Practical Risk-Based Testing

Product RISk MAnagement: the PRISMA® method

Drs. Erik P.W.M. van Veenendaal CISA

Improve Quality Services BV, The Netherlands
www.improveqs.nl

May, 2009

Table of Content

1. Introduction

Often the activities prior to test execution are delayed. This means testing has to be done under severe pressure. It would be unthinkable to quit the job, to delay delivery or to test badly. The real answer is a differentiated test approach in order to do the best possible job with limited resources. Which part of the systems requires most attention? There is no unique answer, and decisions about what to test have to be risk-based. There is a relationship between the resources used in testing and the cost of finding defects after testing. There are possibilities for a stepwise release. The general approach is to test some important functions that hopefully can be released, while delaying others.

At system level, one probably has to test first what is most important in the application. This can be determined by looking at visibility of functions, at frequency of use and at the possible cost of failure. Secondly, one has to test where one may find most defects. This can be determined by identifying defect prone areas in the product. Project history gives some indication, and product measures give more. Using both, one finds a list of areas to test more and those to test less. After test execution has started and one has found some defects, one may analyse these in order to focus testing even more and tune the test approach. The idea is that defects clump together in defect prone areas, and that defects are a symptom of particular trouble the developers had. Thus, a defect leads to the conclusion that there are more defects nearby, and that there are more defects of the same kind. Thus, during the latter part of test execution, one should focus on areas where defects have been found, and one should generate more tests aimed at the type of defect detected before.

2. Product risk management

This paper describes a method for identifying the areas that are most important to test. The items that have the highest level of risk. The Product RISk MAnagement (PRISMA®)[1] method has been develop in practice and is being applied in many projects and companies in a wide range of industries. The PRISMA method supports the test manager in doing risk-based testing, especially for the risk identification and analysis in close co-operation with stakeholders.

The PRIMSA method can be used at every level of testing, e.g. component, integration, system, and/or acceptance testing. It can be applied at both organizational and project level. On an organizational level the method can be used to address properties that are common for most projects in the organisation or for a development program. The result can be documented as part of the overall test strategy, which can be considered as a blue print to be applied by the projects. On a project level, the product risk analysis is input for the projects' test approach documented in a test plan. Note that PRISMA is a method for product risk management, not for project risks. In practice it is often the combination of product risks and project risks that determines the detailed test approach.

Product risk analysis should be used to determine the appropriate test approach and to select test design techniques in such a way that the items with the highest risks are tested first and more intensively than the areas with low risk. The output of the risk analysis may even also

[1] A tool is available to support the PRISMA method and product risk management process.

influence the development approach, e.g. developing high risk areas as part of an initial increment to allow early testing or assigning the most experienced engineers to higher risk areas.

Risk tracking should be done throughout the project. This can be done by periodically repeating (part of) the risk analysis and validating the initial risk analysis. Also test monitoring and control, including reporting, should be organised around these risks. Too often initially risks are identified and never looked at again further on in the project.

3. Good enough testing

At a closer look risk based testing, and therefore also PRISMA, are highly related to the concept of good enough testing. James Bach introduced the idea in 1997 (Bach, 1997) and it has caused some division in the testing community, at least. On one side of the fence, some experts think it's a cop-out; a compromise too far; it's too simplistic to be useful; it promotes shoddy work; and so on. On the other hand, its supporters promote it as something that reflects what we do in real life where something less than a "perfect" solution is inevitable. The good enough approach is helpful to understanding the risk-based test approach. It is a good framework for the (release) decision-making in projects where risks are being taken. Were you ever asked as a tester, "is the system good enough to ship?" When the time comes to make the big decision, how could you answer that question? If you say, "well, it's just not ready", the project manager just thinks, "testers always say that, they're never happy" and you are dismissed as a pessimist. Suppose you say, "Well, it looks ready to me", will your project manager put a piece of paper under your nose, asking you to sign it? If you sign, are you taking someone else's responsibility? So, what is "good enough?" and how does it help with the risk based testing?

"Good enough" is a reaction to the formalism in testing theory. It's not reasonable to aim at zero-defects (at least in software), so why do you pretend to yourself and pretend to the users and customers that you're aiming at perfection? The zero-defect attitude just doesn't help. Your customers and users live in the real world, why don't we? Compromise is inevitable, you always know it's coming, and the challenge ahead is to make a decision based on imperfect information. As a tester, don't get upset if your estimates are cut down or your test execution phase is squeezed. Guilt and fear should not be inevitable just because a project is constrained for budget and resources and has more than zero defects remaining.

The definition of "good enough" in the context of a system (or increment / enhancement) to be released is:

1. It has sufficient benefits;
2. It has no critical problems;
3. Its benefits sufficiently outweigh its non-critical problems;
4. In the present situation, and all things considered, delaying its release to improve it further would cause more harm than good.

This definition means that there is already enough of this product working (this system, increment or enhancement) for us to take it into production, use it, get value, and get the benefit. "It has no critical problems" means that there are no severe faults that make it unusable or unacceptable. At this moment in time, with all things considered, if we invest more time or money trying to perfect it that will probably cost us more than shipping early with the known problems. This framework allows us to release an imperfect product on time because the benefits may be worth it. So how does risk based testing fit into this good enough idea?

Firstly, have sufficient benefits been delivered? The tests that we execute must at least demonstrate that the features providing the benefits are delivered completely, so we must have evidence of this. Secondly, are there any critical problems? The incident reports that record failures in software provide the evidence of at least the critical problems (and as many other problems as possible). There should be no critical problems for it to be good enough. Thirdly, is our testing good enough to support this decision? Have we provided sufficient evidence to say these risks are addressed and those benefits are available for release? In essence these questions are all about balancing; spending the resources on testing to deliver good enough quality and acceptable level of risk.

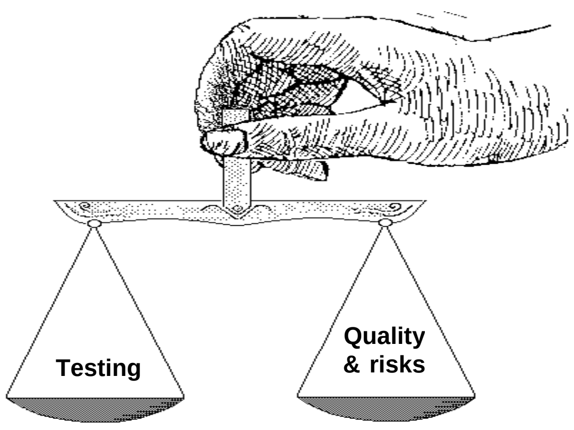


Figure 1: Balancing testing with quality & risks

Who decides?
It is not a tester's responsibility to decide whether the product is good enough. An analogy that might help here is to view the tester as an expert witness in a court of law. The main players in this familiar scene are:

- the accused (the system under test)
- the judge (project manager)
- the jury (the stakeholders)
- expert witness (the tester).

In our simple analogy, we will disregard the lawyers' role. We will assume that the prosecution and defence are equally good at extracting evidence from witnesses and challenging "facts" and arguments. We will focus on the expert witness role; these are people who are brought into a court of law to present and explain complex evidence in a form for laymen (the jury) to understand. The expert witness must be objective and detached. If asked whether the evidence points to guilt or innocence, the expert explains what inferences could be made based on the evidence, but refuses to judge. In the same way, the tester might simply state that based on evidence "these features work, these features do not work, these risks have been addressed, these risks remain". It is for others to judge whether this makes a system acceptable.

The tester is there to provide information for the stakeholders to make a decision. After all, testers do not create software or software faults; testers do not take the risks of accepting a system into production. Testers present to their management and peers an informed and independent point of view. When asked to judge whether a product is good enough, the tester might say that on the evidence obtained, these benefits are available, but these risks still exist. However, if as a tester you are actually asked to make the decision, what should you do? The answer is that you must help the stakeholders make the decision, but not make it for them. The risks, those problems that we thought say 6 months ago could occur, and which in your opinion would make the system unacceptable, might still exist. If those were agreed with the stakeholders at that time, the system cannot now be acceptable, unless they relax their perceptions of the risk.

The judgement on outstanding risks must be as follows:

- there is enough test evidence now to judge that certain risks have been addressed.
- there is evidence that some features do not work (the risk has materialized).
- some risks (doubts) remain because of lack of evidence (tests have not been run, or no tests are planned).

This might seem less than ideal as a judgement, but is preferable to the unrealistic, ideal world acceptance criteria discussed earlier. You may still be forced to give an opinion on the readiness of a system but we believe that by taking this principled position as an expert witness (and taking it as early in the project as possible) you might raise your credibility with management. Management might then give you the right responsibilities on future projects.

The Classic Squeeze on Testing
The well-known squeeze on testing occurs when the developers deliver late into test but the go-live deadline remains fixed. The exit criteria might be used to determine what happens next, but too often it is obvious that these criteria cannot be met in time. The pressure to release might be so great that the exit criteria are set aside. There may be some attempt to downgrade the severe bugs perhaps. Exit criteria are an uncomfortable reminder of the idealistic attitude in the early stages of the project, but do not make the decision any easier. The risks that were apparent at the start of testing have been visible throughout. When testing is squeezed, some risks may have been addressed, some benefits may be available, testers may have revealed new risks to be addressed, but the outstanding risk is apparent to all involved. If the decision to release is actually made, the stakeholders have explicitly chosen to accept a product before they have evidence that all risks are addressed, and that all benefits are available. This should not simply be considered as a tester’s problem. The stakeholders have judged that they had enough information to make that decision.

In fact, information for making the release decision becomes available from the first day of test execution onwards; it’s just that the balance of testing evidence versus outstanding risks weighs heavily against release. A positive (though perhaps surprising) principle for risk-based testers is therefore that the time available for test execution has no bearing on your ability to do ‘good testing’.

4. The testing challenge

The scenario is as follows: You are the test manager. You have to make a plan and a budget for testing. Your ideas were, as far as you knew, reasonable and well founded. When testing time approaches you might find that the product is not ready, some of your testers are not available, or the budget is just cut. You can argue against these cuts or argue for more time or other resources, but that doesn't always help. You have to do what you can with a smaller budget and time frame. You have to test the product as well as possible, and you have to make sure it works reasonably well after release. How to survive? Doing bad testing will make you the scapegoat for lack of quality. Doing reasonable testing will make you the scapegoat for a late release.

You need some creative solution: you have to change the game. Inform your management about the impossible task you have, in such a way that they understand. But it's also important to present alternatives. They need to get a product out of the door, but they also need to understand the risk.

One strategy is to find the right quality level. Not all products need to be free of defects. Not every function needs to work. Sometimes, you have options related to lowering product quality. This means you can cut down testing in less important areas. Another strategy is priority: Test should find the most important defects first. Most important means often "in the most important functions". These functions can be found by analysing how every function supports the objectives of the system, and checking which functions are critical and which are not. You can also test more where you expect more defects. Finding the worst areas in the product soon and testing them more will give you more defects. If you find many serious problems, management will often be motivated to give you more time and resources. In practice it is often about a combination of most important (discussed in section 5) and worst (discussed in section 6) area priority. Risk based testing should take care that whenever the team has to stop testing, they have done the best testing is the time available.

5. The most important parts of the product

Testing is always a sample. You can never test everything, and you can always find more to test. Thus you will always need to make decisions about what to test and what not to test, what to test more or what to test less. The general goal is to find the worst defects first, and to find as many such defects as possible. A way to ensure this is to find the most important functional areas and product properties. Finding as many defects as possible can be improved by testing more in the bad areas of the product. This means you need to know where to expect more defects, which will be explained in the next section.

You need to know the most important areas of the product. In this section, a way to prioritise this is described. The ideas presented here are not the only valid ones. In every product, there may be other factors playing a role, but the factors given here have been valuable in many projects. Important areas can either be functions or functional groups, or quality attributes such as performance, reliability, security etc. In this paper we will use the generic term test items for this.

Major factors to look when determining the importance of test items include:

Critical areas (cost and consequences of failure)
You have to analyse the use of the software within its overall environment, analyse the ways the software may fail. Find the possible consequences of such failure modes, or at least the worst ones. Take into account redundancy, backup facilities and possible manual checks of

output by users, operators or analysts. A product that is directly coupled to a process it controls is more critical than a product whose output is manually reviewed before use. If a product controls a process, this process itself should be analysed.

A possible hierarchy is the following:

- *A failure would be catastrophic*
  The problem would cause the system to stop, maybe even take down things in the environment (stop the entire workflow or business or product). Such failures may deal with large financial losses or even damage to human life.
- *A failure would be damaging*
  The program may not stop, but data may be lost or corrupted, or functionality may be lost until the program or computer is restarted.
- *A failure would be hindering*
  The user is forced to work around, to execute more difficult actions for reaching the same results.
- *A failure would be annoying*
  The problem does not affect functionality, but rather makes the product less appealing to the user or customer.

Of course damage will mean very different things depending on the product, for some products it is related to (human) safety and for some 'only' to financial damage. Another way of looking at user importance is to take the view of marketing. What are the (unique) selling points of this new product for our customers?

Visible areas
The visible areas are areas where many users will experience a failure, if something goes wrong. Users do not only include the operators sitting at a terminal, but also final users looking at reports, invoices, or the like, or being dependent on the service delivered by the product which includes the software. A factor to take into account under this heading is also the tolerance of the users to such problems. It relates to the importance of different functions or quality attributes, see above. Software intended for untrained or naive users, especially software intended for use by the general public, needs careful attention to the user interface. Robustness will also be a major concern. Software which directly interacts with hardware, industrial processes, networks etc. will be vulnerable to external effects like hardware failure, noisy data, timing problems etc. These kinds of products need thorough validation, verification and re-testing in case of environment changes. Regarding visibility often a distinction is made between external visibility (outside the organisational boundaries) and internal visibility where 'only' our own users experience the problem.

Most used areas
Some functions may be used every day, other functions only a few times. Some functions may be used by many, some by only a few users. Give priority to the functions used often and heavily. The number of transactions per day may be an idea in helping to find priorities.

A way to set priorities is to skip the testing of functional areas, which will only be used once per quarter, half-year or year. Such functionality may be tested after the release, before its first use. Sometimes this analysis is not so obvious. In process control systems, for example, certain functionality may be invisible to the outside. It may be helpful to analyse the design of the complete system.

A possible hierarchy is outlined here:

- *Unavoidable*
  An area of the product that most users will come in contact with during an average usage session (e.g. start-ups, printing, saving).
- *Frequent*
  An area of the product that most users will come in contact with eventually, but maybe not during every session.
- *Occasional*
  An area of the product that an average user may never visit, but that deals with functions a more professional or experienced user will need occasionally.
- *Rare*
  An area of the product which most users never will visit and which is visited only if users do very uncommon steps of action. Critical failures, however, are still of interest.

An alternative method for picking important requirements is described in (Karlsson *et al*, 1997).

6. The worst areas of the product

The worst areas are the ones having most defects. The task is to predict where most defects are located. This is done by analysing probable defect generators. In this section, some of the most important defect generators and symptoms for defect prone areas are presented. There exist many more, and often local factors must be included in addition to the ones mentioned here. This applies to the factors for identifying the most important parts and for the factors identifying the worst areas.

Complex areas
Complexity is maybe the most important defect generator. More than 200 different complexity measures exist, and research into the relation between complexity and defect frequency has been going on for more than 20 years. However, no predictive measures have until now been generally validated. Still, most complexity measures may indicate problematic areas. Examples include number of variables used, complex logic and complex control structure. This means one may do several complexity analyses, based on different aspects of complexity and find different areas of the product that might have problems.

Changed areas
Change is an important defect generator (Khoshgoftaar *et al*, 1998). One reason is that changes are subjectively understood as easy, and thus not analysed thoroughly for their impact. Another reason is that changes are done under time pressure and analysis is not completely done. The results are side effects. In general, a change log is available as part of the configuration management system (if something like that exists). You may sort the changes by functional area or otherwise, and find the areas which have had exceptional amount of changes. These may either have been badly designed from the start, or have become badly designed after the original design has been destroyed by the many changes. Many changes are also a symptom of poor analysis. Thus, heavily changed areas may not correspond to user expectations.

New technology and methods
Programmers using new tools, methods and technology experience a learning curve. In the beginning, they may generate many more faults than later. Tools include CASE (Computer Aided Software Engineering) tools, which may be new in the company, or new in the market and unstable. Another issue is the programming language, which may be new to the programmers. Any new tool or technique may give trouble.

Another factor to consider is the maturity of methods and models. Maturity means the strength of the theoretical basis or the empirical evidence. If software uses established methods, like finite state machines, grammars, relational data models, and the problem to be solved may be expressed suitably by such models, the software can be expected to be quite reliable. On the other hand, if methods or models of a new and unproven kind, or near the state of the art are used, the software may be more unreliable.

Most software cost models include factors accommodating the experience of programmers with the methods, tools and technology. This is as important in test planning, as it is in cost estimation.

People involved
The idea here is the thousand monkey’s syndrome. The more people that are involved in a task, the larger is the overhead for communication and the greater the chance that things will go wrong. A small group of highly skilled staff is much more productive than a large group with average qualifications. In the COCOMO (Boehm, 1981) software cost model, this is the largest factor after software size. Much of its impact can be explained from effort going into detecting and fixing defects. Areas where relatively many and less qualified people have been employed may be identified for better testing. It is important in this context to define what qualified means, e.g. is it related to the programming language, domain knowledge, development process, working experience in general, etc.

Care should be taken in that analysis: some companies (Jørgensen, 1984) employ their best people in more complex areas, and less qualified people in easy areas. Then, defect density may not reflect the number of people or their qualification. A typical case is the program developed by lots of hired-in consultants without thorough follow-up. They may work in very different ways.

Time pressure
Time pressure leads to people making short cuts. People concentrate on getting the problem solved, and they often try to skip quality control activities, thinking optimistically that everything will go fine. Only in mature organizations this optimism is controlled.

Time pressure may also lead to overtime work. It is well known, however, that people loose concentration after prolonged periods of work. Together with short cuts in applying reviews and inspections, this may lead to extreme levels of defect density. Data about time pressure during development can best be found by studying time lists, or by interviewing management or programmers.

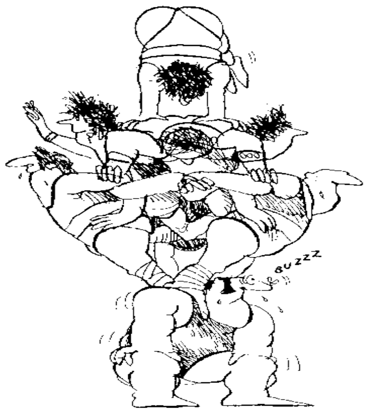


Figure 2: Under pressure things only get worse

Optimisation
The COCOMO cost model mentions shortage of machine time and memory as one of its cost drivers. The problem is that optimisation needs extra design effort, or that it may be done by using less robust design methods. Additional design effort may take resources away from defect removal activities, and less robust design methods may generate more defects.

Defect history
Defect repair leads to changes, which lead to new defects, and therefore defect prone areas tend to persist. Experience shows that defect prone areas in a delivered system can be traced back to defect prone areas in reviews and unit and subsystem testing. Evidence in studies (Khoshgoftaar *et al*, 1998) and (Levendel, 1991) shows that modules that had faults in the past are likely to have faults in the future. If defect statistics from design and code reviews, and unit and subsystem testing exist, then priorities can be chosen for later test phases.

Geographical spread
If people working together on a project have a certain distance between each other, communication will be worse. This is true even on a local level. Here are some ideas that haven proven to be valuable in assessing if geography may have a detrimental effect on a project:
- people having their offices in different floors of the same building will not communicate as much as people on the same floor.
- people sitting more than 25 meters apart may not communicate enough.
- a common area in the workspace, such as a common printer or coffee machine improves communication.
- people sitting in different buildings do not communicate as much as people in the same building; people sitting in different labs communicate less than people in the same lab do.
- people from different countries may have communication difficulties, both culturally and with the language.
- if people reside in different time zones, communication will be more difficult.

In principle, geographical spread is not dangerous. The danger arises if people with a large distance *have to* communicate, for example, if they work on a common part of the system. You have to check areas where the software structure requires the need for good communication between people, but where these people have geography against them.

Other factors that can be considered include:
- **new development vs. re-use**: areas that are totally newly developed (from scratch) are likely to contain more defects than those that are (largely) re-used.

- **interfacing**: practice has shown that many defects are related to interfaces between components, often due to communication problems. Components with more interfaces are therefore often more defect-prone. A distinction in this context is often made between internal and external interfaces.
- **size**: sometime people loose overview if components get too large. Therefore (too) large components may be more defect prone than average sized components.

What to do if you do not know anything about the project, also the defect generators cannot yet be identified? In that case, start with running an exploratory test. A first rough test should find defect prone areas, in the next test you can then concentrate on these areas. The first test should cover the whole system, but be very shallow. It should only cover typical business scenarios and a few important failure situations, but cover the entire system. You can then determine in which areas the most problems were revealed, and give priority to these areas in the next round of testing. The next round will then do deep and thorough testing of prioritised areas. This two-phase approach can always be applied, in addition to the planning and prioritising done before testing.

7. The PRISMA process

In this section the process is described that can be followed when performing a product risk assessment using the PRISMA method.

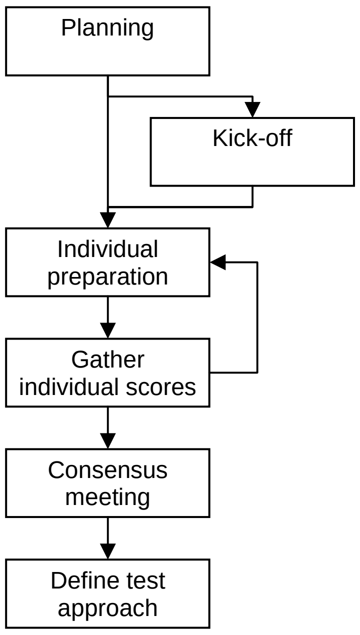


Figure 3: PRISMA process overview

The central theme is the PRISMA process is the creation of the so-called product risk matrix. Using the factors, as explained in the previous sections, for each item to be tested (risk item) the impact of defects and the likelihood of defects is determined. By assigning numeric values to both, a risk item can be positioned in the product risk matrix. The standard risk matrix is divided in four areas (quadrants I, II, III and IV), each representing a different risk level and type. A different risk level / type should lead to a different test approach documented in a test plan.

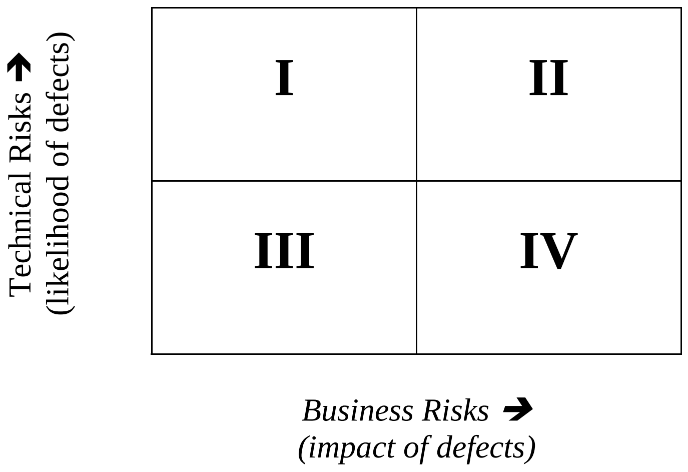


Figure 4: Product risk matrix

To support the PRISMA process a supporting software tool has been developed by TASS software professionals in cooperation with Improve Quality Services. Output examples of this tool are used throughout the remainder of this paper (e.g. figures 5, 6 and 8).

7.1 Planning

Gathering input documents

As always a structured planning phase is the key to success. During the planning the input documents (often the same as the test basis) are determined and collected. Of course the relevant input documents depend largely on the test level on which the risk analysis is performed. One needs to check that the input documents are at the required quality level and that they contain the items (referred to as test items) that can be used in this process. The inputs don’t have to be ‘final’ or ‘accepted’, but sufficiently stable to use them for product risk analysis. Gaps (e.g. in requirements) should be identified and reported to the document owner. If the input documents are not available at the correct level, one needs to consider how and if to continue with the product risk analysis process. Ideally the documents to be used are a requirements document (for testing at system level) or an architectural document (for development testing).

Identifying risk items

The items that are used for the risk assessment are identified based on the input documents: the risk items. If assumptions are made (e.g. related to ambiguousness in input documents), these should be documented. The most likely scenario can be determined by interviewing the document owner and/or stakeholders. As a rule of thumb there should not be more than approximately 30 – 35 risk items to keep the process workable. This often means the items (e.g. requirements) as stated in the input document need to be grouped into logical units. The identified risk items will be structured following a hierarchy. It is most often not useful to consider each and every elementary requirement as a separate risk item. Depending on the test strategy, the project may decide to do a separate, more detailed, product risk analysis to assess these elementary requirements per component or subsystem at a later stage using the higher level risk analysis result as an input.

The identified risk item list should be uniquely identified and understandable to the participants. The reference identification can be a number, but it may also contain a code or abbreviation that is meaningful in the project. Per risk item, if possible, a link should be provided to the relevant parts of the input documentation. A description of the risk item can be added as a one-liner. For the participants, this description should give a clear idea which risk item they have to assess.

Determine impact and likelihood factors
For product risk analysis two ingredients are relevant: the likelihood that a failure occurs and the impact when this happens. In sections 5 and 6 these two ingredients are discussed and several factors that influence the impact and/or likelihood that are recommended to be used are presented. The test manager determines the factors that the project will use to assess the risk items in terms of likelihood and impact. Ideally an initial list of factors is already determined at organisational level, e.g. in the test strategy. The project may of course have good reasons to deviate from the standard set of risk factors.

In addition one will have to select a value set for each factor. Preferably these values are not presented as numbers, but with meaningful descriptions, e.g. ‘less than 5 interfaces’, ‘more than 10 interfaces’ etc. If possible, the values should have an objective and measurable meaning (e.g. ‘< 1 KLOC’ instead of ‘medium size’). However, in practice most often numeric value sets are used such as 1 to 3, 1 to 5 or 0, 1, 3, 5, 9. The first one is probably the easiest one to use and related to the low (1), medium (2) and high (3) concept. Purely from a testing perspective the last one is preferred since it more clearly identifies the high risks, since the 9 is highly distinguishing compared to the other values within the set. Preferably the value set with interpretations for each factor should already have been determined at a higher level than at project level.

Define a weight for each factor
It is also possible to use weights where one factor is considered more important than another factor, e.g. one factor could have 2 times the ‘worth’ of another factor. Again weights for the factors are preferably determined in the test strategy, but can be tailored for project specific purposes. The general method is to assign weights, and to calculate a weighted sum for every area of the system. Focus your testing on the areas where the result is highest! For every factor chosen, assign a relative weight. You can do this in very elaborate ways, but this will take a lot of time. Most often, three weights will suffice. Values may be 1, 2, and 3. (1 for “factor is not very important”, 2 for “factor has normal influence”, 3 for “factor has strong influence”). Once having gathered historical project data one can start fine tuning the weights.

Select stakeholders
The stakeholders that will be involved in the product risk analysis are identified and selected. Typically different roles from the business and from within the project will be selected. Examples are project manager, developers, software architect, marketing, end user, business manager and application engineer. This is an important activity, which is explained by the statement “a stakeholders that is forgotten implies the risks that he/she own are most likely not identified”.

Theoretically every stakeholder could be asked to assign values to every factor. In practice it is much more useful to assign only those factors that are relevant to the stakeholder, i.e. related to his role. Typically the factors for impact should be assigned to business representatives and factors for likelihood to technical experts, e.g. the software architect, senior engineer. For obvious psychological reasons the method prescribes not to use weight factors for stakeholder, i.e. each stakeholder is considered equally important. It is also highly recommended to assign each factor to at least two stakeholders.

The test manager who is responsible for assigning the roles has to make a good balance in:

- choosing the appropriate persons for the roles depending on the test level
- choosing “technical” roles to fill in the likelihood and “business” roles to fill in impact
- involving sufficient knowledge areas, both for impact and likelihood parts

Scoring rules
Finally the rules are set that apply to the scoring process. One of the common pitfalls of the risk analysis is that the results tend to cluster, i.e. the result is a two-dimensional matrix with all risk items close to each other. To prevent this in an efficient way, the stakeholders should be enforced to score the factors on a full range of the values. Rules will support this process. Examples are ‘the full range of values must be used’, ‘all factors shall be scored, no blanks allowed’ and ‘homogeneous distribution of values assigned to a factor’.

7.2 Kick-off

Optionally, a kick-off meeting can be organized in which the test manager explains to all stakeholders their role in the process. Although optionally, the kick-off meeting is highly recommended. After the kick-off meeting the process and expected actions should be clear to all participants. The kick-off meeting can also be used to explain the list of risk items, factors and to make clear to which factors they have to assign a value.

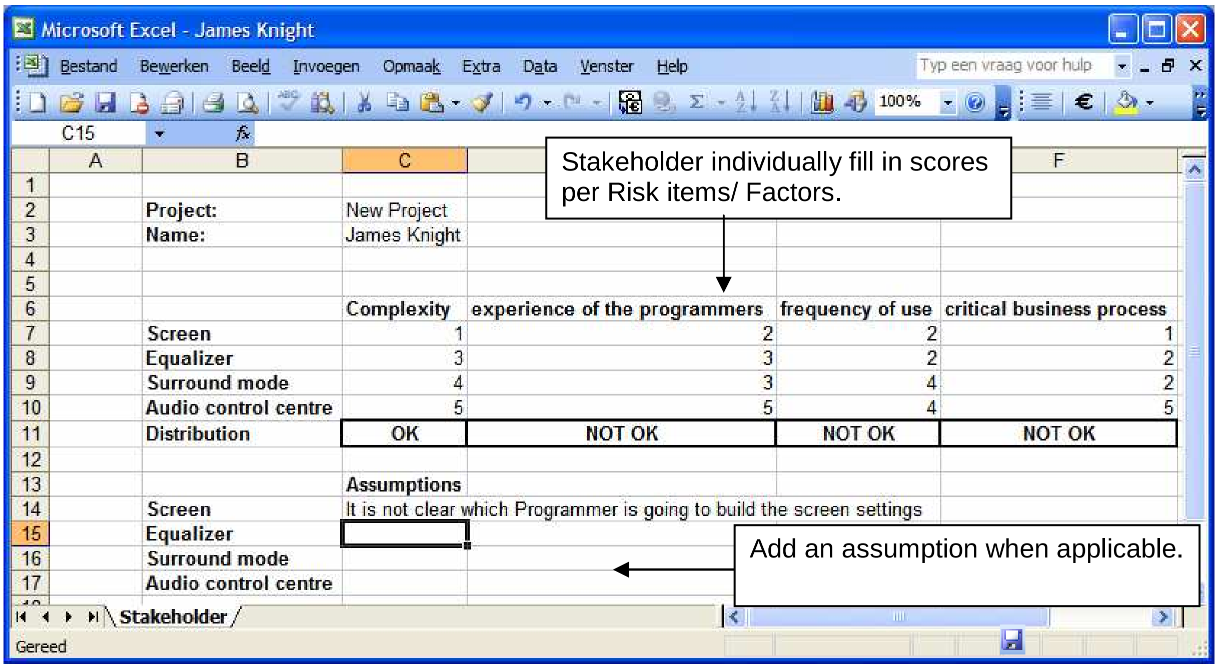
The kick-off phase of the risk analysis is to ensure that not only the process, the concept risk based testing and the risk matrix, but also the purpose of the activities are clear. A clear explanation and common view will contribute to a better deployment and motivation. Discussion about the usefulness of the process and expected benefits should take place here, not later during the process. Items to discuss here are e.g. how to perform the individual preparation, explanation of the tools to be used, and the applicable deadlines. The test manager also provides an overview of the remainder of the process. It should be made clear to the stakeholders what to expect at the consensus meeting organized at a later stage and what will happen at the end of the process. The test manager explains to the stakeholders what their role in this process is and how their contribution influences the test approach for the project.

The risk items and factors are explained in detail, as the stakeholders will be requested to score them. The exact meaning of the risk items, the factors, the value set(s) and assumptions must be made very clear to them. To obtain reliable results there must be a common understanding of all properties of the risk assessment.

At the end of the kick-off meeting, commitment from the stakeholders is needed, to ensure they will participate in a meaningful manner.

7.3 Individual Preparation

During the individual preparation values are assigned to the factors per risk item by the participants. The participants score by selecting (the description of) the value that fits best the perceived risk for the corresponding factor regarding a risk item. This step can be done manually, but is often supported by providing the participants with Excel sheets that even support automatic checking against the scoring rules.



Microsoft Excel - James Knight

	A	B	C	D	E	F
1						
2		Project:	New Project			
3		Name:	James Knight			
4						
5						
6			Complexity	experience of the programmers	frequency of use	critical business process
7		Screen	1	2	2	1
8		Equalizer	3	3	2	2
9		Surround mode	4	3	4	2
10		Audio control centre	5	5	4	5
11		Distribution	OK	NOT OK	NOT OK	NOT OK
12						
13			Assumptions			
14		Screen	It is not clear which Programmer is going to build the screen settings			
15		Equalizer				
16		Surround mode				
17		Audio control centre				

Stakeholder individually fill in scores per Risk items/ Factors.
Add an assumption when applicable.
Stakeholder
Gereed

Figure 5: Example of participants' score sheet (from PRISMA tool)

The values filled in by the stakeholder are based on perceived risks: the stakeholder expects that something could go wrong (likelihood of defects) or is important for the business (impact of defects). Perceived risks are often based on personal assumptions of the stakeholder. These assumptions have also to be documented. Later, in the consensus meeting, these assumptions are highly useful to compare the scores, explain unexpected results and outliers, e.g. 'Why did we score this way?', 'Did we make different assumptions?'.

The values are checked against the rules that were pre-determined by the test manager. For example: are the selected values sufficiently distributed, and are all assigned factors to the stakeholder scored upon completion. It is important to get an (as much as possible) even distribution between the various possible values when scoring for a certain factor. This is essential for a useful product risk analysis. Assigning values is not about absolute values but about relative values, e.g. which is the most complex item, what item is most often used. By using both high and low values during the scoring process, clearer testing priorities can be set at a later stage thanks to a more differentiated scoring.

Example of scoring:

Good Practice	
Risk item	Complexity
001	Low
002	High
003	Medium
004	Low

Bad Practice	
Risk item	Complexity
001	High
002	High
003	Medium
004	High

Table 1: Examples of good and bad scoring practices

During this phase, the test manager has a supporting role to the (first time) participants, e.g. re-explaining in detail the process, the rules and possibly the factors.

7.4 Gather individual scores

Entry check
During the gathering of individual scores the test manager first checks whether scoring has been done correctly. If there are still violations to the rules, the test manager should discuss this with the participant. Maybe the meaning of a factor or value should be clarified or some additional assumptions have to be made and documented. The test manager should also check if the participant documented his assumptions. If needed, the applicable participant has to (partly) repeat the individual preparation. A check needs to be done by the test manager whether at least two scores for each factor have been received. If a rule violation can’t be resolved by stakeholder and test manager, e.g. when they agree that ‘even distribution’ is not applicable for a certain factor, this should be discussed in the consensus meeting.

When the deadline is approaching, the test manager should remind the stakeholders to submit their score and return them in time. Stakeholders that didn’t return their scores on time shall be approached individually. Before the next phase of the process is entered, there should be at least a representative response from each group of stakeholders or roles in the project.

Processing individual scores
The test manager now processes and analyses the individual scores by calculating the average value. He also prepares a list of issues to be discussed in the consensus meeting. For each risk item the likelihood and impact is determined. Per risk item the scores of the factors determining likelihood are added up and separately the scores of the impact factors are added up. Each risk item can now be positioned in the so-called risk matrix.

Candidates for the issue list to be discussed in the consensus meeting are all outstanding violations of rules:

- when because test manager together with stakeholder have decided to escalate a rule violation
- when the total result of all assessment forms has lead to unresolved risk items. A risk item is qualified as unresolved when the distribution of all assigned values for a factor exceeds a pre-determined threshold, e.g. a stakeholder assign the highest value and another stakeholder assigns the lowest value for the same factor regarding a certain risk item.
- also risk items that are positioned too close to the centre of the risk-matrix where all quadrants come together should also be discussed, e.g. within the circle of the risk matrix example in figure 6.

The threshold value and the other rules are determined in the project rule set by the test manager during the planning phase.

7.5 Consensus meeting

The consensus meeting starts by the test manager explaining the objectives of the meeting. At the end of this meeting a common understanding should be achieved on the (perceived) product risks. The final result should be a risk matrix committed by the stakeholders and adhering to the rule set. A consensus on all scores is not necessarily needed and sometimes even impossible. After all each participant has its' own interest and views regarding the importance of specific risk items depending on his background and role.

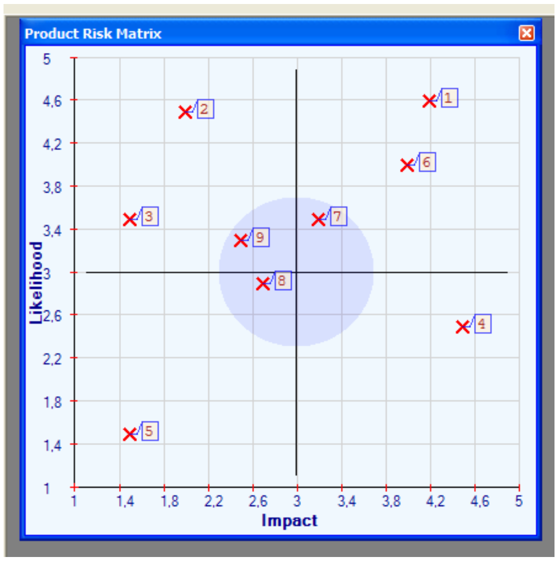


Figure 6: Example of a risk matrix (from the PRISMA tool)

During the meeting the items from the issue list are discussed between stakeholders and test manager with the purpose to reach consensus and understanding each others arguments. The discussion takes place per risk item per factor using the documented assumptions. Often different understanding of the requirements is a source of different scoring. This should lead to a change request on the requirements since these are then obviously not unambiguous.

At the end of the discussion final scores are determined and the resulting risk matrix is presented (see figure 6). This resulting matrix should always be validated with the stakeholders: "Is the matrix as expected or are there any surprising results?". If results are not according to the expectations of the stakeholders they should be (re-)discussed. Common sense should always be a part of any method. Using the method too strictly can be dangerous.

At the end of the meeting, the test manager summarizes the results and checks if the intended objectives are met. If needed, a follow-up meeting could be organised, or a specific discussion meeting in a smaller group, e.g. when the requirements team has dealt with questions that were raised.

Larger projects: more risk matrices
Within one project several risk matrices can be created. Examples include:

- a matrix for an acceptance test and a risk matrices for the supplier tests
- a matrix for a master test plan level and a matrix for specific test phases.

Whenever several risk matrices are created, they should be consistent. It is often possible to re-use parts of a matrix from a higher-level test plan for a lower-level test plan. For example, when selecting stakeholders for a risk matrix for a lower-level test plan, it can be decided not to select any stakeholders to assess the business risks (impact factors). Instead the business risks are copied from the higher-level test plan. Consistent means that risk items, which are present in more than one matrix, have comparable scores. When a risk item in the risk matrix at the higher-level test plan is split into several items at a lower-level risk matrix, the average risk score of these items must to be comparable to the score of the risk item in the higher level product risk matrix.

7.6 Define a differentiated test approach

Based on the test items' location in the risk matrix, all test items are prioritized. The result is ordering of all test items, the most important item first. In addition to prioritization, a differentiated test approach for the test items needs to be defined, based on their position in the risk matrix. The test approach usually has two major aspects: the test depth used and the priorities for testing. Test depth can be varied by using different test design techniques, e.g. using the decision table technique on high-risk test items and using 'only' equivalence partitioning for low risk test items. The problem with varying test depth based on the application of test design techniques is that not all (test) projects are mature enough to use a set of test design techniques. Many test projects still directly write test cases based on requirements and do not practice test design.

In addition to using different test design techniques there are alternatives to define a differentiated approach based on the resulting risk matrix. Practices to consider which can be applied for defining a differentiated approach include static testing, review of test designs, re-testing, regression testing, level of independence and exit criteria such as a statement coverage target. Also high-risk items can be tested by the most experienced engineers, another way to mitigate the risk.

Let's consider the testing practices mentioned more in detail and how they can be used to define a differentiated test approach:

- *Static testing*
  Based on the identified risks one can choose to do more review, e.g. inspection, on those areas that are considered high risk.
- *Reviewing of test designs*
  For high-risk areas the test designs (or test cases) can be reviewed with stakeholders or other testers.
- *Re-testing*
  With re-testing, also called confirmation testing, one can decide to re-run the full test procedure or just the step that failed and re-test the defect solved in isolation.
- *Regression testing*
  Of course the outcome of the risk assessment can also drive the regression test, where high-risk areas should be most intensively covered in the regression test set.
- *Level of independence*
  It is possible to have one tester define the test cases and test procedures, and another tester execute the test procedures. The independent test executor tends to be more critical towards the test cases and the way they are executed and as a result will likely find more defects. Also for component testing one can make pairs whereby two programmers each tests each others' software.

- *Exit criteria*
  Different exit criteria, also called completion criteria, can be used for different risk levels. Requirements coverage or code coverage criteria should be more strict for higher risk areas. Other exit criteria that can be used to differentiate include percentage of test cases executed, number of outstanding defects and defect detection rate.

Note that the outcome of the product risk assessment can also influence the development process. Choices made in the development process will often have an effect on the residual product risks, especially on the likelihood of defects.

Of course initially, the content of the risk matrix is based on the perceived risks at an early stage of the project. During the project the test manager has to maintain the matrix, based on lessons learned e.g. defects found or other measured indicators like DDP (Defect Detection Percentage), (changed) assumptions, updated requirements or architecture. Changes in the projects' scope, context or requirements will often require updates of steps in the risk assessment process. The risk assessment process therefore becomes iterative.

8. Practical experiences with PRISMA

Unfortunately, it has not been not possible to perform a controlled comparison between test projects using PRISMA and other test projects, either using another method for risk based testing or not applying risk based testing at all. Therefore, to determine whether PRISMA offers benefits for the testing, a questionnaire study was performed within approximately twenty companies that already had applied PRISMA on several of their test projects.

The main goal of the study was the investigation whether the test managers (and testers) consider PRISMA beneficial for their task of detecting defects. The rationale behind this goal is that test managers tend to use or not use a method according to the extent to which they believe it will help them perform their job better. This determinant is referred to as the *perceived usefulness* of the PRISMA method. However, even if a test manager believes that a given technique is useful, he may, at the same time, believe that it is too difficult to use and that the performance benefits are outweighed by the effort of using a more systematic approach. Hence, in addition to usefulness, *perceived ease-of-use* is a second important determinant to take into account.

To define these concepts in more detail, the following definitions are introduced:
- *Perceived usefulness* is "the degree to which a person believes that using a particular method or technique would enhance his or her job performance." This follows from the definition of the word useful: "capable of being used advantageously." Hence, a method high in perceived usefulness is one for which a test manager believes in the existence of a positive use-performance relationship.
- *Perceived ease of use*, refers to "the degree to which a person believes that using a particular system would be free of effort." This follows from the definition of "ease": "freedom from difficulty or great effort". A technique, method, or tool that is easy to use is more likely to be accepted by users.

To measure the concepts of *usefulness* and *ease-of-use*, no objective measures are available. Hence, subjective measures are employed on a scale from 1 (strongly disagree) to 10 (strongly agree. One may also use the term *linear composite* to designate such a scale. A concept (i.e., *usefulness* or *ease-of-use*) is often characterised by a set of items. Usefulness has therefore been broken into two aspects: efficiency and effectiveness. For each of these items a statement is presented to the participants. The participants are asked to respond to each statement in terms of their own degree of agreement or disagreement. A score is assigned to each response. Table 1 presents the items that were considered for the survey.

Usefulness	
U1	To what extent is PRISMA *useful* for performing the test management task?
U2	To what extent does using PRISMA make the test process more *efficient*?
U3	To what extent does using PRISMA make the test process more *effective*?
Ease-of-Use	
E1	How *easy* is it to apply PRISMA in a real-life project?

Table 2: Survey items for usefulness and ease-of-use

8.1 Results for the Usefulness of PRISMA

Table 3 exhibits the results of the usefulness survey. Considering that the score rating is 10, we can conclude that most of our subjects tend to consider the PRISMA approach useful. The results of the usefulness determination reveal that PRISMA provides the expected benefits for test project. The test managers' perception on the various items indicates that PRISMA increases the effectiveness and efficiency of the defect detection tasks. This confirms the assumption that PRISMA exhibits a positive use-performance relationship.

Usefulness		Average score
U1	Usefulness	**7.5**
U2	Efficiency	**7.2**
U3	Effectiveness	**7.0**

Table 3: Results for the Usefulness of PRISMA

Interesting comments made by the participants regarding the usefulness of PRISMA include:

- it supports making the right decisions when the project is under pressure;
- risk is a business language and therefore PRISMA is a good tool to communicate with stakeholders;
- a good basis for test estimation and defining a differentiated test approach;
- it provides a framework for test monitoring and control during the project;
- a clear focus for the testing activities, also already during development testing;
- it ensures that the most important parts of the product are tested before release;
- as a result of the risk assessment, the priority level of the defects found has increased;
- a basis for learning and raising awareness about factors that influence the risk, also to be used during process improvement.

8.2 Results for the Ease-of-Use of PRISMA

Table 4 exhibits the results of the ease-of-use survey. It is clear that the score of ease-of-use is lower than with usefulness. In practice we found that consultancy training, workshop etc. are needed to get projects stated. Only having substantially invested in the implementation test managers find the method easy to use. Also the ease-of-use seems to depend on the test maturity level and test awareness in the organisation (see also specific comments hereafter).

Both statements can also be derived from the scores, whereby the ease-of-use score had a large standard deviation. Test managers with much experience using PRISMA tend to score much higher, e.g. 7 and above. Based on these findings, we can conclude that test manager tend to consider PRISMA easy to use provided that they have received the necessary initial practical support. Hence, there is a large probability that test managers adopt the PRISMA method in their test management practices. Also beware that within the organisation for stakeholders implementing PRISMA is a real change process. The process is simple but not easy.

Ease-of-Use		Average score
E1	Ease-of-use	**6.4**

Table 4: Results for the Ease-of-Use of PRISMA

Interesting comments made by the participants regarding the ease-of-use of PRISMA include:

- defining the more or less independent risk items at the correct level and grouping them into approximately 30 items is often a challenge;
- it is sometimes difficult to identify the (business) stakeholders and to get them involved, especially the first time;
- for some stakeholders it involves a change of mind set, they now more explicitly become the risk owners;
- making explicit choices is difficult to some (business) stakeholders, they always thought everything was tested 'fully';
- the interpretation of factors is not easy, a kick-off and good definition of the factors (including scoring rules) are highly recommended;
- development is not always in line with testing priorities, the most important risk items are delivered relatively late in the process;
- defining a differentiated approach based in the risks is difficult, it also depends on the knowledge and skill level of the test engineers involved;
- finally most of the risk assessment is built upon perceived risk early in the project, as projects tend to be dynamic and people learn throughout the project the risk assessment also should be treated this way and (partly) repeated at several instances, e.g. at milestones.

Finally, one company released defect numbers recently after measuring DDP for several years at alpha level. In addition to a shorter test execution lead time their DDP has improved by approximately 10% after the introduction of PRISMA in 2003, as can be observed by looking at the graph hereafter.

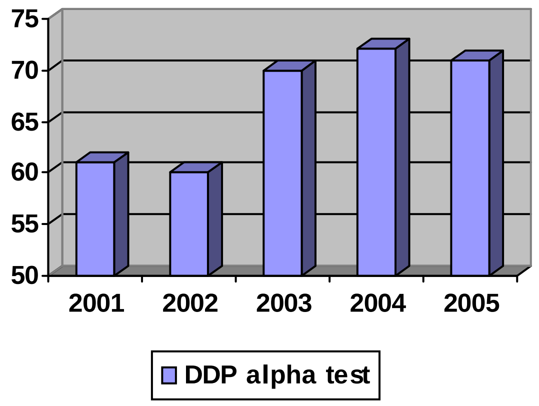


Figure 7: DDP number Alpha test level

Acknowledgements

I would especially like to thank Ruud Cox, Stephanie van Dijk, Paul Gerard, Rob Hendriks, Jurian van der Laar and Hans Schaefer for their contribution to this paper. I would also like to thank the many companies that have applied earlier versions of PRISMA in their test projects and provided us with valuable feedback making the method what it is today.

References

Bach, J. (1997), Good Enough Quality: Beyond the Buzzword, in: *IEEE Computer*, August 1997, pp. 96-98

Bach, J. (1998), A framework for good enough testing, in: *IEEE Computer Magazine*, October 1998

Boehm, B.W. (1979), *Software engineering economics,* Prentice-Hall, Englewood Cliffs, NJ

Jørgensen, M. (1994), *Empirical studies of software maintenance*, Thesis for the Dr. Sceintific degree, Research Report 188, University of Oslo

Gerard, P., and N. Thompson, *Risk-Based E-Business Testing,* Artech House Publishers, ISBN 1-58053-314-0

Karlsson, J. and K. Ryan (1997), A Cost-Value Approach for Prioritizing Requirements, in: *IEEE Software*, September 1997

Khoshgoftaar, T.M., E.B. Allan, R. Halstead, G.P. Trio and R. M. Flass (1998), Using Process History to Predict Software Quality, in: *IEEE Computer*, April 1998

Levendel, Y. (1991), Improving Quality with a Manufacturing Process, in: *IEEE Software*, March 1991

Pol, M. R. Teunissen, E. van Veenendaal (2002), *Software Testing, A guide to the TMap Approach,* Addison Wesley, ISBN 0-201-745712

Schaefer, H. (2004), Risk Based Testing, in: E. Van Veenendaal, *The Testing Practitioner – 2nd edition,* UTN Publishing, ISBN 90-72194-65-9

Veenendaal, E. van (2004), *The Testing Practitioner – 2nd edition,* UTN Publishing, ISBN 90-72194-65-9

The Author

Erik van Veenendaal is the director and a senior consultant of Improve Quality Services Ltd., a company that provides consultancy and training services in the area of testing and quality management. He is the author of numerous papers and a number of books on software quality and testing, including the best-sellers ‘*The Testing Practitioner*’ and ‘*Testing according to TMap*’. He is a regular speaker both at national and international testing conferences and a leading international (ISTQB accredited) trainer in the field of software testing. Erik has also been a senior lecturer at the Eindhoven University of Technology, Faculty of Technology Management for almost ten years. Currently he is the vice-president of the International Software Testing Qualification Board (ISTQB). He is also the editor of the ISTQB “Standard Glossary of terms used in Software Testing” and recently became the vice-chair for the TMMi Foundation.

Annex: Calculation example

In this simplified example three stakeholders are involved in providing input and scoring. A project manager who will provide a score for all factors, a business manager who will only score the impact related factors, and the architect who will only score the likelihood related factors. For this example the value set 1 to 5 is used.

<u>Results of individual scoring:</u>

For every factor chosen, you assign a number of points to every risk item (product requirement, function, functional area, or quality characteristic). The more important the risk item is, or the more alarming a defect generator seems to be for the area, the more points.

Project manager	**Impact of defects**		**Likelihood of defects**	
Factor:	**User Importance**	**Usage intensity**	**Complexity**	**New development**
Weight:	**2.0**	**1.0**	**1.0**	**2.0**
Risk item 1	5	3	5	5
Risk item 2	3	5	5	1
Risk item 3	3	2	1	5
Risk item 4	4	1	2	2
Risk item 5	1	2	3	3

Business manager	**Impact of defects**		**Likelihood of defects**	
Factor:	**User Importance**	**Usage intensity**	**Complexity**	**New development**
Weight:	**2.0**	**1.0**	**1.0**	**2.0**
Risk item 1	4	2		
Risk item 2	3	5		
Risk item 3	4	3		
Risk item 4	5	1		
Risk item 5	2	3		

Architect	**Impact of defects**		**Likelihood of defects**	
Factor:	**User Importance**	**Usage intensity**	**Complexity**	**New development**
Weight:	**2.0**	**1.0**	**1.0**	**2.0**
Risk item 1			4	5
Risk item 2			4	2
Risk item 3			1	5
Risk item 4			3	1
Risk item 5			5	3

Average scores (calculated by the test manager):

	Impact of defects		Likelihood of defects	
Factor:	**User Importance**	**Usage intensity**	**Complexity**	**New development**
Weight:	**2.0**	**1.0**	**1.0**	**2.0**
Risk item 1	4.5	2.5	4.5	5.0
Risk item 2	3.0	5.0	4.5	1.5
Risk item 3	3.5	2.5	1.0	5.0
Risk item 4	4.5	1.0	2.5	1.5
Risk item 5	1.5	2.5	4.0	3.0

Weighted sum:

The average number of points for a factor is now multiplied by its weight. This gives a weighted number of points. These weighted numbers are then summed up for impact and likelihood. Testing can then be planned by assigning the greatest number of tests to the areas with the highest number of points.

	Impact of defects	Likelihood of defects
Risk item 1	9.0 + 2.5 = 11.5	4.5 + 10.0 = 14.5
Risk item 2	6.0 + 5.0 = 11.0	4.5 + 3.0 = 7.5
Risk item 3	7.0 + 2.5 = 9.5	1.0 + 10.0 = 11.0
Risk item 4	9.0 + 1.0 = 10.0	2.5 + 3.0 = 5.5
Risk item 5	3.0 + 2.5 = 5.5	4.0 + 6.0 = 10.0

Resulting risk matrix:

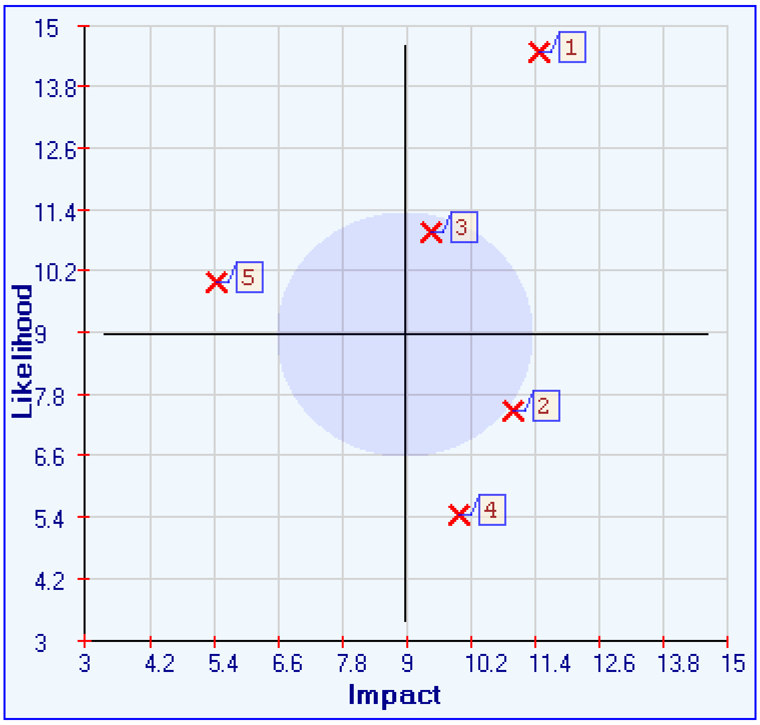


Figure 8: Resulting product risk matrix (from the PRISMA tool)

The risk matrix suggests that items 1 and 3 are first priority, although item 3 may need some discussion since it is close to the centre. From a business perspective 2 and 4 are the second priority level.

A word of caution: This calculation is not theoretically 100% correct. Actually you are multiplying apples with oranges and summing up fruit salad. Thus, the number of points can only be a rough guideline. It should be good enough to distinguish the high-risk areas from the medium and low risk areas. That is its main task. This also means you don't need to be more precise than needed for just this purpose.