

# Advanced Test Case Design

## Using Logic Modeling

### Logic Modeling Highlights

Logic modeling works with any language, any O/S and any platform. As a test design discipline, it

- Supports Test requirements and code with one practice
- Reduces development scrap and rework by up to 70%
- Cuts defects by 50% before coding
- Shortens development, design, and test time
- Increases test coverage while reducing the number of tests
- Provides new & earlier measurements of project health

Traditional test design analysis, when employed rigorously, normally achieves 40-60% coverage of business scenarios described in requirements. Most enterprises cannot even measure the complexity of business scenarios, much less the coverage afforded by their test suites. What should the measure be? To achieve test coverage in complex applications, analysts often resort to a combinatory approach in testing, where they try to test as many input combinations as they can. Although exhaustive, this tactic can raise costs dramatically while driving extensive test case duplication against logic branches. When 2% of the defects present in an application cause 1,000 times more failures than 60% of the defects, thorough and efficient testing becomes imperative to minimizing risk.

Thorough testing that attempts to address possible permutations simply is not feasible or efficient. Even good testing cannot be achieved without a complete analysis of business requirements. In turn, incomplete and ambiguous requirements represent unfinished analysis and discovery that must be addressed. This, presents a basic measurement question: how does anyone know when business analysis is complete?

When traditional test design is employed, discovery of incomplete requirements is achieved during final testing – or worse, after launch. The top reasons why software projects fail are incomplete and/or changing requirements and specifications and missing user input. The most sophisticated means of identifying incomplete requirements involves construction of software designs and then completing walkthroughs and testing. None of these methods allow a direct assessment for completion of analysis. Nor do the methods allow organizations to quantify, and thus measure, test coverage.

### The QA Labs Solution

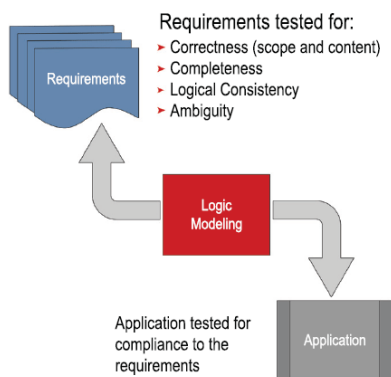


Figure 1: Logic Modeling bi-directional testing

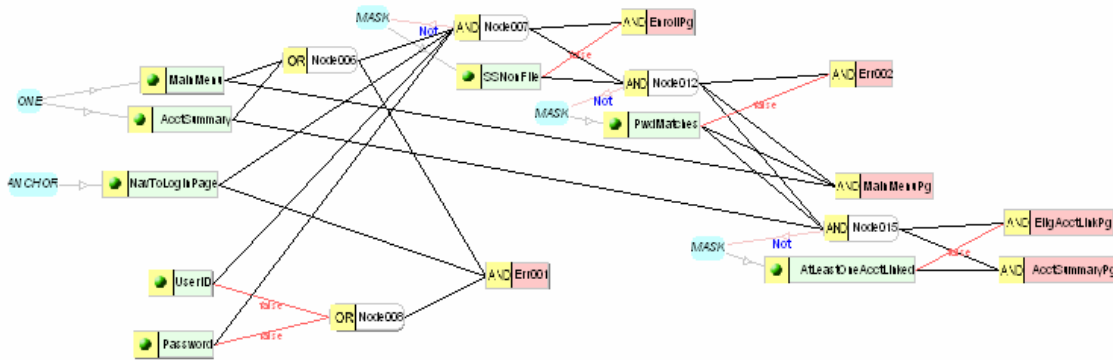
### The QA Labs Solution

QA Labs approaches the dual problems of incomplete requirements analysis and inadequate test coverage with one solution: Advanced test case design, using logic modeling. Logic modeling practices allow analysis and development teams to ascertain when analysis is complete.

Logic modeling is a bidirectional testing discipline. Construction of the model helps analysts expose gaps, inconsistencies, and ambiguous requirements before software design is complete and coding begins.

The model is used to generate tests with mathematical rigor unmatched by any other discipline. Every business path is identified and sensitized to identify the fewest number of tests that maximize the functional coverage of requirements.

## Logic Model



The logic model pictured above accounts for all cascading logic in a “Login User” use case. The model is submitted to an analysis engine that uses calculations of the “D” algorithm employed to efficiently detect faults among the millions of logic gates in an integrated circuit. Fault rates in a CPU must be extraordinarily low to zero. Failure to achieve such pristine quality has cost major chip manufacturers millions. Further, the algorithm defines the complexity (measured by the number of functional variations) of the functionality modeled, the fewest paths through the logic to achieve 100% detection of faults, and raises warnings to the analyst where logic could be faulty, inconsistent, or incomplete. When the model “works” the analyst has solid assurance that the requirements are complete and have been captured in the model.

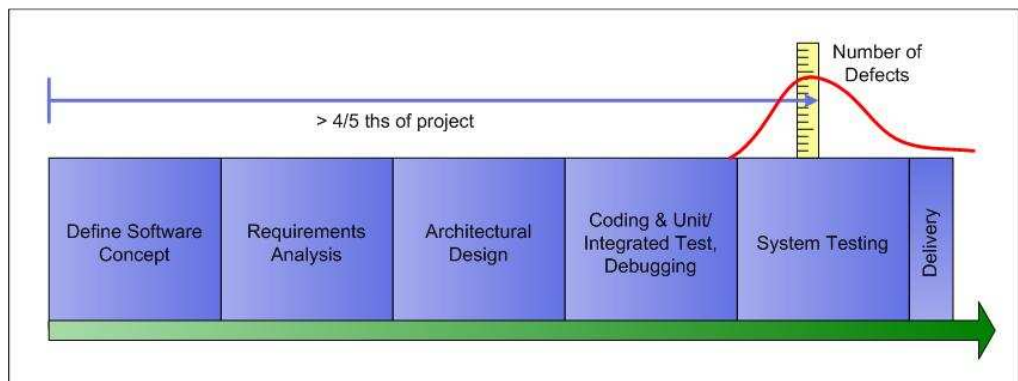
In this example, there are eight inputs. Thorough combinatory testing demands  $2^8$  or 256 test cases. Focusing testing efforts against logic gates identifies the input combinations that are material to all feasible paths. This reduces the number of test cases to 31 (the number of feasible functional variations). Thus, the same rigor can be achieved with a 8:1 reduction in tests. Accounting for path overlaps, all 31 functional variations can be tested at least once with 8 tests, representing a 32:1 reduction from combinatorial tests with no sacrifice in coverage. A typical *moderately* complex use case could demand over 100 trillion combinatory inputs. In one instance, 274+ trillion combinatory input possibilities were reduced to 40 test cases yielding 100% functional test coverage of the requirements.

Validation of analysts’ work is achieved when the test cases are used to walk though the design and requirements documents.

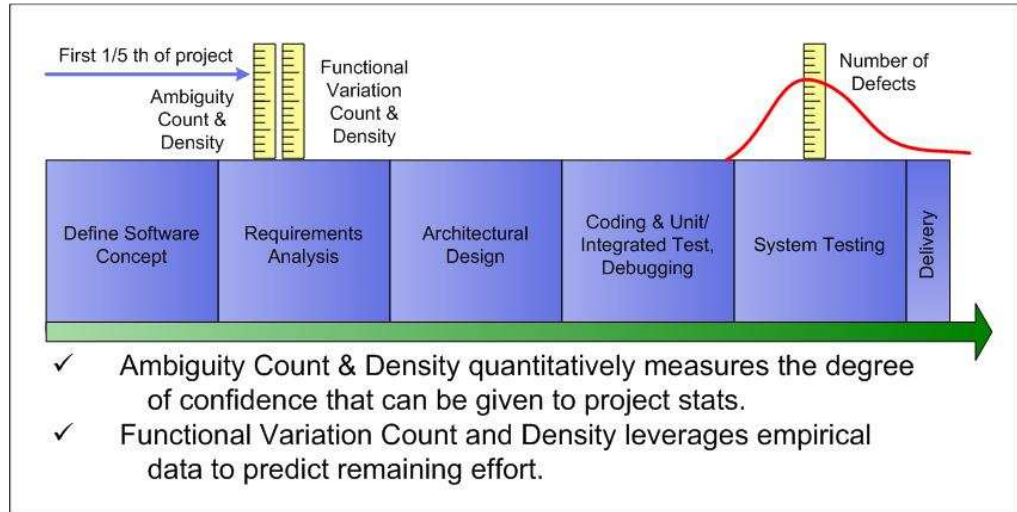
Changes to requirements are rapidly assimilated into existing models and revised test suites are re-generated, so the value of initial analysis is retained and leveraged as changes occur.

## Early Project Health Measurements

Today, you’ll find that most IT managers agree that the first solid measurement of a project’s success is in the number of defects detected during system test. Unfortunately, System Test begins roughly 80% of the way through the project. The first time defects yield a picture of project health is well into this phase, when the defect count trend defines a peak which then yields to a decline.



Since the logic model counts the number of “functional variations” (FV) in the requirement or use case to be built, the number of FVs can be used to quantify the complexity of what must be built. Thus by tracking the average density of FVs in the requirements, and understanding the effort associated with implementing those FVs, one is in the position to make better guesses as to remaining effort.



### Functional Variation (FV)

FV	A	B	C	D
0	<del>T</del>	<del>T</del>	<del>T</del>	<del>T</del>
1	F	T	T	T
2	T	F	T	T
3	T	T	F	T
4	F	F	F	F

Inputs to a logic gate must be varied in ways that expose all potential faults. In our example we cite a rule that states: If “A” or “B” or “C” is true, then “D” is set to true, else “D” is set to false. The principles of fault detection assert that the number of tests that must be conducted to validate the gate’s operation is the number of inputs (n=3) +1. You can see that the relevant tests for this gate consist of exercising and observing FVs 1-4 with FV 0 as a redundant test.

### Ambiguities

Ambiguities refer to areas of the requirement not understood, thus not consumable for logic modeling – or development. This practice measures the density of ambiguity in requirements. If a requirement has a high density of ambiguity, then any forecast based on FV density has low confidence. If ambiguity density is low, then confidence is high.

### Advanced Test Design: The ROI

The early identification of requirements faults adds quality while cutting costs and development time. The ROI includes:

- Software defects initially reduced 50%, reducing testing cycles
- Fewer tests, higher coverage
- Unbudgeted software development scrap and rework reduced by up to 70%
- No severity 1 or 2 defects in production for modules covered by Advanced Test Design
- Requirements analysis improves and becomes more efficient
- Early measurement of project health
- Can be implemented mid-project and against poor requirements
- Drives requirements detail to the point of effective testing and development
- Measures complexity, leading to improved predictability
- Compatible with traditional, UML, and lean SDLC models
- Works for software written in any language in any O/S on any platform
- Models retain the value of the analysis effort through reuse