Comparing Requirements Based Testing Techniques

Richard Bender
Bender RBT Inc.
17 Cardinale Lane
Queensbury, NY 12804
Phone: 518-743-8755
rbender@BenderRBT.com
www.BenderRBT.com

Testing By Gut Feel

Totally dependent on who is doing the testing:

- How experienced they are at testing
- How experienced they are in the technology that the application runs on
- How experienced they are in the application
- How they are feeling today

Even if all the tests run successfully, all you know is that *those* tests run -- not that the system runs successfully

Overview

- Define the criteria for comparison
- Evaluating the techniques
  - Pair Wise / Equivalence class testing
  - Path coverage through models
  - Bender RBT Process with path sensitizing via Cause-Effect Graphing

Information Needed to Design Test Cases

- Identify all of the variables
- Resolve aliases within/across processes
- Identify the possible states of the variables
  - Both positive and negative states
- Know which variables are mandatory versus optional
- Identify all of the preconditions
  - Based on the physical structure of the data
  - Based on the post conditions of prior functions

Information Needed to Design Test Cases

- Understand the precedence relationships
- Understand concurrency
- Know which variables are observable
- Identify implicit information and get it clarified
- Identify the transforms
- Identify the expected results

Test Case Design Challenges

1. Testing is comparing an expected result to the observed result – implies clear specifications

2. The number of potential tests exceeds the number of molecules in the universe

3. Did you get the right answer for the right reason

Test Case Design Challenge #1

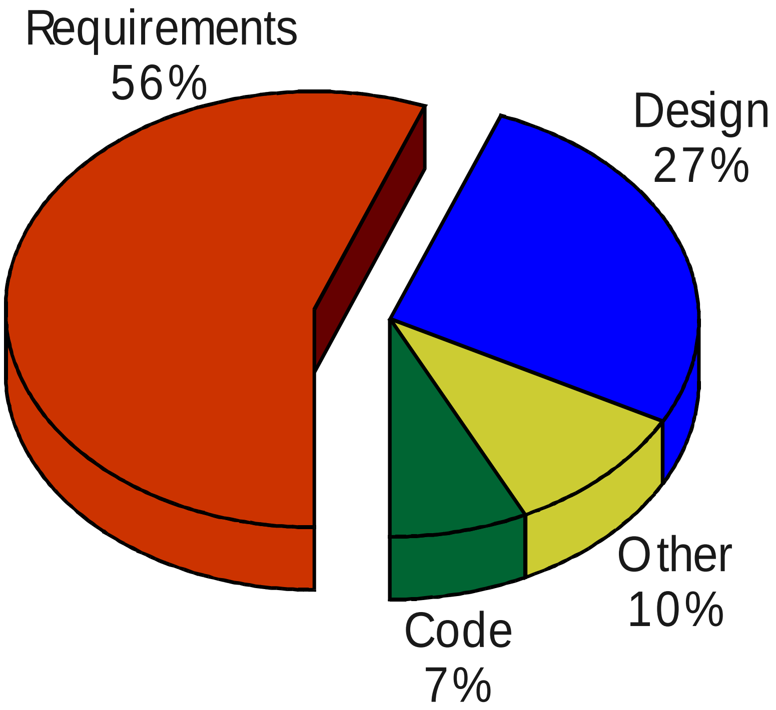
- Testing is comparing an expected result to an observed result – implies clear specifications

- Given an initial system state and a set of inputs can predict exactly what the outputs will be

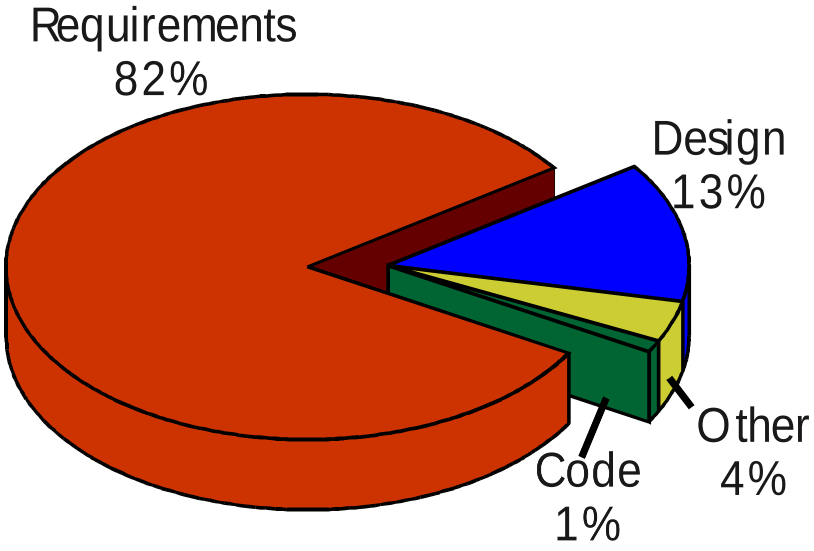
How Common Are Clear Specifications?

- Bender RBT Inc. founded in 1977
- Working with 100’s of clients and many hundreds of projects we have see just TWO testable specs going into a new client.

Distribution of Bugs



Distribution of Effort to Fix Bugs



James Martin

Ambiguous Specifications And Signoffs

A difference between Version I and Version II exists only when mixed data types are used, and then only when operand lengths differ, and then only sometimes.

Definition of Ambiguous

- If one person wrote it with one intent and another person read it differently, it is ambiguous.

Inputs to Test Design Process

Process **cannot** assume that good requirements specifications exist

- Inputs:
  - High-level requirements
  - Somewhat “detailed” design documents written in “technicaleze”
  - Screen prototypes
  - Supplemented by memos, e-mails, conversations, rumors
  - User stories in agile methodologies

Process must drive down the level of detail

Test Case Design Challenge #2

- Make the big number a small number:
  - If you have **just 6 variables** and they have only two states each and then factor in all of the unique orders then:

$$2^6! = 64! = 1.27 * 10^{89}$$

Test Case Design Challenge #3

- Did you get the right answer for the right reason
  - Two or more defects may sometimes cancel each other out
  - Something going right can hide something going wrong

Requirements Based Testing Process

- VALIDATE That The Requirements Are:
  - Correct
  - Complete
  - Unambiguous
  - Logically Consistent

- Design Sufficient Tests To VERIFY That The Design And Code Correctly Implement The Requirements

Equivalence Class Testing With Boundary Analysis

- Domain defined by range:
  - Select value in the middle
  - Select the highest valid value
  - Select the lowest valid value
  - Select something higher than the highest valid value
  - Select something lower than the lowest valid value

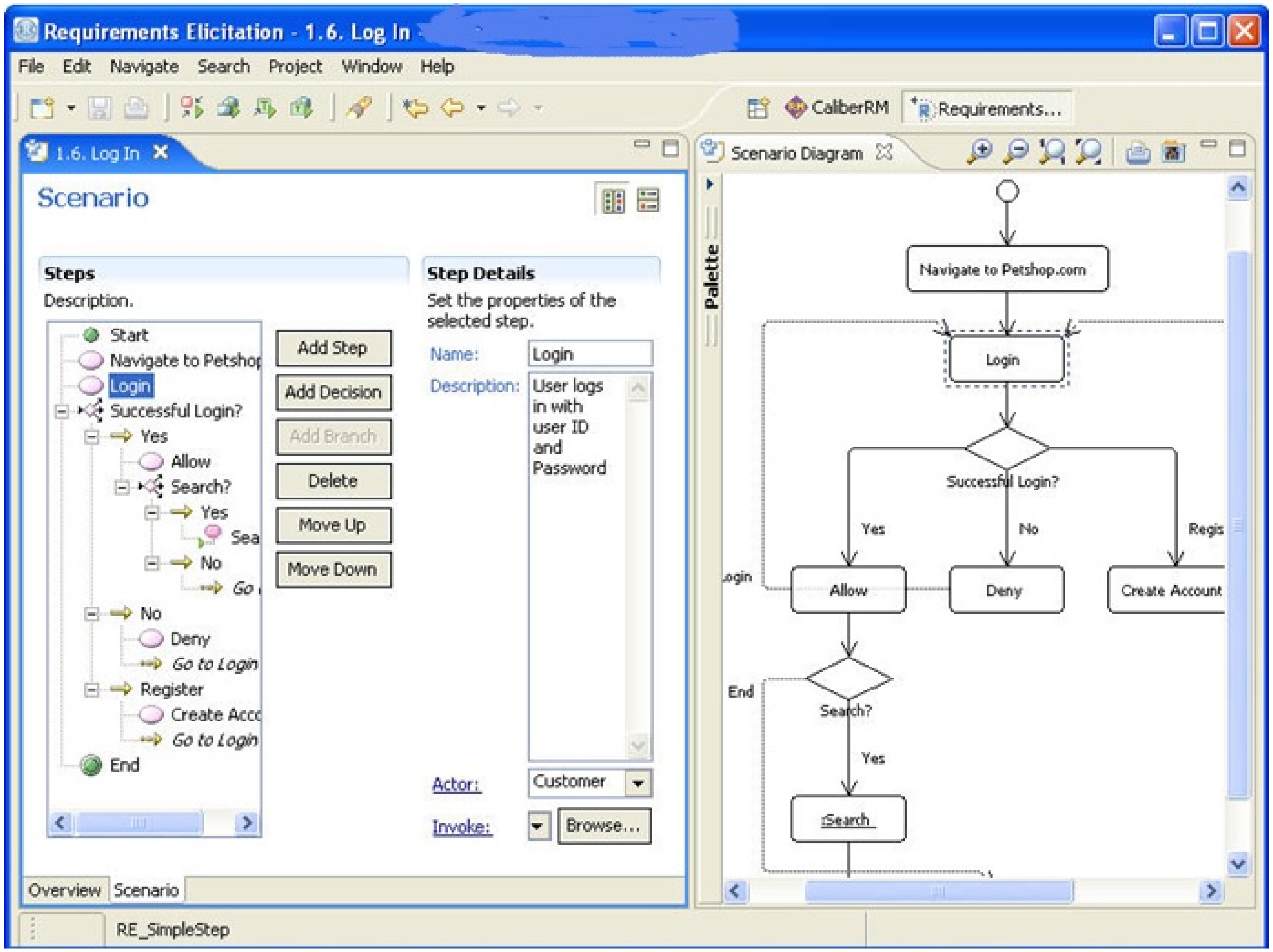
Pair Wise Testing

- Steps:
  - 1. Identify variables
  - 2. Identify states for each variable
  - 3. Identify constraints across variables/states
  - 4. Create pairs by combining all states of a variable with all states of the other variables
  - 5. Merge feasible pairs into test cases, ensuring compliance with constraints

Pair Wise Testing

- Identifies variables/states
- Weak on identifying aliases
- Precedence, concurrency not addressed
- Preconditions usually not addressed
- Expected results not identified
- Weak at clarifying specifications
- Logical consistency not validated
- Often generate illogical tests
- Does reduce the number of tests

Path Coverage Through Models



Scenario

Steps

Description.

- Start
- Navigate to Petshop
- Login
- Successful Login?
  - Yes
    - Allow
    - Search?
      - Yes
        - Sea
      - No
        - Go
  - No
    - Deny
    - *Go to Login*
  - Register
    - Create Acco
    - *Go to Login*
- End

Step Details

Set the properties of the selected step.

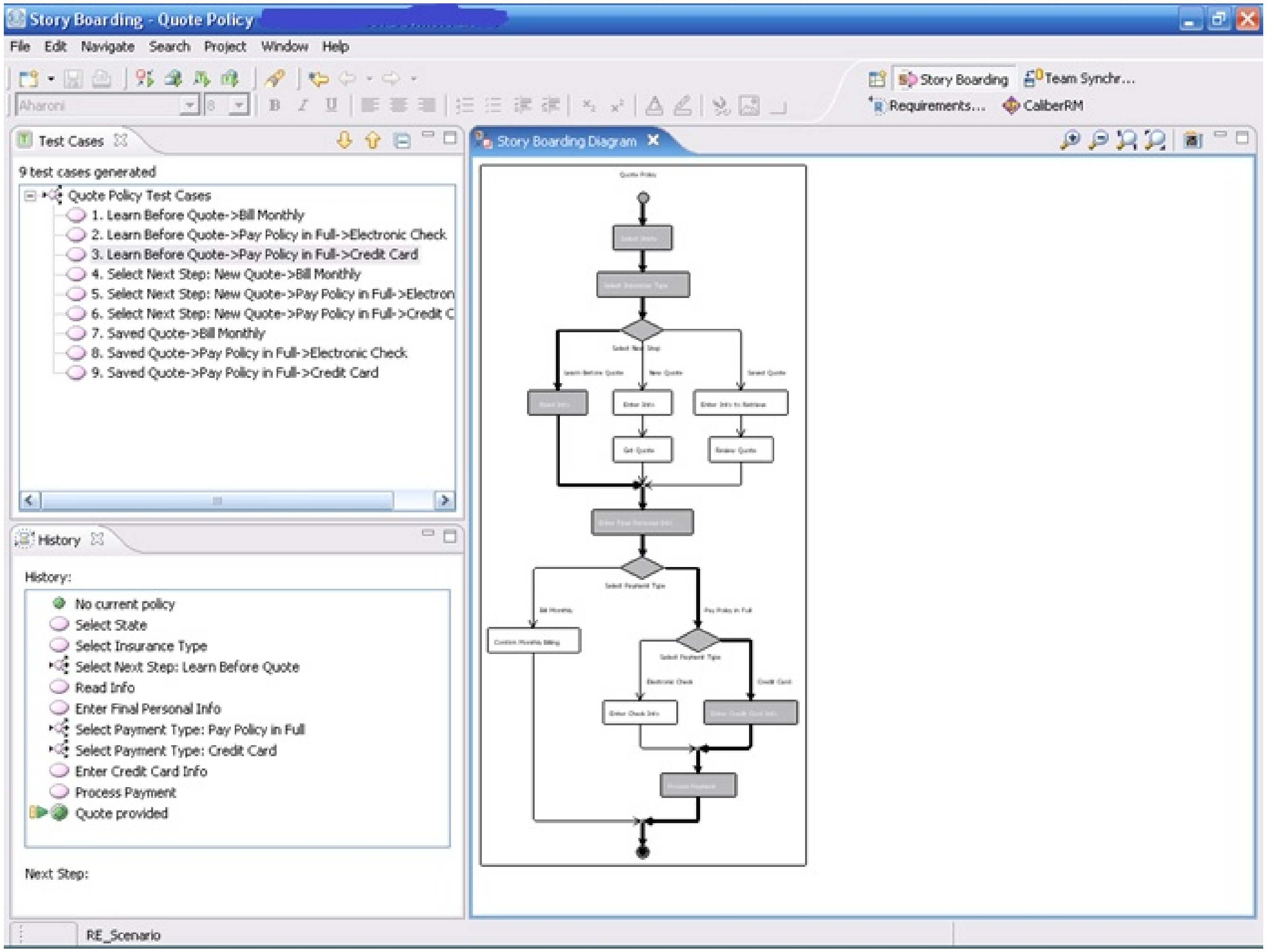
Name: Login

Description: User logs in with user ID and Password

Actor: Customer

Invoke:

Path Coverage Through Models



9 test cases generated

- Quote Policy Test Cases
  - 1. Learn Before Quote->Bill Monthly
  - 2. Learn Before Quote->Pay Policy in Full->Electronic Check
  - 3. Learn Before Quote->Pay Policy in Full->Credit Card
  - 4. Select Next Step: New Quote->Bill Monthly
  - 5. Select Next Step: New Quote->Pay Policy in Full->Electron
  - 6. Select Next Step: New Quote->Pay Policy in Full->Credit C
  - 7. Saved Quote->Bill Monthly
  - 8. Saved Quote->Pay Policy in Full->Electronic Check
  - 9. Saved Quote->Pay Policy in Full->Credit Card

History:

- No current policy
- Select State
- Select Insurance Type
- Select Next Step: Learn Before Quote
- Read Info
- Enter Final Personal Info
- Select Payment Type: Pay Policy in Full
- Select Payment Type: Credit Card
- Enter Credit Card Info
- Process Payment
- Quote provided

Next Step:

Path Coverage Through Models

- Specifications must all be in the requirements component of the tool
- They must all be machine readable/parsable
- Our experience is that the set of requirements is in multiple formats in multiple documents
- The vast majority are not machine parsable
  - MS Word, Excel, Visio

Path Coverage Through Models

- Does factor in precedence
- Does not factor in concurrency
- Usually does not include the expected results
- Does not factor in preconditions
- Some can identify intra-functional logical inconsistencies
- Often generate illogical tests
- Does not aid in clarifying the specifications
- Does reduce the number of tests

Bender RBT Process

Quality filters

1. Validate requirements (WHAT) against objectives (WHY)
2. Apply scenarios against requirements / use cases
3. Perform initial ambiguity review
4. Perform domain expert reviews
5. Create cause-effect graph
6. Logical consistency check by BenderRBT
7. Validate test cases with specification writer
8. Validate test cases with users/domain experts
9. Validate test cases with developers
10. Verify design via walking test cases through design
11. Verify code via walking test cases through code
12. Verify code via executing test cases against code

Ambiguity Review Checklist

- Dangling else
- Ambiguity of reference
- Scope of action
- Omissions
  - Causes without effects
  - Missing effects
  - Effects without causes
  - Complete omissions
  - Missing causes
- Ambiguous logical operators
  - Or, And, Nor, Nand
  - Implicit connectors
  - Compound operators
- Negation
  - Scope of negation
  - Unnecessary negation
  - Double negation
- Ambiguous statements
  - Verbs, adverbs, adjectives
  - Variables, unnecessary aliases
- Random organization
  - Mixed causes and effects
  - Random case sequence
- Built-in assumptions
  - Functional/environmental knowledge
- Ambiguous precedence relationships
- Implicit cases
- Etc.
- I.E. versus E.G.
- Temporal ambiguity
- Boundary ambiguity

Dangling Else

Must be, will be, is one of, should be, could be.

Example:

“The code must be either A, B, or C.”

Else? An error condition?

Benefits from Ambiguity Reviews

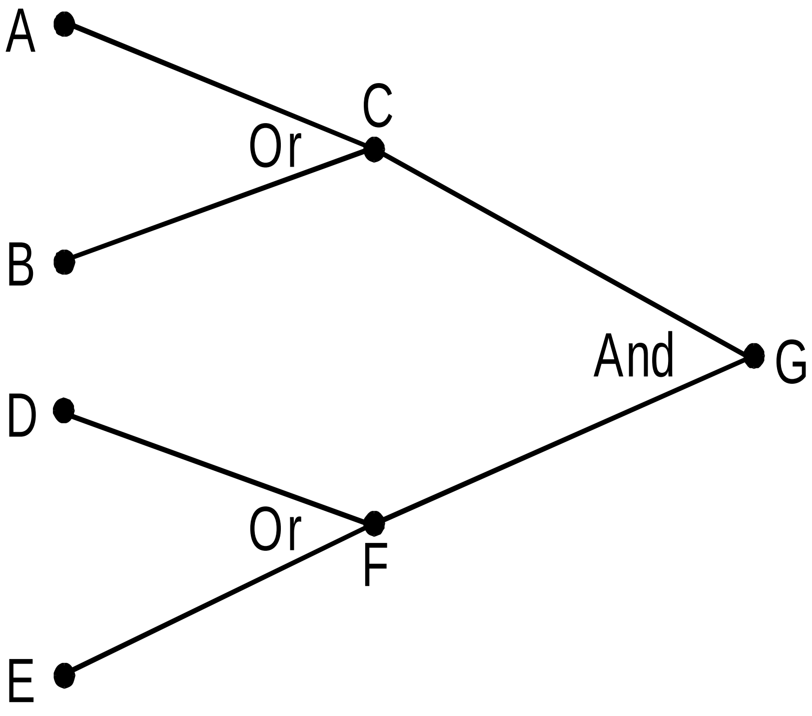
- Timely feedback reduces issue resolution time.
- Explicit feedback leads to defect avoidance – 95% reduction.
- Critical to outsourcing.

If something is ambiguous in the specs it will nearly always result in a defect(s) in the code

Cause-Effect Graphing

1. If A or B, then C.
2. If D or E, then F.
3. If C and F, then G.

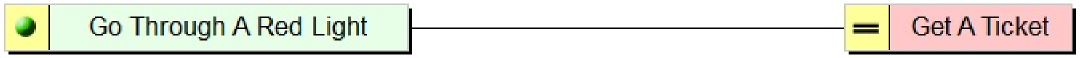
- Resolve Aliases
- Clarify Precedence Rules
- Clarifies Implicit Information



Cause-Effect Graphing

- Independent of the format of the requirements
- Can support agile projects
- Identifies variables, states, aliases
- Clarifies precedence, concurrency
- Factors in preconditions
- Identifies expected results
- Clarifies implicit results

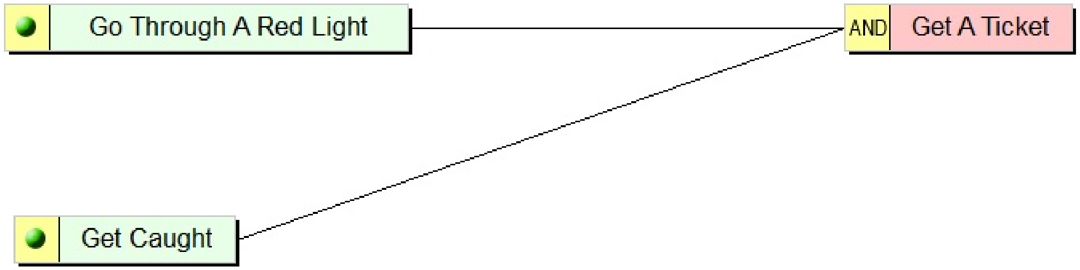
Clarifying Requirements Via Cause-Effect Graphing



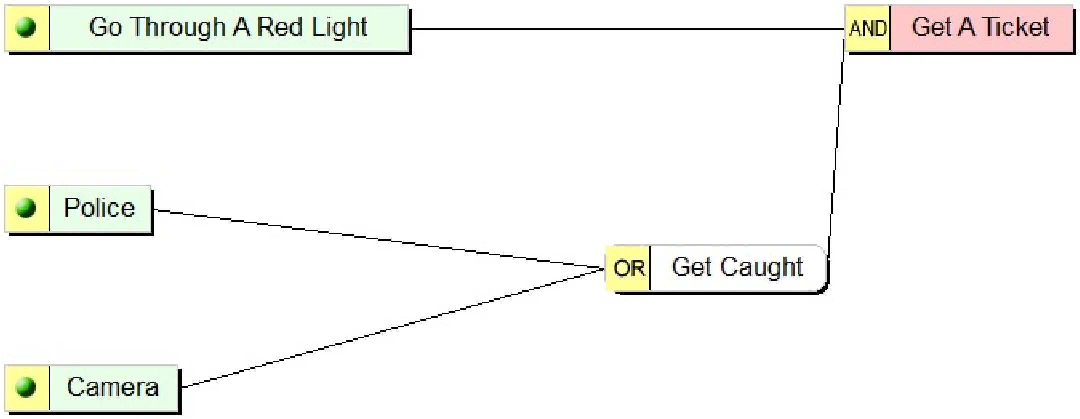
● Go Through A Red Light

═ Get A Ticket

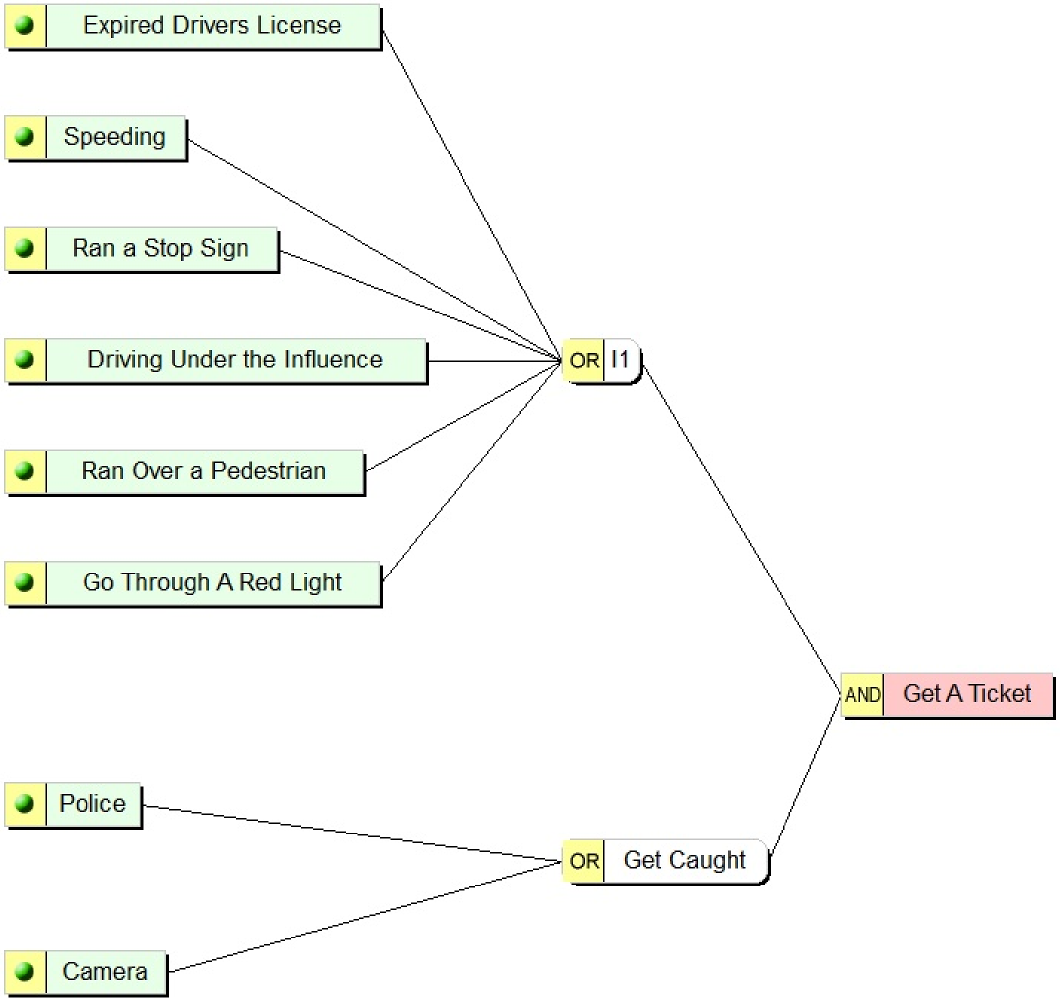
Clarifying Requirements Via Cause-Effect Graphing



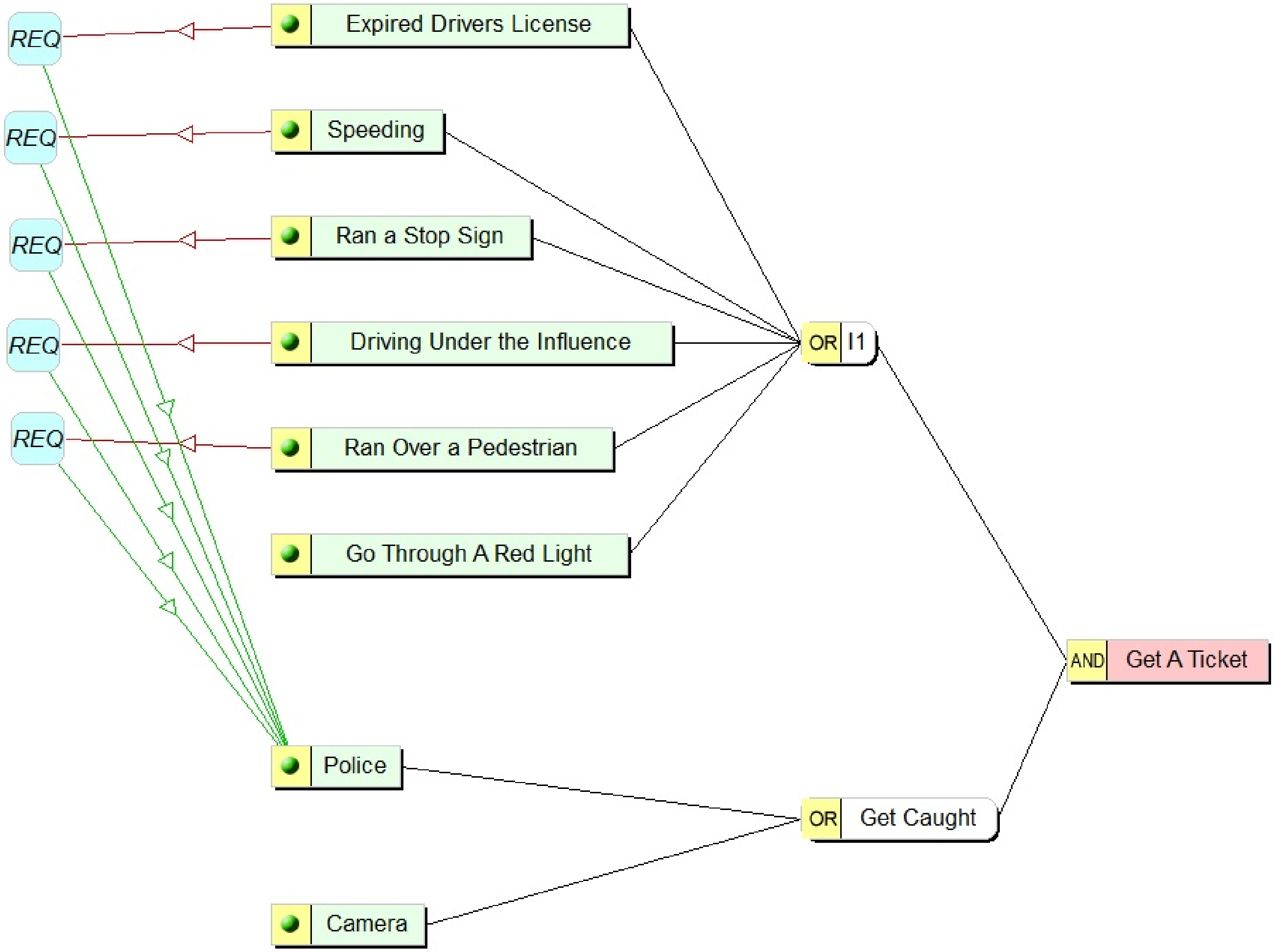
Clarifying Requirements Via Cause-Effect Graphing



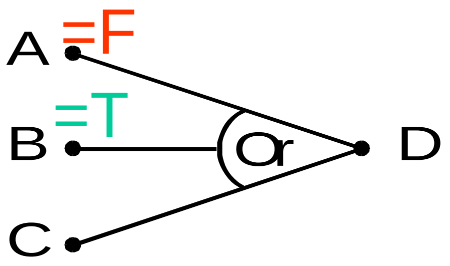
Clarifying Requirements Via Cause-Effect Graphing



Clarifying Requirements Via Cause-Effect Graphing



Cause-Effect Graphing

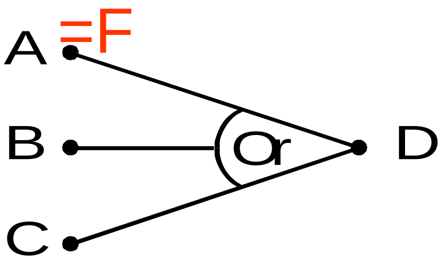


1.	A	—	—	D
2.	—	B	—	D
3.	—	—	C	D
4.	—	—	—	—

Assume A is stuck at FALSE and B is stuck at TRUE.
The machine would interpret:

1.	A	—	—	as	—	B	—	D
2.	—	B	—	as	—	B	—	D
3.	—	—	C	as	—	B	C	D
~~4.~~	—	—	—	as	—	B	—	Ⓓ

Cause-Effect Graphing



1. A — — | D
2. — B — | D
3. — — C | D
4. — — — | —

Assume A is still stuck at FALSE.
The machine would interpret:

~~1~~. A — — as — — — | ⊖
2. — B — as — B — | D
3. — — C as — — C | D
4. — — — as — — — | —

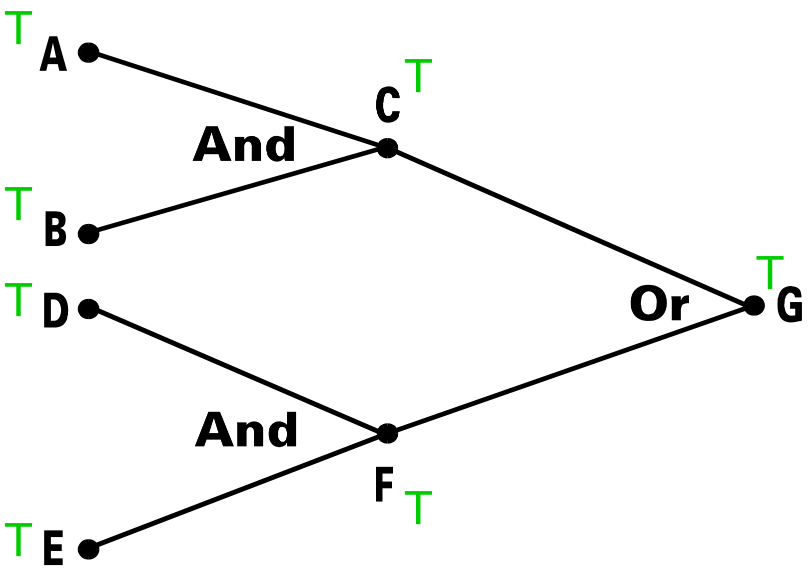
Fix the bug found by #4 and #1 fails.

Must rerun ALL tests until ALL pass!

Cause-Effect Graphing

Observable Events and Path Sensitizing

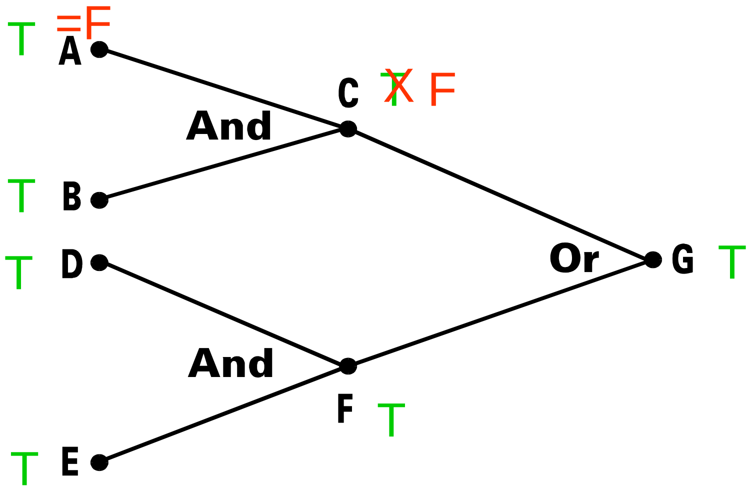
- Assume C and F are not observable events.
- Assume A is stuck at FALSE.
- Enter as a test case A(T), B(T), D(T), E(T).
- Results should be C(T), F(T) and G(T).



Cause-Effect Graphing

Observable Events and Path Sensitizing

- Results should be C(T), F(T) and G(T).
- A, stuck at FALSE, causes C to be (F).
- The error is not detected since G is still (T) due to F(T).
- Therefore, no test of C can be combined with tests of F which would result in F(T).



Cause-Effect Graphing

Observable Events and Path Sensitizing

Challenge:

- Design a set of test cases, factoring in:
  - The relations between the variables
  - Constraints between the data attributes
  - Functional variations required to test
  - Node observability

… such that if any logical defect or any combination of defects are present, at least one test case will fail at an observable point.

Cause-Effect Graphing

- Highly optimized test design since based on the hardware path sensitizing algorithms
- Generally results in test libraries reduced by a factor of four for equivalent coverage
- Results in significantly reduced effort to:
  - Build the executable tests
  - Run the tests
  - Verify the test results
  - Maintain the test libraries

Test Statistics For A Typical Screen

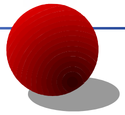
For n = 37 Primary causes, then
2^n = [a little more than] 137,438,953,472
THEORETICAL Maximum Number of Test Cases.

RBT generated 22 Test Cases, which yields a
6,247,225,157 to 1 Test Case Compression
Ratio.

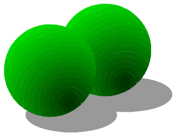
RBT Elapsed Time: 00:00:01 (hh:mm:ss)

Test Statistics

Thought Experiment



- Put 137,438,953,450 red balls in a giant barrel.
- Add 22 green balls to the barrel and mix well.
- Turn out the lights.
- Pull out 22 balls.



What is the probability that you have selected the 22 green ones?

- Pull out 1,000 balls

What is the probability that you have the 22 green ones now?

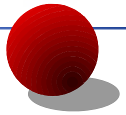
- Pull out 1,000,000 balls

What is the probability that you have the 22 green ones now?

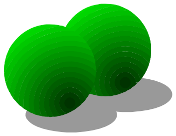
This is what “GUT FEEL” testing really is.

Test Statistics

Thought Experiment



- Put 137,438,953,450 red balls in a giant barrel.
- Add 22 green balls to the barrel and mix well.
- Turn out the lights.
- Pull out 22 balls.



What is the probability that you have selected the 22 green ones?

- Pull out 1,000 balls $7.3X10^{-180}$

What is the probability that you have the 22 green ones now?

- Pull out 1,000,000 balls $9.2X10^{-114}$

What is the probability that you have the 22 green ones now?

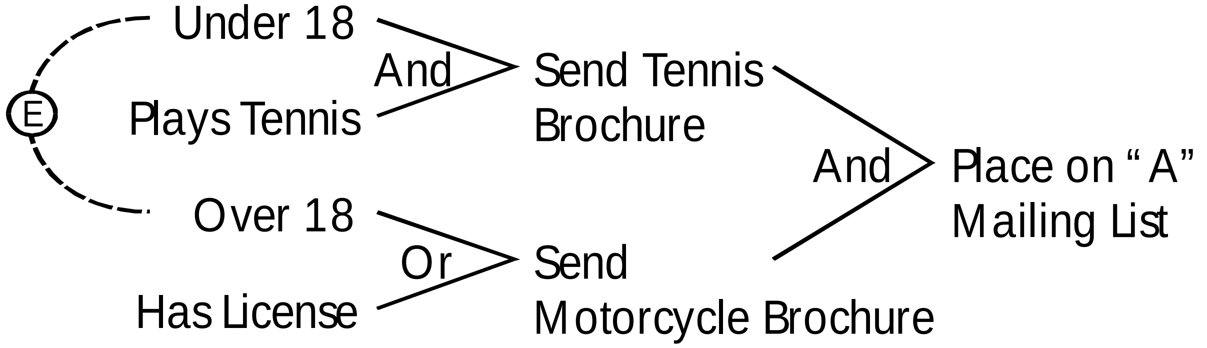
This is what “GUT FEEL” testing really is.

C-E Graphing Validates the Logical Consistency

If the person is under 18, and plays tennis,
then send them a tennis club brochure.

If the person is 18 or older, or has a motorcycle license,
then send them a motorcycle club brochure.

If the person was sent both brochures, then put them
on the “A” mailing list.



E

Under 18

Plays Tennis

And

Send Tennis Brochure

Over 18

Has License

Or

Send Motorcycle Brochure

And

Place on “A” Mailing List

You must be over 18 to have a motorcycle license.
[Has License(T) requires Over 18(T)]

C-E Graphing Validates the Logical Consistency

```
 Functional Variations for:
A_list:-Tennis-brochure AND Motorcycle-brochure.
<INFEASIBLE> T01--Due to constraint(s) ACROSS
   relationships (or faulty logic)
       7. If Tennis-brochure and Motorcycle-brochure
          then A_list.

       8. If not Tennis-brochure
            (and Motorcycle-brochure)
          then not A_list.

       9. If not Motorcycle-brochure
            (and Tennis-brochure)
          then not A_list.

T11--Possible graph logic error.  TRUE state of A_list
   always Infeasible
T12--Note:  TRUE state of A_list not covered in any test
   case
```

Tests From C-E Graphing are Functionally Equivalent to the Rules in the Specs

Original Requirement

Dental Insurance Claims Payment Specification

Dentists with membership codes of 2, 3, or 9 are member dentists. For claims referencing a non-member dentist or for procedures not within the referenced dentist’s record, a system table is used to calculate the amount paid. Otherwise, the amount submitted is paid. However, an override code of 1 or 9 allows the amount submitted to be paid for non-member dentists or for procedures not within the referenced dentist’s record. When an override code is used an entry is made on the paid claims report.

C-E Graph Generated Tests

(not the full set)

TEST 1

Cause States:

The Dentist is a Member Dentist

The procedure was not preauthorized

An override code was entered

Effect States:

Override the partial payment

Make an entry on the paid claims report

TEST 2

Cause States:

The Dentist is a Member Dentist

The procedure was preauthorized

Effect States:

Pay the full amount of the claim

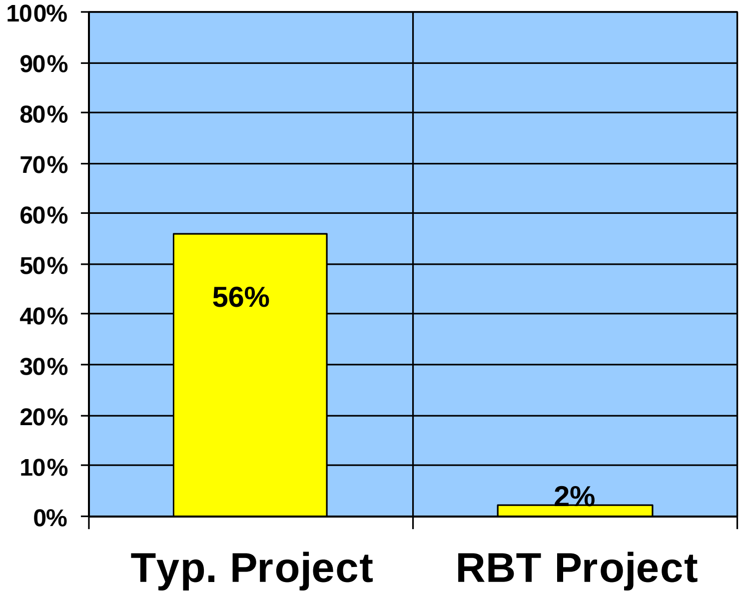
Do not make an entry on the paid claims report

Test Case Reviews

- Moves User Acceptance Test up before coding starts
- 90% of all tests needed defined before start of coding

Eliminate Requirements Defects

Percentage of Requirements Based Defects Found
From Unit Test Through Deployment



Test Design Summary

Validate Requirements	Cause-Effect Graphing	Path Coverage	Pair-Wise
Flexible Requirements Format	X		X
Ambiguity Eliminated	X	*	*
Implicit Requirements Clarified	X		
Sequencing Clarified	X	X	
Concurrency Clarified	X		
Logical Relationships Clarified	X	x	
Logical Consistency Verified	X	x	

Test Design Summary

Test Design	Cause-Effect Graphing	Path Coverage	Pair-Wise
Expected Results Included	X	x	
Boundary Constraints Factored In	X		x
Observability of Defects	X		
Reduce Number of Tests	X	x	x
Test Coverage	100%	<50%	<50%
Can Support Agile Projects	X		X