helps for estimating, assignments, grooming.
- Benefits for all:
  - A common framework to understand why things take the time, resources that they do to be delivered.

Approaches

- **Informal**
  - Uses experience, history, rules of thumb.
  - Usually a good place to start when starting a test practice.
  - Not good for Safety or regulation heavy systems.
- **ISO 9126 System Quality Risk Analysis**
  - A good way to analyze risks. More formal & time consuming.
  - Not always good for speedy teams or uncooperative stakeholders.
  - It should be at least used informally as the group gets used to RBT.
- **Cost of Exposure**
  - Cost of risk happening vs cost of mitigation.
  - Useful when true probability can be calculated. Often used in finance domains.
  - Not good for speedy teams or where cost tracking is bad.
- **FMEA, Failure Mode & Effect Analysis**
  - A formalized drill down tactic to find where failures can happen before they do.
  - The informal version of this is always one of the first things an experienced test designer will consider, but formal methods involving the whole technical Team should be used only if the deliverables are high risk (medical, aerospace...), or where an area of risk is identified & the group agrees to kill it with fire.
  - Way too time consuming & resource hungry for agile team use unless it is considered a deliverable in itself
- **Hazard Analysis**
  - Similar to FMEA but starting with known failures in the past.
  - Again, as an informal process it's a mainline activity for teams during Bug analysis, and formal analysis should be reserved to high risk domains, or when the group agrees to go after a problem area.
  - Again, way too time consuming for agile team use unless it is considered a deliverable in itself.

How does it work?

Since there are always more tests then we can do, we need a neutral way to ranking the testing importance, *name the risks and rank them for effort*.

1. ID your risk categories - use these to estimate scores.
2. ID the work units where you will assign risk scores - features, epics, stories, tasks, releases, etc.
3. Estimate the risk scores for each work unit for each work cycle (release, sprint, Kanban item, ...) and use those to get the Risk Rank.
4. Plan the testing based on risk scores – It should also influence other work done outside of testing too.

Risk Categories

We map all deliverables to these categories.

It is best for the tester to ask their Dev & PM to help them detail these categories and rank the detailed factors for importance to the team.

- Functionality, Reliability, Availability, Usability, Efficiency, Maintainability, Portability

Other Factors to consider

- Bug frequency by feature, source area, developer, ... gives likelihood.
- Bug severity by feature, source area, developer, ... gives impact.
- Is the work adding to an existing feature or adding a new feature?
- 3rd party or uncontrollable dependencies - impact & likelihood from support forums, past experience.

The tester (& the developer hopefully) then maps the current work items in the sprint/cycle/release to the detailed categories and uses their relative importance to assign risk impact & likelihood scores to each work item

Who Decides Risk Scores?

The technical view

- Developers, architects/designers/lead devs, domain experts
- Tells us what could happen if the deliverable fails in production.
- Tells us how likely it is that it could fail.

The business view

- Program & Product managers, Product owners, stakeholders, sponsors.
- Tells us what could happen to the business with a fail.
- Tells us how likely it will be that a user/customer will see the fails.

We would love to get participation from all, but culturally this is frequently regarded by stakeholders as not their job, and by developers as a “waste of time”.

Actually Getting Contributors to Talk

- Should be done by the PM or the testers.
- Transparency & visibility of the process & its results are critical for the maximum benefit.
- Participate or be left out & noticed.
- Makes sure they know what you're doing & why.
- Make sure they know how to participate
  ...whenever they decide to do it.

Quality of Requirements

This is a major source of risk, but cannot be fixed using these techniques.

Instead this reflects back to the question of whether the user will accept what is delivered.

- The worse the requirement quality the more likely the deliverable will fail UAT, even after passing extensive testing.
- The effects of bad requirements are "outside" of the normal dev/testing cycle but can force rework if it is guessed incorrectly.
- Testers MUST ensure that the Devs & PMs agree on the interpretation of the requirements they have. Without agreement the testers could be blamed for any UAT failures.
- Testers cannot fix this. They can only report it and state the implied requirements they are following for their testing.

Example: Our Process - Informal

- We work with Dev lead & PM to verify our Risk Categories in each of our component areas.
- In Scrum it's easiest to tie to stories & Epics.
  - Bridges the divide between "Feature" (stakeholders' perspective) and Code changes (dev team's perspective)
- Determine scores in the Grooming / Planning phase.
- Use factors just discussed to come up with an Impact score & a Likelihood score for both business & technical dimensions.
- Use a simple score proportional to amount of risk. We use 1-3, others 1-5, 1 is low, 3 or 5 is very high.
  - Business impact & likelihood
  - Technical impact & likelihood

What's the Score?

- We need a single risk score for each unit of risk.
- Many variations oh how this is calculated. We do it the simple way.

1. Use the highest Impact and likelihood scores from either Tech or business.
2. Multiply them together, that's your risk score.
3. Use the table to determine the testing depth ranking.

The Table

Conservative Version

		Likelihood		
		Low (1)	Medium (2)	High (3)
Impact	Low (1)	Low Risk (1)	Some Risk (2)	Pretty Risky (3)
	Medium (2)	Some Risk (2)	Pretty Risky (4)	Riskiest (6)
	High (3)	Pretty Risky (3)	Riskiest (6)	Riskiest (9)

Test Ranking

- We use it as a release criteria
- We divide it into 4 Ranks
  - Riskiest (scores 6 or 9) - Deepest testing during sprints, includes full or area-wide use case based regression testing before the release's feature is considered DONE. Almost all the tests are automated to add to our regression inventory.
  - Pretty risky (scores 2 or 4), almost equally deep testing during sprints in the code areas affected by the changes to the code base but if coupling is not there then use case based regression testing is not needed until release bits are in the UAT phase. Test cases that exercise code areas likely to fail or that are dependent on other, high churn components get automated.
  - Some risk (score = 2), enough testing to cover the functionality changes and to ensure that affected code areas aren't broken. Use case based regression testing not needed until the UAT phase.
  - Low risk (score = 1), no sprint testing needed. Is covered in use case based regression testing in UAT.

What Else?

- To save time, beware of skewing the distribution of Risk Scores. If they all score high then you aren't going to save any time or resources. If you can't avoid it then maybe that's the message you want to tell the team.
- Automate smoke & regression testing and run in a CI build as much as economically useful.
- The quality of your technical comments and documentation is a major factor in risk exposure.
- How you analyze risk is as or more important than score keeping. Push the score hard at first so the Group begins talking in that framework.
- Small teams can drop scores or reuse risk profile templates for each sprint when there are only a handful of active work items and they "get the process".
- Consider adding risk analysis to story point determination & any other kinds of estimating.
- Always get agreement across the group on quality risk mitigation determining release readiness.
- Any time a developer does not know how they're going to do something when they start coding, you have a top rank risk.

References

There are hundreds of articles on these techniques & practices. Most can be mapped to agile processes. Do not let the formalities scare you off.

The quick summary:

- https://en.wikipedia.org/wiki/Risk-based_testing
  Good, quick summary of the basics.
- http://rbcs-us.com/site/assets/files/1151/quality-risk-analysis-article-_updated.pdf
  This outlines my favorite risk scoring approach. I've adopted it for use with my Agile test team.
- http://rbcs-us.com/site/assets/files/1163/risk-based-testing.pdf
  The PowerPoint version of Rex Black's RBT summary
- https://www.stickyminds.com/article/risk-based-testing-test-only-what-matters-0
  Another very quick summary
- http://www.methodsandtools.com/archive/archive.php?id=31
  This paper best covers the overall justification for RBT and factors to consider when coming up with risk categories and scoring rules. However the calculations suggested for ranking are unnecessarily complex and the examples poorly documented. I would skip that part and focus only on this article's coverage of risk areas.
- http://www.satisfice.com/articles/hrbt.pdf
  What kind of test talk would this be without a reference written by James Bach?
- http://rbcs-us.com/site/assets/files/1159/a-case-study-in-risk-based-testing.pdf
  Case Study using informal techniques.