RBT Estimating Guidelines

These estimating guidelines are based on using the full Bender Requirements Based Testing process, supported by the BenderRBT Test Case Design tool. It also adds some commentary so that any comparison to the effort using alternative test design techniques can be compared in an "apples to apples" manner.

Ambiguity Reviews

Document based specs: 20 pages of detailed rules per person day.

Requirements Management tool with ambiguity property sheets set up: 40 pages per person day.

Note: this is the effort to identify the ambiguities, not resolve them.

Working with analysts whose specs have not been through this process before, we expect to find 5 to 10 ambiguities per page of detailed specs. Once they have had 5 or more use cases reviewed we expect the initial ambiguity rate to drop by 95%. We have found that if something is ambiguous in the specifications, there is nearly a 100% probability that there will be one or more defects in code caused by that ambiguity. Therefore, a significant number of defects are avoided before ever being created. Defect avoidance is a lot cheaper than even early defect detection.

Also note that the task of doing ambiguity reviews is not an additional "overhead" task. Specs are always passed around for review. The ambiguity review process just refines how that task is done so that it is far more effective.

Cause-Effect Graphing

For each screen, use-case, or event/transaction about 2 graphs (models) are generated.

An average model will have about 175 functional variations.

Each model will take about 12 to 15 hours to create. Another view is that the combined ambiguity review and modeling will take about 20 to 25 hours per model. Note that some practitioners prefer to combine these two tasks and iterate until the model and specs are complete.

Anything under 20 models is a small project. 20-50 is a medium project and big projects are over 50. You also need to factor in complexity, functional stability, and other factors to refine the model count, but it is the model count that drives the sizing.

Test Case Generation

Each model will generate about 15 tests. These are the detailed test specifications, not the executable test.

The total effort to design tests via Cause-Effect Graphing has been shown to be about the same as it would have taken to design tests via alternative methods such as pair-wise testing. You just spend your time differently.

When pair-wise testing is used, for example, you wind up with about 4 times as many tests. The tests generated by the various pair wise tools only identify the inputs. The testers must then manually determine the outputs for each test. Without that information you really do not have a test case. BenderRBT, on the other hand, is an oracle – it identifies the expected results.

Most of the test design tools, unlike BenderRBT, do not factor in Constraints. The tester must manually weed out illogical tests – i.e. tests that cannot be physically created since that combination of states/data is physically impossible due to the structure of the data and/or editing that preceded that function. Note that this is different than negative tests, which BenderRBT is very strong at.

In addition to the above BenderRBT is the only test design tool that validates that the requirements are logically consistent – i.e. the pre and post conditions of the prior functions are in synch with the intra-process rules of the function being analyzed.

Also, BenderRBT is the only test design tool that addresses ensuring that, when you execute a test, you got the right answer for the right reason. That is it addresses defects that can cancel each other at times. It also addresses propagating a defect to an observable point. The other techniques/tools just address trying to reduce the big number of possible tests down to a manageable number.

Executable Test Creation

The effort to create an executable test varies greatly depending on the application profile – e.g. on-line versus embedded; real time, asynchronous versus sequential; flat file versus complex relational data bases. If the DTT add on is being used, then the executable scripts are automatically generated in the appropriate playback tools. If the Grid Tools Datamaker utility is being used then the test data bases are automatically generated. If scripting and data base creation are being done manually the effort increases.

That said, numerous studies by our clients show that RBT generates about half the tests for twice the coverage – a 4 to 1 reduction in the number of tests for equivalent coverage. We normally estimate that it takes about 3 to 5 times the effort to create an executable test versus the time it took to spec it out. That means that the reduction in creating the executable tests is at least a 12X savings over traditional approaches (one fourth the number of tests times a 3X multiple).

Test Execution

If the ambiguity reviews and modeling processes have been done properly, then less than 10% of the tests should fail on the initial execution. This is compared to over 50% in traditional testing. If it is greater than 10%, then there are generally issues around the specs being updated without the testers being informed and/or issues with the code quality and unit testing.

Test Coverage

By definition, BenderRBT generates tests giving 100% functional coverage. This is not our definition but that used in functionally testing high end integrated circuits – which very rarely have any functional defects.

The tests generated by BenderRBT generally achieve 80% to 90% code coverage as measured by statement and branch coverage. If it is less than that, there are usually numbers of functions in the code for which there were no specifications nor were they communicated to the modelers in any way. Our studies have shown that the industry average for code coverage is under 50% at the end of all of the testing. With RBT we are going into code based testing with 90% of all the functional tests we'll need already defined before coding starts.

Defect Rates

The RBT process is very focused on not just early defect detection, but defect prevention. One measure of how effectively RBT has been deployed is the percentage of defects encountered during machine based testing – unit test through full integration test – that are attributable to requirements errors. Normally over half of the defects have their root cause in requirements that are incorrect, incomplete, ambiguous, and/or logically inconsistent. This should be reduced down to fewer than 5%. In production there should be no severity one or severity two defects.

Richard Bender
BenderRBT Inc.
rbender@BenderRBT.com