Testing and Risk Reduction

Testing and test planning in the Iterative Project Model (IPM)

Evaluating Your Project

The purpose of testing is to reduce risk.

Any project includes a fair amount of risk. The unknown factors within the development and design of a project can derail the whole endeavour and more minor risks can delay the project or inflate costs to unacceptable levels. The purpose of testing is to reduce uncertainty and hence the risk in the project.

It is important to note that testing is the only tool in a project manager's arsenal which reduces risk. Any kind of development work introduces more risk since it introduces more complexity and increases the potential for errors to be introduced. Planning and design can help to limit risk but they will not reduce or eliminate the risk already present in a project.

Through a cycle of testing and resolution a project manager can progressively eliminate risks and errors in a project. Note however that the resolution of errors still offer the chance of introducing more. For this reasons testing and evaluation must always be the last phase of a project. The objective is to leave the project in a "known good state".

The Testing Mindset

There is particular philosophy that accompanies "good testing".

A professional tester approaches a product with the attitude that the product has defects and it is their job to discover them. They assume the product or system they receive is inherently flawed and it is their job to 'illuminate' the flaws. This attitude is necessary but can bring them into conflict with developers and designers.

This 'chicken little' approach really is necessary to testing. Designers and developers approach software with the an optimism based on the assumption that the product is basically working correctly and just needs to be refined to be complete. This means that they often overlook fundamental issues or fail to recognize them when they see them. By taking a skeptical approach the tester offers a balance in that assessment. A project manager should draw on both sources of opinion in making his or her decisions.

The project manager must manage the schedule and delivery but also be responsible for the ultimate quality of the project. In the final stages where an issue is uncovered it is often far too tempting for a project manager to simply overlook quality in favor of the delivery deadline. Project managers who succumb to this temptation will pay the price later.

Principles of Testing

Test Early, Test Often

There is an oft quoted truism of software engineering that states : a bug found at design time costs ten times less to fix than one in coding and a hundred times less than one found in implementation.

The principles embodied in this book emphasize the need to reduce the introduction of design and coding errors through the use of good development methodology. Ever process is flawed however and errors will unavoidably creep in despite your best efforts. Early use of testing and evaluative procedure will help you eliminate errors before the increasing complexity of your project makes them difficult or costly to rectify.

A single pass of testing is never enough either. Your first past at testing simply identifies where the issues occur. At the very least a second pass of (post-fix) testing is required to verify that the issues have been correctly resolved. The more passes of testing you conduct the more confident you become and the more you should see your project metrics converge on the final state of your project.

Regression vs Retesting

From the above explanation the need for re-testing is fairly self evident. You must re-test fixes to ensure that the issues has been resolved satisfactorily before development can progress. By contrast the need for regression testing is not well understood, except within enlightened circles of the software development industry.

Broadly speaking re-testing is the act of repeating a test to verify that a previously found issue has been correctly fixed. Regression testing is the act of repeating other tests in associated areas to ensure that the applied fix has not introduced other errors or unexpected behavior.

For example if a plane suffers from excessive flex in the wings then the structural members of the wing and the brackets holding it to the body of the plane may be stiffened. This will cure the flex in the wings but it will have the unfortunate effect of placing more load on the body of the plane, possibly with disastrous consequences.

Verification and Validation

There are two major types of tasks in testing : verification and validation.

Verification tasks are tasks designed to ensure that the product is internally consistent. They ensure that the product meets the design which meets the specification which meets the requirements and so on. By and large the majority of testing tasks fall into the verification category with the final product being checked against known good references to ensure the output is within expected tolerances.

Validation tasks are just as important but less common. Validation is the use of external sources of reference to ensure that not just the product but the whole design meets the expectations of users or clients. As previously discussed specifications and requirements are all necessarily incomplete models of the final product. Basing the ultimate success or failure of a project on these is a dangerous prospect at best and the wise project manager will seek independent and external validation of their finished product.

Independence

Independence in testing is an absolute requirement. In order to effectively do their job testers must be given a degree of freedom from design and build teams. Testers have the unenviable job of assessing and reporting the quality of a product and must have a degree of independence to be able to do so objectively. Conversely the test team relies upon the other teams for the entirety of its input and so must tread a fine edge between independence and co-operation to build trust within the teams.

As a project manager however you are most concerned with the independence of your test team and the accuracy and objectivity of their results. A good test team will take care of the intra-team relationships for you and you can focus on maintaining their independence so you have a clear and reliable source of information on the status of your project.

Test Planning

The Purpose of Test Planning

Like many elements of software project management testing is a discipline in itself and embodies a mini-project in its own right. Like all good projects testing must be planned to ensure it delivers on its expected outcomes.

Test planning represents a special challenge however. Essentially the aim of test planning is to decide where the bugs in a product or system will be and then to design tests to locate them. The paradox is clear, if we knew where the bugs were then we could fix them without having to test for them.

The naïve retort is that you simply test “all” of the product. Even the simplest program however will defy all efforts to achieve 100% coverage. Even the term coverage itself is misleading since this represents a plethora of possibilities.

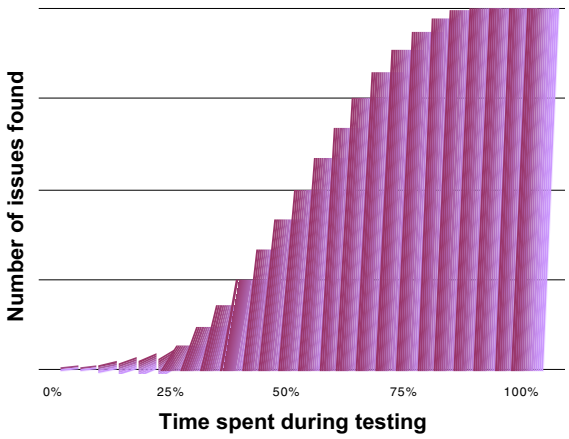
For example do we want 100% coverage of the code ? But does 100% coverage of the code mean “line coverage” (executing every statement in the system or product at least once) or does it mean “branch coverage” which is more logical (executing every logical branch and path within the code at least once). Taking branch coverage for example we can calculate the number of paths within a simple system or unit of code using techniques such as LCSAJ and the McCabe complexity metric. This in turn will determine the number of separate tests we will need to conduct to achieve “100% coverage”. Not surprisingly this number turns out to be frustratingly large for anything but the simplest piece of code.

So perhaps we have achieved 100% coverage of the code, but what about 100% coverage of the input and output? Exercising the code simply means you have tested every executable statement or branch decision with a particular value of input. To test it completely you would also need to cover all the possible input and output values since each one could possibly provoke an error. The possible input and output space of even the simplest of programs can run into trillions upon trillions of combinations and while this can be reduced using techniques such as "equivalence classes" and "boundary value analysis" the task is simply too great.

The stark reality for testing is that complete coverage of any sort is simply not possible.

The answer is reasonably simple. We already have a key predictor of project success and that is risk. By using risk to drive our testing we can achieve a reasonable balance between the risks associated with a project and the commercial returns. If additional testing will subsequently reduce our risk and increase our potential returns then the cost of performing that testing can be weighed against the risk of failure should it not be carried out.

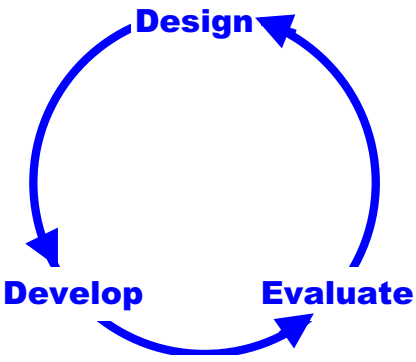
At the start of testing there are a (relatively) large number of issues with the project and these can be uncovered with little effort. As testing progress the law of diminishing returns applies and more an more effort is required to uncover subsequent issues. At some point the return-on-investment to uncover that last 1% of issues is outweighed by the high cost of finding them. The cost of letting the customer or client find them will actually be less than the cost of finding them in testing.



The purpose of test planning therefore is to put together a plan which will deliver the right tests, in the right order to discover as many of the issues with the product or system as possible.

The Process of Test Planning

Given that we don't *a priori* know where issue are going to occur in the product or system we must design a test regime which makes the most efficient effort to uncover issues.



This is where the Iterative model comes into play again. By using the model of "design-develop-evaluate" we can constantly refine our testing approach to ensure it delivers the best results.

By starting with a broad based testing approach we can identify areas of likely risk and focus our efforts on

But how to identify those areas of risk ?

It is useful to think of software as a multi-dimensional entity with many different axes of complexity. For example one axis of complexity is the code of the program, which will be broken down into modules and units. Another axis will be the input data and all the possible combinations. Still a third axis might be the hardware that the system can run on, or the other software the system will interface with or possible

Testing can then be seen as an attempt to achieve "coverage" of as many of these axes as possible in an attempt to find issues. Remember we are no longer seeking the impossible 100% coverage but merely an indication of where issues lie, where the likely areas of risk are. Testing can then be focused on these areas in order to find and eliminate the issues.

Outlining

To start the process of test planning a simple process of 'outlining' can be used.

Taking each of the axes of complexity, break each one down into its component parts. This is essentialy a process of deconstructing the software into constituent parts based on different taxonomies. For each axis simply list all of the possible combinations you can think of.

The number and extent of your 'axes' will vary dependent upon your system or product but some likely axes are presented here :

- Functionality (as per the spec)
- Internal and external interfaces
- Physical components (manuals etc)
- Data
- Configuration elements
- Localisation / Internationalisation
- Code structure and organization
- Input and output parameter space
- User Interface elements
- Platform variables (hardware, O/S, etc)
- Error conditions
- End-users (differing roles, interfaces etc)

Test Case Design

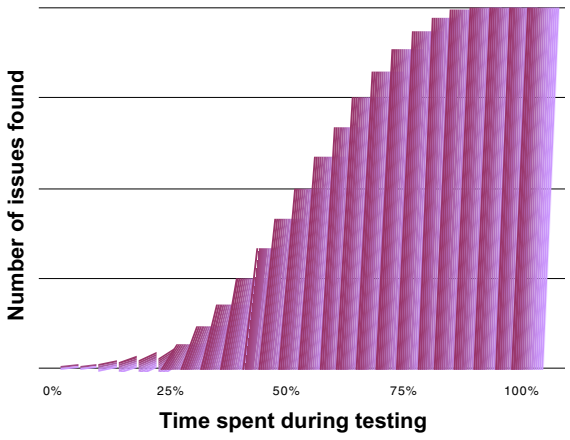
The next step in testing is to design test cases which cover or exercise each of the points detailed on your outline. Note that a single test may in fact validate more than one point on one axis. A test could simultaneously validate functionality, code structure, a user interface element and error handling.

It is likely that the detail of your outlining will preclude a complete set of test cases so your aim should be to provide a broad coverage for the majority of your outline points and deep coverage for the most risky areas outlined. Broad coverage implies that an element in the outline is evaluated in a cursory or elementary fashion while deep coverage implies a number of repetitive, overlapping test cases which exercise every conceivable variation in the element under test.

The aim of broad coverage is to identify risk areas and focus the deeper coverage of those areas to eliminate the bulk of issues. It is a tricky balancing act between trying to cover everything and focusing your efforts on the areas that require most attention.

Remember too that risk is defined by the likelihood of an issue arising and it's impact when it does arise. Therefore your testing should focus deep coverage of areas where issues are likely to occur or where the impact will be particularly significant. Likelihood is determined or identified by factors such as code or system complexity, historical occurrences of issues, stress under load and reuse of code. Impact is determined by the visibility of the issue and the importance of the implied failure to the end-user or to the system as a whole.

Refinement



As you progress through each Iterative cycle of design-develop-evaluate you can further refine your test plan. As each cycle of testing uncovers more issues you can shift testing to the more risky areas of your product or system. Typically issues are found in a frequency demonstrated by the curve shown at right. During the early stages of testing not many issues are found. As testing hits it stride issues start coming faster and faster until the development team gets on top of the problem and the curve begins to flatten out again.

This is the point where your risk/reward ratio begins to flatten out and it may be that you have reached the limits of effectiveness with this particular form of testing. If you have more testing planned or more time available now is the time to switch the focus of testing to a different point in your outline strategy.

Cem Kaner said it best when he said “the best test cases are the ones that find bugs”. A test case which finds no issues is not necessarily worthless but it obviously is worth much less than a test case which *does* find an issue. Your efforts must focus on those test case that find issues. The cycle of refinement should be geared towards discarding inefficient tests and diverting attention to more fertile evaluation of the software.

Also referring each time to your original outline will help you avoid losing sight of the wood for the trees. While finding issues is important you can never be sure where you’ll find them so you can’t assume the issues that you are finding are the only ones that exist. You must keep a continuous level of *broad coverage* testing active to give you an overview of the product or system while you focus the *deep coverage* testing on the trouble spots.

For more discussions on the benefits of iterative development methods, see my other papers on the Iterative Project Model (IPM).