



PACIFIC NW
28TH ANNUAL
SOFTWARE
QUALITY
CONFERENCE

OCTOBER 18TH – 19TH, 2010



ACHIEVING
QUALITY
IN A COMPLEX
ENVIRONMENT

*Conference Paper Excerpt
from the*
CONFERENCE
PROCEEDINGS

Permission to copy, without fee, all or part of this material, except copyrighted material as noted, is granted provided that the copies are not made or distributed for commercial use.

Testing As a Risk Management Activity

Alan S. Koch
President, ASK Process, Inc.
ask@ASKProcess.com
www.ASKprocess.com

Abstract

Managers and testers alike often describe testing as being all about finding bugs. This is a natural conclusion one might draw after observing the testing process. Look for bugs, report bugs, then ensure they get fixed. Pretty simple.

But there is a nasty problem with this. As we all know, testing all the defects out of a product is impossible in nearly all cases. And as any student of logic will affirm, even if we *could* find and dispatch every defect, it is logically impossible to prove that no more defects exist. (You can not prove the absence of something; only its presence.)

So if the objective of testing is to remove the defects from the product, we are virtually guaranteed to fail to some extent. Defects will persist.

But testing has a higher objective. Yes, testing finds defects (and of course we fix the defects we know about), but finding those defects is a means to a different – although a related end. Every product may fail in some non-trivial way that would have adverse impacts on the users or the organization. This is a risk that needs to be managed, and testing enables us to manage that risk.

Biography

Author, speaker and consultant Alan S. Koch and his company ASK Process, Inc. are celebrating 10 years of consulting with IT organizations and training IT professionals to ensure dependable, smooth, cost-effective IT Systems and Services. They capitalize on published and emerging standards to address customers' unique challenges, employing pragmatism to make customers self-sufficient.

Mr. Koch's more than three decades of experience gives him unique insight into testing from the viewpoints of tester, developer and manager. As he has come to grips with the costs, benefits, strengths and weaknesses of testing, he has progressed from merely finding bugs to viewing testing as a much more strategic activity.

The peer-reviewed journals Better Software Magazine, Crosstalk and Software Quality Professional have featured articles authored by Mr. Koch.

Mr. Koch has been invited to speak, lead workshops and provide training in such diverse places as Orlando FL, Portland OR, Zurich Switzerland, Edinburgh Scotland and Tianjin China.

IEEE (Institute for Electronic and Electrical Engineers) awarded to Mr. Koch the designation of **Senior Member**, "In recognition of significant contributions to the industry".

© Copyright 2010, ASK Process, Inc. (Reproduced by PNSQC with permission)

1. The Purpose of Testing

Software projects follow a regular progression. Requirements, Design, Code, Test, Deploy. We don't tend to give much thought to this progression or the purpose of each step. If we *do* engage in such thought, it becomes clear that most of these steps have a clear reason for being that can be used to judge their effectiveness.

- The **Requirements** phase is all about capturing a clear understanding of what should be built. It is effective if the end product is fit for purpose and fit for use.
- The **Design** phase is all about deciding *how* the product should be built. It is effective if development goes smoothly without technical surprises and issues.
- **Coding** is all about transforming the Design into executable code. It is effective if the code functions as specified in the Requirements and Design.
- **Deployment** is all about delivering the product to the customer. It is effective if the transition goes quickly and smoothly, as planned.

But what of Testing? If the purpose of testing is to find bugs, then how do we judge its effectiveness? Number of bugs found? How many is enough? How many is too few?

What if the Requirements, Design and Code steps were *very* effective? Would the lack of defects to be found mean that Testing is *ineffective*? What if the Requirements, Design and Code steps were *ineffective*? Would the multitude of defects to be found mean that Testing is *more* effective?

Ultimately, if the purpose of testing is to find defects, then every defect that escapes Test is a failure of the testing activity. If you do not find them all, then you have failed. And the more defects that testing misses, the “worse” the testing fails. But how do we know how many defects we missed? We cannot know at the time. We come to understand this very important metric only after the product has been in use for some period of time.

The reality is that testing is a filter; it will detect only a certain percentage of the existing defects, and the rest will escape. The major determiner of defect density of the product *after* test is defect density *before* test. More defects to be found means that we not only *find* more defects in Test, but we *miss* more as well.

All of this can be very depressing – *if* testing is all about finding defects. But the purpose of testing can and should extend far beyond finding defects. To discover how, we need look no further than the needs of Management.

2. What Management Really Needs to Know

As testers, we report lots of metrics to management. Numbers of defects. Defect arrival rate. Defect backlog (number of defects found but not yet fixed). Number of tests run. Number of tests passed. Percent of tests failed. And on and on.

Sometimes they ask us for this data, but more often we report these things because it is the best way we can think of to report status to them. But what do they *really* want to know? What is the burning question behind their desire to see these data (or whatever data they ask for)?

“When will it be ready to deploy?”

Of course, the key word in this question is “ready”! What does it mean for the product to be “ready” to deploy? What constitutes “ready”? If our job is to find the defects, then one might interpret “ready” to mean defect-free. (This interpretation is a common problem when managers do not understand the nature of defects in software.) But we know that defect-free is not a realistic expectation, and is most certainly beyond our reach, given our limited budgets and time.

So, if “ready” does not (or should not) mean defect-free, then what is management asking us when they pose the question, “When will it be ready?” In order to understand, we must look at it from their perspective.

The calculus of management can be boiled down to four R’s:

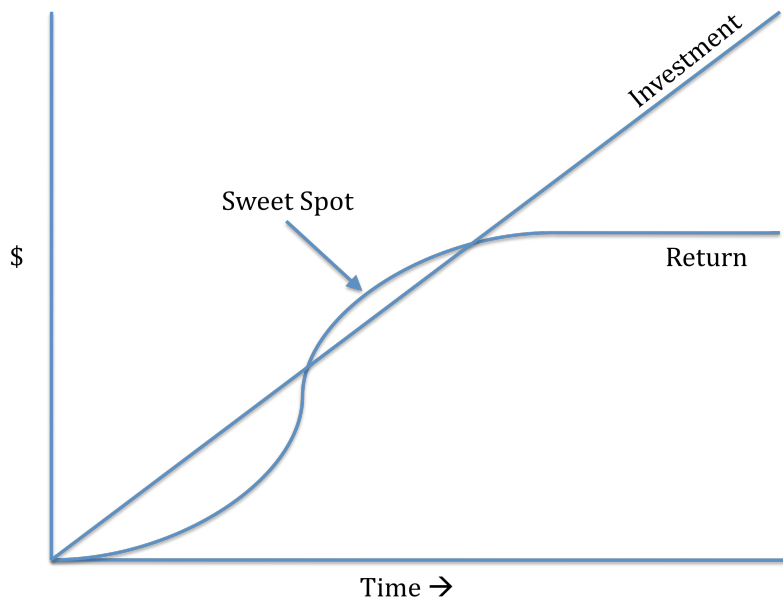
- Requirements: What we need to achieve
- Resources: The investment we must make to achieve the Requirements
- Returns: The benefits or payback on those invested Resources
- Risk: The things that get in the way of those Returns

At the point in the project when they are asking when the product will be ready to deploy, they have (hopefully) understood the Requirements, they have invested the Resources, and they are anticipating the Return. What they are asking about is the Risk.

“If we deploy today, what is the Risk that problems will reduce the Returns we expect to realize?”

“If we delay deployment and invest more Resources (time, effort, money), will it reduce the Risk and give us a better certainty of the Returns?”

Management is always looking for the “Sweet Spot” where the Return exceeds the Resources invested. The big wildcard in the whole calculus is Risk. Without Risk, Return on Investment (ROI) is easy to compute. Management could run the numbers, create a graph like the one below, and determine precisely when to stop making investments.



But risk changes the shapes of those two lines in unpredictable ways. So making the right choice about when to stop making investments requires a good understanding of the Risks. That is what they are looking to us as testers to help them to understand.

If we can help them to understand how much risk there is, they can do a better job of predicting if they have reached that Sweet Spot. If we can be more precise about risk (e.g. what types of risk remain, or what parts of the product constitute risk), then they can make even better estimates!

The good news is that we are in a very good position to provide the information that management needs to make good choices. And to the extent that we understand our role as advisors on the subject of risk,

we will be able to plan and tune the testing process so we can provide the critical information that management needs.

3. What testing can tell us about risk

Let's take a look at the types of things that our job as testers can help us to understand about risk in a software product.

When we plan our tests, we make choices about the types of tests to run and the conditions under which to run them. We know that we will not be able to test every possible set of inputs or conditions, so we try to choose those conditions and inputs that will give us the most information. Because a test that fails tells us more about the system under test than a test that passes, we have a natural bias toward tests that we judge to be more likely to fail. We are probing for defects, looking in the most likely places.

Our job is not unlike that of an exterminator brought in to determine if there are bugs in a house. The exterminator will spend little time probing hard materials like concrete and brick because few bugs are likely to be found in those conditions. Instead, he will focus on soil and wood, and especially areas of moisture. Those are where the bugs will be (if indeed there are any in the house). Testers, like exterminators know where to look for bugs, and the more experienced testers are, the better they become at knowing where to look.

Research has shown that bugs in software are indeed much like those in houses in that they tend to cluster together. Any exterminator will tell you that if you see one bug, there is a high likelihood that there are hundreds more nearby. The same is true of software, not because software bugs breed, but because they result from the same root causes. Root causes like:

- Complexity of functionality
- Errors in design
- Oversights by the developer
- Lack of knowledge or skill in development

Armed with this understanding, we can use the results of our tests to make hypotheses about the existence of bugs related to the tests we just ran. Specifically, after running a suite of related test cases, we can posit these sorts of hypotheses:

- If the tests resulted in no defects found, then we can posit that the likelihood of defects that we failed to detect in that particular part of the product is relatively low.
- If the tests revealed one or two defects, then we might suspect more in that particular part of the product and look for ways to confirm that suspicion.
- If the tests revealed many defects, then we can posit that the likelihood of even more defects in that particular part of the product is relatively high.

So, what does this tell us about risk? One of the two dimensions of risk is the likelihood that something bad will happen. Since undetected defects in the product are clearly bad, the fact that we can postulate about their likelihood means that we have an important handle on risk. This kind of handle is valuable to managers.

The other dimension of risk is the impact if that bad thing comes to be. Testers often have good insight into this as well, and that along with our understanding of likelihood constitutes a valid assessment of risk. (And even if we cannot evaluate the impact of those likely defects, management or others will be able to fill in that dimension, making our assessment of the likelihood quite valuable!)

So, we test where we suspect defects may be lurking. If we find defects there, we suspect an infestation and distrust that part of the product. Conversely, if we fail to find defects where we suspected they would be, it strengthens our confidence in that part of the product.

4. Risk-Based Testing: A Test Planning method

Knowing how to translate test results into an assessment of risk is powerful. But we can tune our test plans to make this information about risk even more valuable to management by adopting a test planning technique called “Risk-Based Testing”. This is a method that starts the test planning process with an assessment of risk and focuses testing in the areas of highest risk.

The first step in Risk-Based testing is to identify the types of risks inherent in the product. There is a wide diversity in these types of risks, but they tend to fall into two broad categories:

- 1) Risks that are likely because of how development is done. Some examples are:
 - New technology may not function as expected
 - Technology or techniques that are unfamiliar to the development team may be misused
 - Less knowledgeable or experienced developers may make mistakes
 - Schedule pressures may cause developers to take shortcuts
 - Changing scope can invalidate design decisions causing complex workarounds in the code
 - Resumption of interrupted work can result in oversights
 - Changes in staffing can result in developers working with code they do not understand
 - Changed code is more likely to fail than new code
 - New code is more likely to fail than un-touched code
- 2) Risks that will have a serious impact on the customer or user of the system. Some examples are:
 - Portions of the product upon which the customer will depend for critical business functions
 - Failures that could go undiagnosed in production resulting in long-term impacts
 - Failures that would have catastrophic impacts (e.g. loss of life or significant financial harm)

The second step is to rank the risks against each other by estimating both the probability and the impact of each. (The most important risks are those that are both highly likely and high-impact.) This results in a prioritized risk list that becomes our guide for our test plans.

The third step is to plan testing based on the prioritized Risk list. The highest-priority risks should be tested as early as is practical in the testing phase, and the lowest-risk items should be tested last. This has three beneficial effects:

- First, it ensures that the higher-risk parts of the product receive un-rushed attention during a time in the testing phase that is least chaotic.
- Second, it ensures that if there are problems in that area of the product, the development team will have plenty of time to address and correct them before deployment.
- And third, it ensures that the tests that may be skipped when time crunches at the end of the project will be the ones that are associated with the lowest-risk parts of the product.

In addition to being addressed early, the high-risk areas should also receive the most attention. So we will plan to test those areas more thoroughly. We will create more test cases for them, and will be sure to test more combinations of inputs and conditions.

Finally, when we are actually writing our test cases and creating our test data, we will do so with the nature of each risk clearly in mind. We will tailor each test to ensure that the risk that we have identified is actually tested so we can be sure that the test results provide valuable information about the risks we have identified.

Also, as each test suite and test case is written, it should be tagged with the identifier and priority of the risk that it is designed to address. Having this information at our fingertips will help us to quickly evaluate the implications of the test results, whether the test passes, fails, or is not run at all.

Risk-based Testing is powerful not only because of how it focuses our testing effort, but also because it provides key information we need to interpret our test results and provide management with information about risk.

5. When you find more defects than expected

Of course, we always expect to find defects during testing. But in some projects, there comes a time when we have found *too many* defects – when we begin to believe that the product is buggy. As we observed earlier, finding many defects in a particular part of the product indicates that more are likely to be lurking.

Clearly, this is an area of concern because the risks associated with deployment are greater. With more defects lurking in the product, there is a greater likelihood that users will be adversely impacted by them. This understanding of risk alone is invaluable. It can be the information that management needs to decide to make some additional investment to try to mitigate the risk.

But when we have used a Risk-Based approach to our test planning, we have even more information available. Will the part of the product that is suffering from failures have a high impact on users? If so, the increased likelihood of defects combined with the high impact of the functionality means that this is a high-risk situation. If management is aware of the severity of the risk, they may make different choices about making additional investments of resources to mitigate that risk.

By the same token, if the risk of failure is in a lower-impact part of the product, management will probably be more comfortable with living with the risk, even though the likelihood of problems is high. Again, the additional information is a goldmine for management decision-making.

Of course, when a product has many defects, testing tends to take longer so that there will be some significant number of planned tests that have yet to be run when the scheduled end of testing arrives. This is very much like having your testing schedule compressed, which we will address next.

6. When test time is cut short

A common situation we testers find ourselves in is compression of testing time. Development went over schedule and the product was delivered to test late, but the final delivery date remains unchanged. We are told to test “more quickly” to meet the delivery date. Sometimes this mandate is arbitrary, but other times there are good reasons why the delivery date must be honored.

Taking a Risk-Based approach gives us the most appropriate way to respond to such an edict. We can articulate to management the risks that are involved in the compressed testing schedule as compared to the originally planned one. We do this by assuming that we will execute our risk-prioritized test plan up to the point when time runs out.

If we have been instructed to work overtime, then we can adjust our plan to account for the additional effort. Just beware of the diminishing productivity of overtime. If we are to work 50% more hours per week than normal (e.g. 60 hours per week instead of 40), we should assume we will get 40-45% more work done in each of the first few weeks. If this pattern is to continue for a month or more, the additional productivity will begin to drop until after 2-3 months, productivity of the 60-hour weeks will approximate our normal 40-hour productivity.

After adjusting our plan for overtime (if needed), we can forecast the tests that we will not have time to run. Each set of functionality that we had planned to test but will not have time for represents a risk of defects. And since we already assessed the risk involved with each, we are already armed with the information that management needs. We must give them this risk information.

7. Go? Or No-Go?

Although management may be asking us when the product will be ready to deploy, in most organizations that decision is not ours to make. They aren't actually asking us to make the decision; they are asking us for pertinent information that they will use to make it.

It is our duty to bring this information about the risk of deployment to management so they can make an informed decision about the project schedule. If there is flexibility in the end-date, they might choose to allow for additional testing to reduce the risk of deploying a defect-ridden product. Then again, they may choose to accept the risk and stick with the original schedule. That is their decision to make. Your job is to give them the risk data they need to make an informed decision.

(If you are in an organization that gives you as the tester the authority to stop product releases, then you should use this information precisely as a manager would. You should assess and understand the risks and make appropriate decisions based on that information.)

If there is *no* flexibility to extend the schedule, this information is still important. When management knows of risks, they will often have a variety of ways to mitigate their effect. Even if the schedule is fixed, there may be other ways in which that information can be valuable to them. So again, it is our duty to bring it.

When we testers understand our role in the management of project risks, we can plan our testing process and report status in a way that puts important risk information into the hands of the decision-makers. And this will result in better decisions and ultimately better project results.