PRESENTATION #

6

TUESDAY, MAY 10
4:30

EFFECTIVE TESTING OF FACTOR COMBINATIONS

GEORGE SHERWOOD

AT&T

Third International Conference on
Software Testing, Analysis & Review

May 8–12, 1994 • Washington, DC

GEORGE B. SHERWOOD

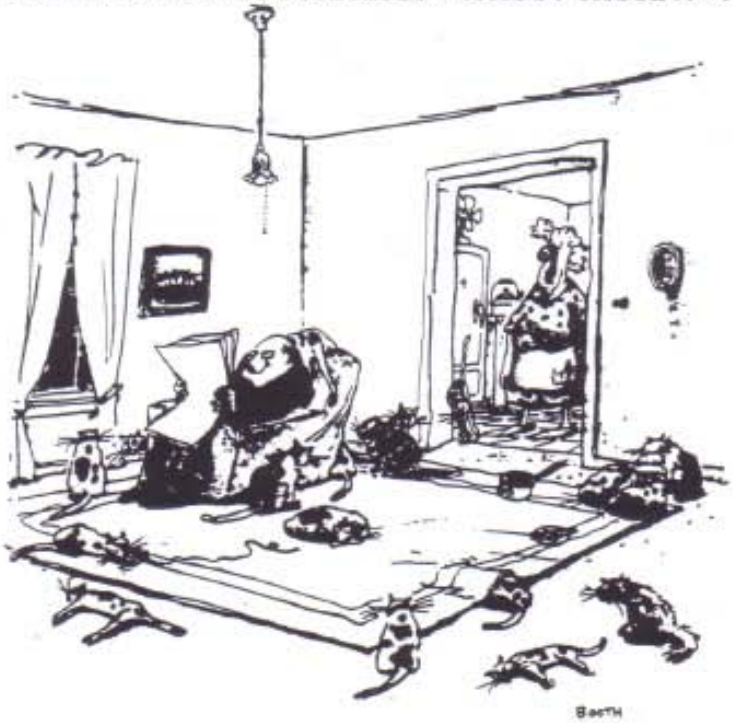
George Sherwood is a member of the Test Management and Tool Development Team at AT&T PersonaLink Services. His organization is developing a new consumer messaging service. He has worked at AT&T Bell Laboratories since 1978 on various software and hardware development projects and has developed a variety of tools to facilitate software development work. George prototyped CATS — the Constrained Array Test System — in early 1990. Since then, its use has spread throughout the AT&T technical community. George received his BS in Physics from Clemson in 1972, and his MPhil and PhD, also in Physics, from Yale in 1974 and 1978.

EFFECTIVE TESTING

OF

FACTOR COMBINATIONS

THE SYSTEM TEST DILEMMA: SURPRISES WITHOUT THOSE AWFUL COMBINATIONS



"Edgar, please run down to the shopping center right away, and get some milk and cat food. Don't get canned tuna, or chicken, or liver, or any of those awful combinations. Shop around and get a surprise. The pussies like surprises."

Drawing by Booth: © 1977 The New Yorker Magazine, Inc.

Effective Testing of Factor Combinations:

- Problem Statement and Definitions
- Features of the Constrained Array Test System (CATS)
- Example -- Generating New Test Cases
- How CATS Works
- Example -- Supplementing Test Cases
- CATS Status

The Problem: How much Testing is Enough?

- Code Coverage Tools
  - — Good for Unit and Subsystem Testing
  - — Not Practical for Large or Mixed Systems
- Test all Transitions of Finite State Model
  - — Useful for Subsystem Testing, Communication Protocol Implementations for Example
  - — FSM Analysis and Test Not Practical for Complex Systems
- Orthogonal Testing Designs
  - — Good for Getting "Small" Number of Test Cases
  - — Difficult with Constraints among Test Factors

What is a test factor?

A test factor is any variable whose values should be controlled during the test process:

- Environmental conditions
- Characteristics of a larger system of which the test system is a part
- Processor speed
- System load characteristics
- Modes of operation and options
- Communication configuration
- Characteristics of users and user interfaces
- Command line arguments
- Input record contents
- Mouse coordinates

What is a test case?

A test case is a set of specific test factor values in which one allowed value is associated with each of the test factors. The set of factor values represents an individual test from a sequence to be performed.

What is a combination of factor values?

A combination of test factor values is an association of some number of factor values. *operating_system=SVR4* with *arguments=-l* is a combination of two factor values which could occur in a test case.

How do you select factors and their values?

CATS does not answer this question.

Selection can benefit from

- **thorough analysis and understanding of the system and its requirements,**
- **experience from testing similar products or previous releases, and**
- **suggestions from the system developers.**

What is CATS?

The Constrained Array Test System is a system *test tool* **which** *generates and analyzes* **test cases.**

Using CATS to select which test cases to execute yields

- *Savings in testing time* **due to a reduced number of test cases, and**
- *Improved ability to find faults* **because of better coverage of combinations of test factor values.**

CATS provides a set of objective *metrics for measuring test coverage.*

What are CATS' Features?

- **Small Number of Test Cases**
- **Capacity for Large Numbers of Factors and Values**
- **Conformance to Test Case Constraints**
- **Ordering of Test Case Sequence**
- **Interfaces for Factor Values and for Test Cases**
- **Detection of Higher Order Mode Failures**

Lunch Development Department Menu (Requirements)

- **Lunch Platform --**
  rye bread, white bread, wheat bread, macaroni, spaghetti
- **Topping Application --**
  cheese, peanut butter, jelly, clam sauce, tomato sauce
- **Beverage Program --**
  Coke, milk, coffee, tea, Chianti

Rules or Constraints:

1. Bread cannot be topped with clam sauce or tomato sauce.
2. Peanut butter is not supported on wheat bread.
3. Macaroni and spaghetti are topped only with cheese, clam sauce or tomato sauce.
4. Coke and tea are not allowed with pasta.
5. Chianti is not served with sandwiches.

Example -- Generating New Test Cases -- Factor Values input for expand

cat lunch.exp

```
rye       white
cheese    pnutbtr   jelly
coke      milk      coffee    tea
append
wheat
cheese    jelly
coke      milk      coffee    tea
append
macaron   spaghet
clams     cheese    tomato
milk      coffee    chianti
```

Expand Command Line Options

-c causes **expand** to generate test cases only; there is no analysis of the test cases when the **-c** option is used. By default (without the **-c** option) **expand** executes **cats**, and the test case analysis is performed.

-i causes **expand** and **cats** to iterate, to analyze problems with too many test cases for **cats** to process simultaneously. When the **-i** option is used, **expand** starts with a subset of the test factors, and **cats** analyzes their test cases. Then **expand** removes the unnecessary test cases and generates new test cases with one or more additional factors. This process continues until all the factors are included.

The remaining options specify the number of factors to use in the analysis of combinations. **-S -D -T** or **-Q** causes **expand** to analyze factors one-, two-, three- or four-at-a-time respectively -- to test for single-mode, double-mode, triple-mode or quadruple-mode failures. The option **-s -d -t** or **-q** causes **expand** to analyze factors for all modes of failure up to one-, two-, three- or four-at-a-time respectively. **-z** leaves the mode unspecified, so **cats** analyzes with as many factors as possible.

Analysis Options for Expand		
Option	**Number(s) of Factors**	**Failure Mode(s)**
-S	1	Single
-D	2	Double
-T	3	Triple
-Q	4	Quadruple
-s	1	Single
-d	1 and 2	Single and Double
-t	1, 2 and 3	Single, Double and Triple
-q	1, 2, 3 and 4	Single, Double, Triple and Quadruple
-z	As many as possible	As many as possible

Example -- Generating New Test Cases -- Test Case input for cats

expand -c -z < lunch.exp

0	5	5	5
1	rye	cheese	coke
2	rye	cheese	milk
3	rye	cheese	coffee
4	rye	cheese	tea
5	rye	pnutbtr	coke
6	rye	pnutbtr	milk
7	rye	pnutbtr	coffee
8	rye	pnutbtr	tea
9	rye	jelly	coke
10	rye	jelly	milk
11	rye	jelly	coffee
12	rye	jelly	tea
13	white	cheese	coke
14	white	cheese	milk
15	white	cheese	coffee
16	white	cheese	tea
17	white	pnutbtr	coke
18	white	pnutbtr	milk
19	white	pnutbtr	coffee
20	white	pnutbtr	tea
21	white	jelly	coke
22	white	jelly	milk
23	white	jelly	coffee
24	white	jelly	tea
25	wheat	cheese	coke
26	wheat	cheese	milk
27	wheat	cheese	coffee
28	wheat	cheese	tea
29	wheat	jelly	coke
30	wheat	jelly	milk
31	wheat	jelly	coffee
32	wheat	jelly	tea
33	macaron	clams	milk
34	macaron	clams	coffee
35	macaron	clams	chianti
36	macaron	cheese	milk
37	macaron	cheese	coffee
38	macaron	cheese	chianti
39	macaron	tomato	milk
40	macaron	tomato	coffee
41	macaron	tomato	chianti
42	spaghet	clams	milk
43	spaghet	clams	coffee
44	spaghet	clams	chianti
45	spaghet	cheese	milk
46	spaghet	cheese	coffee
47	spaghet	cheese	chianti
48	spaghet	tomato	milk
49	spaghet	tomato	coffee
50	spaghet	tomato	chianti

Example -- Generating New Test Cases -- Analyzed Test Case output

expand -z < lunch.exp

0	5	5	5	1:15	2:75	3:125	215
1	rye	cheese	coke	1:12	2:72	3:124	208
2	white	pnutbtr	milk	1:9	2:69	3:123	201
3	wheat	jelly	coffee	1:6	2:66	3:122	194
4	macaron	clams	chianti	1:3	2:63	3:121	187
5	spaghet	tomato	milk	1:1	2:60	3:120	181
6	rye	pnutbtr	tea	1:0	2:57	3:119	176
7	rye	jelly	milk	1:0	2:54	3:118	172
8	white	cheese	coffee	1:0	2:51	3:117	168
9	white	jelly	coke	1:0	2:48	3:116	164
10	wheat	cheese	milk	1:0	2:45	3:115	160
11	macaron	tomato	coffee	1:0	2:42	3:114	156
12	spaghet	clams	coffee	1:0	2:39	3:113	152
13	spaghet	cheese	chianti	1:0	2:36	3:112	148
14	rye	pnutbtr	coffee	1:0	2:34	3:111	145
15	white	cheese	tea	1:0	2:32	3:110	142
16	wheat	jelly	tea	1:0	2:30	3:109	139
17	macaron	clams	milk	1:0	2:28	3:108	136
18	rye	pnutbtr	coke	1:0	2:27	3:107	134
19	wheat	cheese	coke	1:0	2:26	3:106	132
20	macaron	cheese	milk	1:0	2:25	3:105	130
21	macaron	tomato	chianti	1:0	2:24	3:104	128
22	rye	cheese	milk	1:0	2:24	3:103	127
23	rye	cheese	coffee	1:0	2:24	3:102	126
24	rye	cheese	tea	1:0	2:24	3:101	125
25	rye	pnutbtr	milk	1:0	2:24	3:100	124
26	rye	jelly	coke	1:0	2:24	3:99	123
27	rye	jelly	coffee	1:0	2:24	3:98	122
28	rye	jelly	tea	1:0	2:24	3:97	121
29	white	cheese	coke	1:0	2:24	3:96	120
30	white	cheese	milk	1:0	2:24	3:95	119
31	white	pnutbtr	coke	1:0	2:24	3:94	118
32	white	pnutbtr	coffee	1:0	2:24	3:93	117
33	white	pnutbtr	tea	1:0	2:24	3:92	116
34	white	jelly	milk	1:0	2:24	3:91	115
35	white	jelly	coffee	1:0	2:24	3:90	114
36	white	jelly	tea	1:0	2:24	3:89	113
37	wheat	cheese	coffee	1:0	2:24	3:88	112
38	wheat	cheese	tea	1:0	2:24	3:87	111
39	wheat	jelly	coke	1:0	2:24	3:86	110
40	wheat	jelly	milk	1:0	2:24	3:85	109
41	macaron	clams	coffee	1:0	2:24	3:84	108
42	macaron	cheese	coffee	1:0	2:24	3:83	107
43	macaron	cheese	chianti	1:0	2:24	3:82	106
44	macaron	tomato	milk	1:0	2:24	3:81	105
45	spaghet	clams	milk	1:0	2:24	3:80	104
46	spaghet	clams	chianti	1:0	2:24	3:79	103
47	spaghet	cheese	milk	1:0	2:24	3:78	102
48	spaghet	cheese	coffee	1:0	2:24	3:77	101
49	spaghet	tomato	coffee	1:0	2:24	3:76	100
50	spaghet	tomato	chianti	1:0	2:24	3:75	99

Example -- Generating New Test Cases -- Conclusion

CATS suggests

- 6 test cases to cover all factor values individually, or
- 21 test cases to cover all *allowed* factor value pairs, or
- 50 test cases to cover all *allowed* factor value triples.

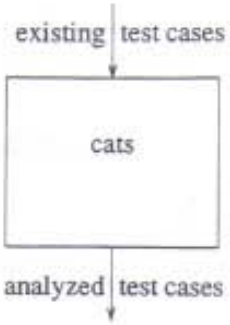
Jack selects the first 21 test cases. He will test all the allowed factor values individually and in pairs. There are 125 – 75 = 50 allowed factor value triples. Of these, Jack will cover 125 – 104 = 21 combinations, for 21/50 = 42% coverage of the allowed three-at-a-time combinations.

There are 24 out of 75 pairwise combinations and 75 out of 125 three-at-a-time combinations which are not covered due to constraints imposed in the problem.

Beginnings of CATS

- Motivated by Client Installation Testing for a Local Area Network Product
  - — 101 Combinations of Personal Computers and Operating Systems
  - — 15 Network Cards
- Orthogonal Arrays Yielded Impossible Test Cases
  - — 8088 Processor with Micro Channel Bus
  - — 80386 Processor with DOS 3.1
- **cats.c** Written to Analyze Test Coverage
  - — Input: Test Cases
  - — Output: Analyzed Test Cases
- **expand** Preprocessor Created for Enumeration of Test Cases
  - — Input: Factor Values
  - — Output: Test Cases
- **expand** Evolved to Include Control for Iteration Option, which Allows Solutions for Large Test Jobs

Normal Operation of Cats

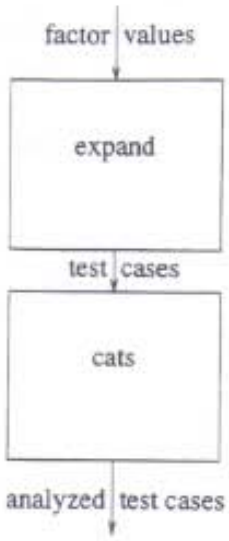


Reordering Test Cases with Cats Arrays

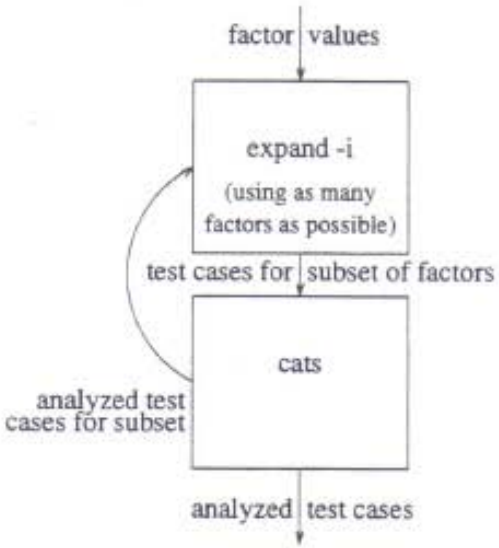
Vector Array	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	

Experiment Array	

Normal Operation of Expand (Simplified)



Iteration Option of Expand (Simplified)



Example -- Analyzing Existing Test Cases -- Test Case input for cats

Pat is testing the interaction between two client modules contending for a server module resource. There are four relevant factors in her test program:

1. **server attribute x**, with five values,
2. **server attribute y**, with four values,
3. **first client request**, with three values, and
4. **second client request**, with three values.

Currently Pat is using twenty test cases which she would like to analyze, for pairwise coverage of factor value combinations.

Pat's input file, **contend.cats**, is shown below.

cat contend.cats

```
-2  5  4  3  3
A   1  1  a  a
B   1  2  a  b
C   1  3  a  c
D   1  4  b  a
E   2  1  b  b
F   2  2  b  c
G   2  3  c  a
H   2  4  c  b
I   3  1  c  c
J   3  2  a  a
K   3  3  a  b
L   3  4  a  c
M   4  1  b  a
N   4  2  b  b
O   4  3  b  c
P   4  4  c  a
Q   5  1  c  b
R   5  2  c  c
S   5  3  a  a
T   5  4  a  b
```

Using Cats The **cats** program takes test cases as input and analyzes them to minimize the number of untested combinations of test factor values. **Cats** reorders the test cases into a test execution sequence in which each case selected is the next "best" test case to perform. Here, "best" means the test case which when executed will leave the fewest number of combinations untested.

Cats reads an array of factor values representing test cases from the standard input. Combinations of the factor values can be analyzed three ways:

1. The *number* of factors to consider at a time can be specified by starting the zeroth row of input data with the negative of the number of factors to consider. For example, for *pairs* of factors the initial string is -2.
2. The *maximum* number of factors to consider at a time can be specified by starting the zeroth row of data with the maximum number of factors to consider. For example, for factor values to be considered one-at-a-time and two-at-a-time and three-at-a-time, the initial string is 3.
3. By *default* **cats** considers as many factors as possible, limited by the number of factors in the data or by computing resources. In this case, the initial string is 0.

Analysis Options for Cats		
Initial String	**Number(s) of Factors**	**Failure Mode(s)**
-1	1	Single
-2	2	Double
-3	3	Triple
-4	4	Quadruple
-5	5	Quintuple
⋮	⋮	⋮
1	1	Single
2	1 and 2	Single and Double
3	1, 2 and 3	Single, Double and Triple
4	1, 2, 3 and 4	Single, Double, Triple and Quadruple
5	1, 2, ... 5	Single, Double, ... Quintuple
⋮	⋮	⋮
0	As many as possible	As many as possible

After the initial string, the zeroth row of data contains the number of values or levels for each test factor, separated by white space (tabs or space characters). Each subsequent row represents a test case. It starts with its test case number (e.g. 1, 2, etc.) followed by character strings corresponding to the values of the factors for the test case.

Example -- Analyzing Existing Test Cases -- Test Case output

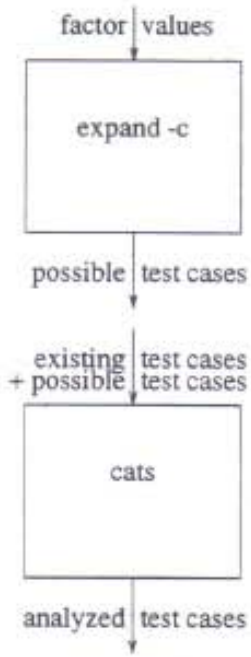
cats < contend.cats

-2	5	4	3	3	2:83	83	4:180
A	1	1	a	a	2:77	77	
E	2	1	b	b	2:71	71	
G	2	3	c	a	2:65	65	
I	3	1	c	c	2:59	59	
K	3	3	a	b	2:53	53	
O	4	3	b	c	2:47	47	
D	1	4	b	a	2:42	42	
R	5	2	c	c	2:37	37	
T	5	4	a	b	2:32	32	
B	1	2	a	b	2:28	28	
P	4	4	c	a	2:24	24	
C	1	3	a	c	2:21	21	
F	2	2	b	c	2:18	18	
J	3	2	a	a	2:15	15	
H	2	4	c	b	2:13	13	
L	3	4	a	c	2:11	11	
N	4	2	b	b	2:9	9	
S	5	3	a	a	2:7	7	
M	4	1	b	a	2:6	6	
Q	5	1	c	b	2:5	5	

The test cases now appear in a different order (identified with the same letters as in the input file).

There are 83 pairs of factor values to be tested in this example; there are 180 possible test cases. This particular set of test cases leaves five of the factor value combinations untested.

Supplementing Test Cases



Example -- Supplementing Test Cases -- Preparing New Test Cases

Pat wants to add a few new test cases, so all pairwise combinations will be tested. First Pat prepares an **expand** input file (**contend.exp**) for this problem.

```
1  2  3  4  5
1  2  3  4
a  b  c
a  b  c
```

Second Pat uses the **-c** option of **expand** to prepare a file of all possible test cases. She types

expand -c < contend.exp > contend.all

and gets a file of test cases (**contend.all**) which looks like this.

```
-2   5  4  3  3
1    1  1  a  a
2    1  1  a  b
3    1  1  a  c
4    1  1  b  a
5    1  1  b  b
:    :  :  :  :
178  5  4  c  a
179  5  4  c  b
180  5  4  c  c
```

Third Pat edits **contend.all** to remove the top line from the file. When she is done, **contend.all** contains only test cases -- candidates to supplement the original test cases.

Example -- Supplementing Test Cases -- Test Case input

Next Pat prepares a **cats** input file containing both old and new test cases. She types

cat contend.cats contend.all > contend2.cats

The following file results.

```
-2   5  4  3  3
A    1  1  a  a
B    1  2  a  b
C    1  3  a  c
D    1  4  b  a
E    2  1  b  b
F    2  2  b  c
G    2  3  c  a
H    2  4  c  b
I    3  1  c  c
J    3  2  a  a
K    3  3  a  b
L    3  4  a  c
M    4  1  b  a
N    4  2  b  b
O    4  3  b  c
P    4  4  c  a
Q    5  1  c  b
R    5  2  c  c
S    5  3  a  a
T    5  4  a  b
1    1  1  a  a
2    1  1  a  b
3    1  1  a  c
4    1  1  b  a
5    1  1  b  b
:    :  :  :  :
178  5  4  c  a
179  5  4  c  b
180  5  4  c  c
```

Example -- Supplementing Test Cases -- Analyzed Test Case output

Pat does her new test case analysis:

cats < contend2.cats

She gets the following **cats** output.

-2	5	4	3	3	2:83	83	4:180
A	1	1	a	a	2:77	77	
E	2	1	b	b	2:71	71	
G	2	3	c	a	2:65	65	
I	3	1	c	c	2:59	59	
K	3	3	a	b	2:53	53	
O	4	3	b	c	2:47	47	
17	1	2	c	b	2:41	41	
48	2	2	a	c	2:35	35	
85	3	2	b	a	2:29	29	
P	4	4	c	a	2:24	24	
T	5	4	a	b	2:19	19	
33	1	4	b	c	2:14	14	
R	5	2	c	c	2:11	11	
110	4	1	a	b	2:8	8	
148	5	1	b	a	2:5	5	
C	1	3	a	c	2:4	4	
H	2	4	c	b	2:3	3	
L	3	4	a	c	2:2	2	
N	4	2	b	b	2:1	1	
S	5	3	a	a	2:0	0	
B	1	2	a	b	2:0	0	
D	1	4	b	a	2:0	0	
F	2	2	b	c	2:0	0	
J	3	2	a	a	2:0	0	
M	4	1	b	a	2:0	0	
Q	5	1	c	b	2:0	0	
1	1	1	a	a	2:0	0	
2	1	1	a	b	2:0	0	
3	1	1	a	c	2:0	0	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
178	5	4	c	a	2:0	0	
179	5	4	c	b	2:0	0	
180	5	4	c	c	2:0	0	

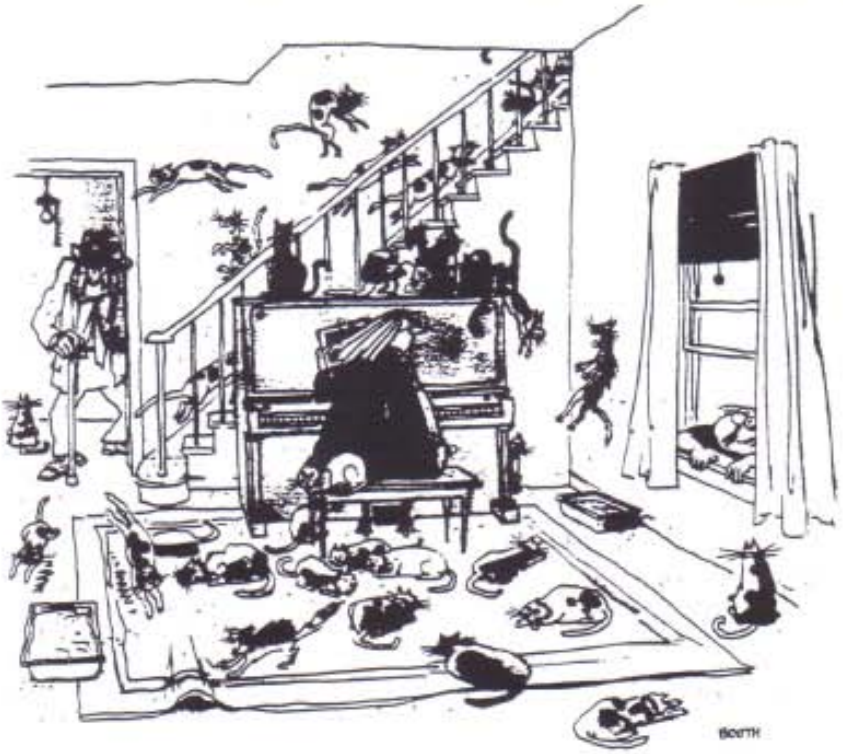
Example -- Supplementing Test Cases -- Conclusion

CATS recommends twenty test cases to do the job -- 14 of the original ones and six new ones (#17, #48, #85, #33, #110 and #148). CATS identifies six of the original test cases as redundant with this new set of twenty (#B, #D, #F, #J, #M and #Q). These six original cases do not improve the coverage after the first twenty test cases are done.

On the basis of this information, Pat will add to her test plan the six new test cases (#17, #48, #85, #33, #110 and #148). Pat also will decide whether the time saved by not testing #B, #D, #F, #J, #M and #Q is worth the cost of removing them from her plan.

- **Available Within AT&T Technical Community**
- **At All Major Bell Labs Locations**
- **Runs on 3B2, 386, Sun and HP Machines**
- **Largest known run --**
  - **— 20 Factors**
  - **— 10 Values for each Factor**
  - **— 240 Test Cases**
- **Informal CATS Testing via attmail!runcats or cats@cmprime!att.com**

THE SYSTEM TEST CHALLENGE: PROCESS IMPROVEMENT



"Glendora Hagan got in another load of cats, Elinor honey. Can you take a couple?"

Drawing by Booth: © 1989 The New Yorker Magazine, Inc.