RISK BASED TESTING

HENRIETTE HARMSE

A question that is asked regularly in testing circles is: "When should you stop testing?". Proponents of code coverage tools may suggest once some percentage of coverage is achieved you can stop testing. However, what do you do when the budget is severely contrained? An even more difficult situation to address is when a project starts out with a given budget, but during its lifetime the budget gets significantly reduced (i.e. due to economic downturn). This forces us to be able to provide the highest level of quality for the least amount of money. In this post I will explain how **risk based testing** can help to answer this question.

1. Motivation

A mistake that is often made is that the testing effort is distributed equally across the system – both critical and non-critical parts of the system are tested equally. This results in critical parts of the system not being tested sufficiently and non-critical parts being tested to the point of diminishing returns.

A further mistaken mindset of developers is that there is no such thing as a useless test. This may entice developers to add tests for the sake of adding tests. In actual fact every test (unit-, integration- or systems test) has to earn its place in the codebase. If there is no good motivation for a test, the test must be deleted from the codebase. Why is that? Because every test adds to volume of code that developers have to master and maintain to be productive members of the team. Tests add to the volume of code that has to be maintained. As such tests adds to the overall cost of maintenance of a system. **The most cost effective code to maintain is the code that has never been written**.

Risk based testing is a testing approach that helps developers and testers to prioritize the testing effort by assigning a relative risk to each software component.

2. Calculating the Relative Risk of a Software Component

The relative risk of a software component is based on the product of the relative risks assigned to the criteria deemed relevant for the project. Criteria for determining risk will be discussed in the next section. For each criteria a value is assigned between $1..X$, where 1 indicates that the relative risk of a criterion is insignificant and X indicates that the relative risk of the criterion is critical for the given software component.

Assuming we have software components (S) $1..m$, criteria (C) $1..n$ where we assign relative risks, $A_{i,1}..A_{i,n}$ for each criterion of a software component S_i, the relative risk R_i for each software component is given by:

$$R_1 = A_{1,1} \times ... \times A_{1,n}$$

Date: 22nd February 2020.

...

$$R_m = A_{m,1} \times ... \times A_{m,n}$$

Ordering $R_1 ... R_m$ in descending order of their values will cause the software components with the highest relative risk to be at the top of the list. Note that the calculated relative risk for a specific software component by itself is meaningless. A calculated risk of an component is only meaningful if it can be compared to the calculated risk of another component calculated in the same way (that is, using the same criteria and value for X). That is why we refer to the **relative** risk of a software component.

What value should you choose for X? In general you should choose X to be equal or greater than the number of criteria you will be using in calculating the risk. If many of the calculated relative risks R_i have the same value, it means you have to increase the value of X.

3. What Criteria should be Used?

In this section we give examples of criteria that can be used in the calculating the risk of each software component. However, these criteria merely serve as examples. You are free to choose whatever criteria are deemed of importance for your project.

Some general criteria that you may want to consider using are:

Frequency of use: How frequently is this software component used? The more often a software component is used, the higher is its potential for having an adverse effect when it fails.

Cost: What will be the cost of this software component failing? The cost can be monatary, but it does not have to be. It can be cost in terms of reputation, loss of clients, or whatever factor that can hurt the business. Here it may be a good idea to get the input of the project sponsor or project owner.

Complexity: What is the complexity of this software component? If this software component forms part of a green fields project, this criteria can express the complexity of this software component in comparison with other newly developed components. In the case where enhancements are made to an existing system, code analysis tools can be used to determine complexity of software components that need to be changed. For example, when an enhancement requires that 2 classes need to be changed substantially, the class with a higher cyclomatic complexity (the number of linear independent paths through the code) represents a higher risk.

Frequency of change: How often will the software component be changed? If a software component realizes the core value proposition of a business, it is likely that this component will constantly need to be enhanced as the business is trying to stay competitive. If there is little chance that a software component is going to be changed after it has been completed, the tests for that software component may be of limited value in the long term.

Number of bugs: For a system that is already in production, the number of bugs that are logged (that can be related back to specific software components) can help give an indication as to where testing effort should be focused.

TABLE 1. Relative Calculated Risk for the Online Share Trading System

Module	Usage	Cost	Complexity	Change	Bugs	Risk
User management	1	2	1	1	1	2
Share price notification	5	5	5	1	1	125
Account management	4	5	3	3	2	360
Mark-to-market	3	5	5	5	5	1875

4. An Example

As an example, let us assume we are working on a legacy online share trading system, naturally with zero automated tests. As with most legacy systems, a large number of users depend on the system. A steady stream of user requests for enhancements and bugs logged require the system to be frequently updated. These changes often introduce new bugs.

We further assume the system consists of the following modules:

- A user management, authentication and authorization module through which users and their permissions are managed. It is very seldom that any changes are required to this module.
- A share price notification engine of which the main purpose is to notify users in realtime of changes in share prices. The correct functioning of this engine is critical to the business since it enables users to buy/sell shares as and when needed. Any failure of the engine could mean that users cannot buy/sell shares as needed which could result in massive losses. However, since the initial problems with the engine have been resolved, bugs are seldomly logged for this module.
- The account management system keeps track of the funds of users as they trade, calculates daily interest earned or charged (a positive balance earns interest and for a negative balance interest is charged) and generate monthly statements of transactions.
- The mark-to-market process runs each day after the stock exchange has closed to calculate the gains/losses for each account. There are frequent changes to this process as quants are forever fine tuning the way commission is calculated on trades. Most of the bugs logged are related to this module.

To improve the situation, management wants the development team to add automated tests. How should we approach testing the system?

If we apply risk based testing as in seen in Table 1, it is clear that we need to focus our testing efforts on the mark-to-market process. If we now further assume the mark-to-market process consists of the following steps for each account:

- determine prices at which shares have been bought,
- determine current prices of shares held,
- calculate the profit/loss, and
- update the account balance.

A detailed plan of how to approach testing of the mark-to-market process can be drawn up (see Table 2), but instead of listing modules, it will list the software components that the mark-to-market process consists of. These can for example be classes, methods, functions, scripts etc.

Table 2. Relative Calculated Risk for the Mark-to-Market Process

Methods/Functions	Usage	Cost	Complexity	Change	Bugs	Risk
determineBuyPrices	2	5	2	1	3	60
determineCurrentPrices	4	1	5	1	1	20
calculateProfitLoss	2	5	5	5	4	1000
postAccountUpdate	2	5	1	1	1	10

5. Advantages/Disadvantages of Risk Based Testing

The advantages of risk based testing are:

- The highest risk items can be developed and tested first which reduces the overall risk on the project.
- If testing has to be watered down, a guideline exists for deciding what to test and what not to test.
- At any given time an indication can be given of the risk of the project by considering the highest risk software components that has not been tested.
- It is a valuable tool for communicating risk to management, project sponsors and project owners.

The disadvantages of risk based testing are:

- With a risk based testing approach many parts of the system will go untested. However, this will be an intentional decision rather than an accidental one.
- The relative risk value, that are assigned for each criterion of a software component, is likely to be subjective. That means that another person may assign a different relative risk value for a criterion of a software component. However, it is just as subjective as, for example, story points in agile methodology. It similarly makes sense to assign relative risk values as part of a team discussion.
- Some teams feel risk based testing adds to their documentation load. That is, they now have to draw up a complete risk profile before they can start testing. Do I always draw up risk profiles for all my projects? Honestly? No. What I will do is to have a discussion with the team regarding what areas of the system we need to test to death and which areas we can skimp on. If there is some disagreement, then I may write it out. Writing it out is very useful when the testing approach needs to be discussed with management, the project sponsor or the project owner.

6. Conclusion

The main aim of risk base testing is to ensure that the testing effort is focused where it will bring the most business value.