Cause-Effect Graphing
By Theresa Hunt

www.westfallteam.com

Cause-Effect Graphing (CEG) is a model used to help identify productive test cases by using a simplified digital-logic circuit (combinatorial logic network) graph. It's origin is in hardware engineering but it has been adapted for use in software engineering. The CEG technique is a Black-Box method, (i.e. it considers the external behavior of a system with respect to how that system has been specified). It takes into consideration the combinations of causes that result in effecting the system's behavior. Although there are some issues with CEG such as the difficulty in discerning causes and effects from some natural language specifications, CEG remains a good way to analyze specification completeness and represent logic relationships from which productive test cases can be identified [Nursimulu-95]. As George E. P. Box, the influential 20th century statistician, said: "Essentially, all models are wrong, but some are useful", he also put it another way: "Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful".

The commonly used process for CEG can be described in the following six steps:

Step one: Break the specification down into workable pieces.

First, the functional requirements are decomposed and analyzed. To do this, the functional requirements are partitioned into logical groupings, for example, commands, actions, and menu options. Each logical grouping is then further analyzed and decomposed into a list of detailed functions, sub-functions, and so forth, as illustrated in Figure 1.

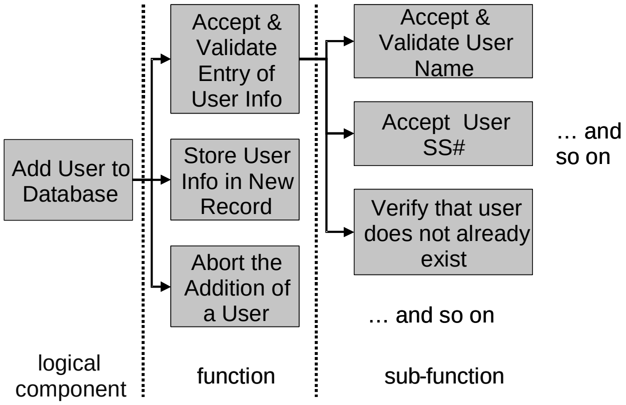

Add User to Database
Accept & Validate Entry of User Info
Store User Info in New Record
Abort the Addition of a User
Accept & Validate User Name
Accept User SS#
Verify that user does not already exist
… and so on
… and so on
logical component
function
sub-function

Figure 1 - Decomposed Function List

Step two: Identify the causes and effects.

a) Identify the causes (the distinct or equivalence classes of input conditions) and assign each one a unique number.

A cause can also be referred to as an input, as a distinct input condition, or as an equivalence class of input conditions. In equivalence class partitioning, each input or output is divided into subset domains that represent both valid and invalid values. For example, if an input has a specified range of 1 to 100 there are three equivalency classes, one valid class containing all values from 1 to 100 inclusive, and two invalid classes, values less than one and values greater than 100. Examining the specification, or other similar artifact, word-by-word and underlining words or phrases that describe inputs helps to identify the causes. An input (cause) is an event that is generated outside an application that the application must react to in some fashion. Examples of inputs include hardware events (e.g. keystrokes, pushed buttons, mouse clicks, sensor activations), API calls, return codes, and so forth.

b) Identify the effects or system transformation and assign each one a unique number.

An effect can also be referred to as an output action, as a distinct output condition, as an equivalence class of output conditions or as an output such as a confirmation message or error message. An output (effect) is an event that an application generates that is sent outside the application. Examples of output include a message printed on the screen, a string sent to a database, a command sent to the hardware, a request to the operating system, and so forth. System transformations such as file or database record updates are considered effects as well. As with causes, examining the specification, or other similar artifact, word-by-word and underlining words or phrases that describe outputs or system transformations helps to identify the effects.

Consider the following set of requirements as an example:

> Requirements for Calculating Car Insurance Premiums:
>
> R00101 For females less than 65 years of age, the premium is $500
>
> R00102 For males less than 25 years of age, the premium is $3000
>
> R00103 For males between 25 and 64 years of age, the premium is $1000
>
> R00104 For anyone 65 years of age or more, the premium is $1500

When examining these requirements, we see that there are two variables that will determine what the premium will be: sex and age. Therefore there are 5 distinct *causes* or input conditions: Sex is either male or female, and age is either less than 25, between 25 and 64 inclusive, or 65 or more. There are also 4 distinct *effects* or ouput conditions: the premium is either $500, $1000, $1500, or $3000. As shown in Table 1 below, each cause and each effect is assigned an arbitrary unique number as part of this process step.

Causes (input conditions)	Effects (output conditions)
1. Sex is Male	100. Premium is $1000
2. Sex is Female	101. Premium is $3000
3. Age is <25	102. Premium is $1500
4. Age is >=25 and < 65	103. Premium is $500
5. Age is >= 65	

Table 1 – Causes and Effects

Step three: The semantic content of the specification is analyzed and transformed into a Boolean graph linking the causes and effects. This is the cause-effect graph.

Semantics, in this step’s instructions, reflect the meaning of the programs or functions. This meaning is discerned from the specification and transformed into a boolean graph that maps the causes to the resulting effects. It is easier to derive the boolean function for each effect from their separate CEGs, then these can be overlaid to produce a single decision table for all effects (see step 5). For example, the following separate CEGs (see Table 2) can be derived from the Calculating Car Insurance Premiums example above.

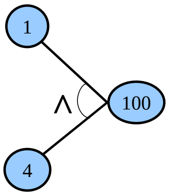
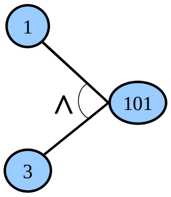
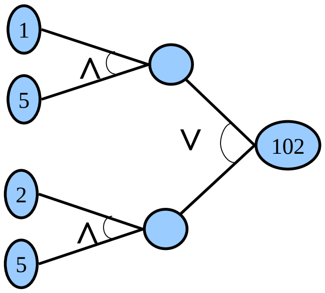
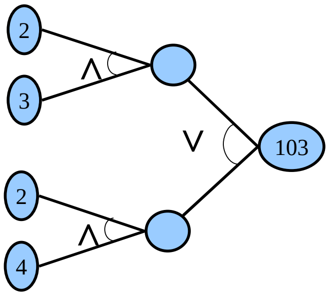
CEG	Interpretation
CEG #1: 	Causes: 1. Sex is Male and (∧) 4. Age is >=25 and < 65 Effect: 100: Premium is $1000
CEG #2: 	Causes: 1. Sex is Male and (∧) 3. Age is <25 Effect: 101: Premium is $3000
CEG #3: 	Causes: 1. Sex is Male and (∧) 5. Age is >= 65 or (∨) 2. Sex is Female and (∧) 5. Age is >= 65 Effect: 102: Premium is $1500
CEG #4: 	Causes: 2. Sex is Female and (∧) 3. Age is <25 or (∨) 2. Sex is Female and (∧) 4. Age is >=25 and < 65 Effect: 103: Premium is $500

Table 2 – Cause-Effect Graphs

Step four: Annotate the graph with constraints describing combinations of causes and/or effects that are impossible because of syntactic or environmental constraints or considerations.

In most software programs, certain combinations of causes are impossible because of syntactic or environmental considerations. For example, for the purpose of calculating insurance premium in the above example, a person cannot be both a “Male” and a “Female” simultaneously. To show this, the CEG is annotated, as appropriate, with the following constraint symbols (see Table 3):

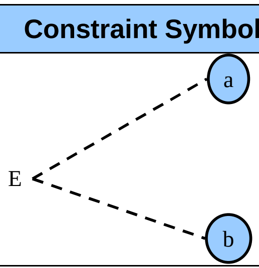
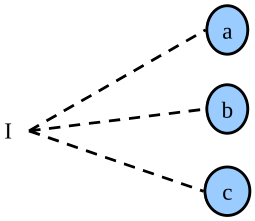
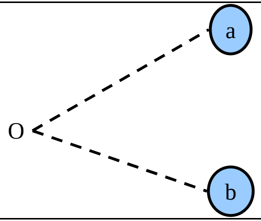
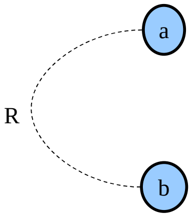
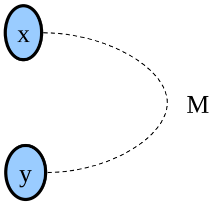
Constraint Symbol	Definition
	The "E" (Exclusive) constraint states that both causes *a* and *b* cannot be true simultaneously.
	The "I" (Inclusive (at least one)) constraint states that at least one of the causes *a, b* and *c* must always be true *(a, b,* and *c* cannot be false simultaneously).
	The "O" (One and Only One) constraint states that one and only one of the causes *a* and *b* can be true.
	The "R" (Requires) constraint states that for cause *a* to be true, than cause *b* must be true. In other words, it is impossible for cause *a* to be true and cause *b* to be false.
	The "M" (mask) constraint states that if effect *x* is true; effect *y* is forced to false. (Note that the mask constraint relates to the effects and not the causes like the other constraints.

Table 3 – Constraint Symbols

For example, as illustrated in the graph in Figure 2, CEG #3 from Table 2 in step 3 for the Calculating Car Insurance Premiums example would be annotated with a one and only one constraint between causes 1 and 2 because the Sex cause has to be either "Male" or "Female" but it can never be both.

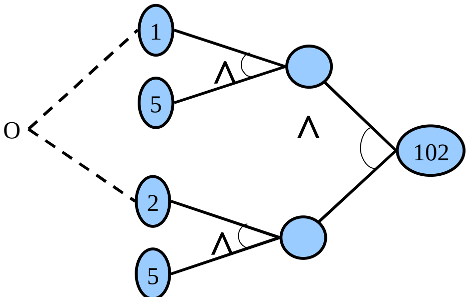


Figure 2 - Example of "O" constraint

Step five: Methodically trace state conditions in the graphs, converting them into a limited-entry decision table. Each column in the table represents a test case.

The ones (1) in the limited entry decision table column indicate that the cause (or effect) is true in the CEG and zeros (0) indicate that it is false.

Table 4 below illustrates the limited-entry decision table created by converting the CEG from the Calculating Car Insurance Premiums example. For example, the CEG #1, from Table 2 in step 3, converts into test case column 1 in the table below. From CEG #1, causes 1 and 3 being true result in effect 101 being true.

Test Case	1	2	3	4	5	6
Causes:						
1 (male)	1	1	1	0	0	0
2 (female)	0	0	0	1	1	1
3 (<25)	1	0	0	0	1	0
4 (>=25 and < 65)	0	1	0	0	0	1
5 (>= 65)	0	0	1	1	0	0
Effects:						
100 (Premium is $1000)	0	1	0	0	0	0
101 (Premium is $3000)	1	0	0	0	0	0
102 (Premium is $1500)	0	0	1	1	0	0
103 (Premium is $500)	0	0	0	0	1	1

Table 4 – Limited-Entry Decision Table

Some CEGs may result in more than one test case being created. For example, because of the one and only one constraint in the annotated CGE #3 from step 4, this CEG results in test cases 3 and 4 in the decision table above.

Step six: The columns in the decision table are converted into test cases.

Converting the decision table above would result in the folowing test cases (see Table 5):

Test Case #	Inputs (Causes)		Expected Output (Effects)
	Sex	Age	Premium
1	Male	<25	$3000
2	Male	>=25 and < 65	$1000
3	Male	>= 65	$1500
4	Female	>= 65	$1500
5	Female	<25	$500
6	Female	>=25 and < 65	$500

Table 5 – Test Cases

Conclusion

It should be noted that the example used in this paper to illustrate the basic steps of CEG was kept very simple. An astute software tester could probably jump right to this set of test cases from the requirements without using the CEG method. However, for large, complex systems with multiple causes (inputs) and effects (outputs or transformations) this method is a systematic way to analyze them to create test cases. If CEG is performed early in the project, it can help in developing and verifying the completeness of the specification.

References

Myers-04 Glenford J. Myers, *The Art of Software Testing Second Edition*, John Wiley & Sons, New Jersey, 2004, ISBN 0-471-46912-2.

Nursimulu -95 K. Nursimulu, K and R. Probert, *Cause-Effect Graphing Analysis and Validation of Requirements*, IBM Centre for Advanced Studies Conference**,** Proceedings of the 1995 conference of the Centre for Advanced Studies on Collaborative research, Toronto, Ontario, Canada

Box-87 George E. P. Box as quoted in Norman R. Draper, *Empirical Model-Building and Response Surfaces*, Wiley, 1987, ISBN 0-471-81033-9.