Guidance for test selection based on the cost of errors

by DAVID A. GUSTAFSON
Kansas State University
Manhattan, Kansas

ABSTRACT

A continual problem in the area of software testing is deciding if and where in a program additional testing should be done. Recent work by Cheung has indicated that the relative reliability of the individual nodes in a software flow graph, or modules in a software structure, can be used to guide the testing. This paper attempts to aid this process by suggesting a method for assigning a cost factor to the individual nodes in the software flow graph. This cost can be used to guide selection of additional tests.
INTRODUCTION

A problem in software testing is deciding how much testing is to be done and what tests should be used. Approaches to software testing are many and varied. Work in the area of reliability is giving guidance on how many tests should be done. Test coverage measures give guidance on both how many tests to run and on what to test. Functional testing proposes that test cases are to be selected for each function in the software specification. Some methodologies combine all of these approaches. However, these approaches are all based on the idea that for the chosen criterion, all the instances should be tested equally, whether the criterion is the testing of statements, decisions, branches, paths, functions, and so forth. That is, these approaches consider all errors, decisions, functions, etc. to be equally serious and important.

This equality is not always a reasonable assumption. Some errors are more probable than others because the necessary input conditions are more likely to occur. Some errors are more serious because their effects are more serious. The testing effort should concentrate on the more serious and more common errors.

As an example, consider a system designed to train people in recognizing equilateral triangles from triangles that are close to being equilateral. A set of triangles could be presented on a visual display, the trainee could be instructed to pick which one is equilateral, and the system could respond with the type of triangle that was picked. Part of the software could be a simple routine to determine the type of the triangle that was picked. The critical errors for this routine are (1) the triangle being incorrectly classified as equilateral and (2) the triangle being incorrectly classified as not equilateral. The most frequent data will probably be equilateral triangles. Those cases that are not triangles may be very rare or nonexistent. Intuitively, more emphasis should be placed on verifying that triangles are classified correctly than on whether they are equilateral or not.

Work by Cheung has indicated that some nodes in the flow graph are more critical than others to correct behavior of the program. His analysis is based on the user profile of transitions between the nodes or modules. Using this empirically derived profile and a Markov model of the transitions between the nodes, he identifies which nodes are more critical. However, he does not include any parameters for the criticality of the possible errors.

There are many times that certain errors (e.g., incorrectly identifying a triangle as equilateral) are much more serious than other errors (e.g., incorrectly identifying a scalene triangle as isosceles). The model proposed in this article is based on the estimated criticality of possible errors and the estimated frequency of occurrence of cases. An estimate is calculated for the criticality of the decisions and computations made at each node in the flow graph.

ASSIGNING COSTS

The errors that occur in programs can be classified as domain errors or computational errors. Domain errors are those where an incorrect decision causes a particular datapoint to be treated as a different type. That is, the inputs are considered to be from the wrong domain. The other type of error is called a computational error. That is, the computation in a particular domain is incorrect. For example, the square root may be incorrectly calculated.

All errors can be considered as one of these two types. For any datapoint (i.e., a point in the domain) a certain path through the program is executed and computations unrelated to control decisions may be done. A datapoint either follows the correct execution path or it does not (a domain error may occur). The noncontrol computations done on the actual execution path are either correct or incorrect (a computation error may occur). Although these two error categories are large, they seem to be inclusive. Other error classification schemes are valuable for other uses, but these two categories are of interest for this model. Therefore, we will consider them to be either domain or computational errors.

A number of complex situations can arise. First, both types of errors can occur for one datapoint. Second, an incorrect computation may seem to be correct (e.g., \(x + 2\) instead of \(2^x\) for the value \(x = 2\)). Finally, a datapoint may execute an incorrect path but later rejoin the correct execution path. All of these situations are considered errors and the differences are not significant in the estimations done in this paper.

Expected costs can be assigned to these two kinds of errors. The expected cost of a potential domain error is the cost of a domain error multiplied by the probability of that input case occurring. The expected cost of a potential computation error is the cost of a computation error multiplied by the probability of that computation being done.

In the triangle example, we may be able to estimate the costs of actual errors. Let us assume that two of the errors have costs associated with them. These costs may be based on the estimate of the amount of additional training necessary for the trainee to develop the necessary skill level after being misled by an incorrect answer. Assume that incorrect identification of a nonequilateral triangle as equilateral might require $200 of additional training and an incorrect typing of an equilateral triangle as nonequilateral might cost $100.

If we also knew the probability of actual errors occurring, we could calculate the expected cost of the errors. In our example, if we knew an actual frequency of these two errors
The relative expected cost of a potential error is the estimated cost of that particular error multiplied by the relative frequency of occurrence of that type of data. In our example, the trainees might select equilateral triangles 80% of the time and other triangles 18% of the time (2% might be nontriangles). Thus, the relative expected costs would be $36 for the nonequilateral and $80 for the equilateral. These costs are relative since the frequencies are relative.

We will assume that the user can assign costs such as these to the domain errors. That is, the user must give a cost for an outcome of type i when the correct outcome is type j. Denote these costs by $C_{ij}$. $C_{ij}$ represents the cost of an incorrect answer of type i when the correct answer was type j. For our example, $C_{eq} = C_{qf} = C_{q} = C_{qa} = C_{aq} = $200, and $C_{eq} = C_{f} = C_{e} = C_{i} = C_{o} = $100, and every other $C_{ij}$ is zero. Note that q stands for equilateral, i for isosceles, r for right scalene, o for obtuse scalene, a for acute scalene, and e for error (nontriangle).

The user also must estimate the cost of computation errors. That is, the cost of an incorrect calculation must be specified. $C_{ii}$ will denote the computation error for type i. In the triangle example, there are no nondcision computations and so $C_{aa} = C_{qq} = C_{rr} = C_{ee} = C_{ii} = C_{oo} = 0$.

In addition, the user must be able to assign the relative frequencies of the datapoints. In the triangle example, 80% were equilateral (i.e., $f_{q} = .8$), 18% were nonequilateral (assume $f_{i} = f_{o} = f_{r} = .045$), and 2% were not triangles (i.e., $f_{e} = .02$).

These values will be used to analyze the criticality of the parts of the program. The analysis will be done on the standard flow graph of the program. In the flow graph, nodes stand for branch-free sections of code. The arcs stand for possible execution paths between these branch-free sections of code. Each type can be assigned to at least one node in the flowgraph. This node is where the datapoint is identified as belonging to that type. In the triangle example, each terminal node is associated with a particular type of triangle. In these nodes, the name of the particular type of triangle is returned to the calling program. Additionally, any node that does nondcision computations has particular types of datapoints associated with that node.

**Fundamental Rule:** The criticality of errors in a node is related to the sum of the expected cost of potential errors in the computations done in that node plus the increase in successor nodes of the expected cost of potential domain errors due to decisions in that node.

The fundamental rule states the criticality of a node or the potential cost of errors in a node is related to two types of errors: errors in the computations done in that node and errors in the decisions made in that node. The expected cost of potential computation errors is directly related to the activities of a node. The expected cost of potential domain errors in a node is the result of decisions made in predecessor nodes. Thus, the increase in this expected cost is related to the criticality of the node. The fundamental rule, besides being intuitively correct, allows for the consistent and logical propagation of the relative expected costs throughout the flow graph.

**ASSIGNING DOMAIN COSTS TO NODES**

The relative expected cost of the potential domain errors can be assigned to the nodes in the flow graph. This cost is interpreted as the expected cost of incorrectly executing that node.

**Rule 1:** The expected cost of potential domain errors in a terminal node is the expected cost of incorrectly being in that node.

The expected cost of incorrectly being in a terminal node that has type i datapoints assigned to it is the summation for all j of $C_{j}*f_{j}$ (i $\neq$ j). Thus, these $C_{j}*f_{j}$ will be put in the cost set of that node. The expected cost is the sum of the terms in the cost set. Thus, every terminal node in the flow graph can be given an expected cost of potential domain error.

**Rule 2:** The expected cost of potential domain errors in a nonterminal node is the sum of the expected costs of potential domain errors of the successor nodes minus the terms related to the decisions made in the node.

The cost of a nonterminal node can be calculated as follows:

1. Add the cost set of each successor node
2. Subtract any pairs of terms $C_{j}*f_{j}$ and $C_{i}*f_{i}$ where both are in the cost set

Note that the criticality of errors is related to the increase in the expected cost of domain errors.

**Rule 2a:** The expected cost of the potential domain errors of the nodes in a cycle is the sum of the expected costs of potential domain error of the successor nodes for all nodes in the cycle minus the terms related to all decisions made in the cycle.

This rule means that all nodes in a cycle (loop) have the same expected cost of potential domain error. This expected cost is the same because all nodes in the loop are potentially executed on each iteration. The criticality of the nodes in a loop may not be the same because of the expected costs of the successor nodes of each node in the loop.

The cost of a nonterminal node in a cycle can be calculated as follows:

1. Add all terms $C_{j}*f_{j}$ from the cost sets of the successor nodes of nodes in the cycle.
2. Subtract all pairs of terms $C_{j}*f_{j}$ where both are in the cost set.
ASSIGNING COSTS OF COMPUTATION ERRORS

The expected cost of a computation error in a terminal node is the product \( C_{ii} \times f_i \) for types \( i \) assigned to the node. The expected cost of a potential computation error in a non-terminal node that is involved in computations is the product of \( C_{ii} \) and \( f_i \) for all types \( i \) that are related to that node.

ASSIGNING CRITICALITY TO NODES

The criticality of a node is the sum of the increase in the expected cost of domain errors and the expected cost of computation error for all computations done in the node.

AN EXAMPLE

Figure 1 is the flow graph of the triangle problem. The task is to identify the type of triangle given the lengths of the three sides. The nodes are labeled by number to the left of each node. There are no computations involved in this problem. The only possible errors are domain errors. Table I gives the expressions for the expected costs of potential domain errors for each of the nodes. The cases are referred to by letter instead of by number. For example, \( Coe \) is the cost of identifying the triangle as obtuse when it should have been an error case. The frequency of occurrence of the cases is denoted by \( f_i \). For example, \( fa \) is the expected frequency of acute triangles. Table I shows the values of the criticality if all of the \( C_{ij} \) are 1 and all frequencies are equal. Also indicated are the values of the expected costs and criticalities if the \( C_{ij} \) and frequencies had the values from the example. Using the sample expected costs and relative frequencies gives a different ranking for the criticality of the nodes. Although nodes 3 and 1 are still ranked number one and two, node 4 is now ranked third and at almost the same ranking as node 1. This would indicate that much more extensive testing should be done on nodes 3, 1, and 4.

Looking at the original flow graph, this recommendation can be converted to a description of the important types of test data. The most critical node is node 3, which involves a decision about whether two of the lengths are equal. Thus, the most important type of test case involves two of the lengths being equal or close to equal.

The second most critical node is node 1. This node tests whether or not the three lengths are properly ordered. Thus the second most important type of test case involves three sides being improperly ordered. Finally, the third most critical node involves a decision about whether all three of the sides are equal or close to equal.

Table I—The triangle example

<table>
<thead>
<tr>
<th>Node</th>
<th>Expected Costs of Potential Domain Errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node 10: ( Coe \times f_e + Coe \times f_i + Coe \times f_q + Cor \times f_r + Coe \times f_a )</td>
<td></td>
</tr>
<tr>
<td>Node 11: ( Coe \times f_e + Coe \times f_i + Coe \times f_q + Cor \times f_r + Coe \times f_o )</td>
<td></td>
</tr>
<tr>
<td>Node 9: ( (Coe + Cae) \times f_e + (Cai + Coi + Cq) \times f_i + (Cor + Car) \times f_q )</td>
<td></td>
</tr>
<tr>
<td>Node 8: ( Cae \times f_e + Cae \times f_i + Cae \times f_q + Cor \times f_r + Cae \times f_o )</td>
<td></td>
</tr>
<tr>
<td>Node 7: ( (Coa + Cae + Cre) \times f_e + (Cai + Coi + Cri) \times f_i + (Cor + Car + Cre) \times f_q )</td>
<td></td>
</tr>
<tr>
<td>Node 6: ( Cae \times f_e + Cae \times f_i + Cae \times f_q + Cor \times f_r + Cae \times f_o )</td>
<td></td>
</tr>
<tr>
<td>Node 5: ( Cie \times f_e + Cie \times f_i + Cie \times f_q + Cio \times f_r + Cie \times f_o )</td>
<td></td>
</tr>
<tr>
<td>Node 4: ( (Cq + Cia) \times f_e + (Cia + Cre) \times f_i + (Cor + Cre + Cae) \times f_q + (Ceo + Cia) \times f_o )</td>
<td></td>
</tr>
<tr>
<td>Node 3: ( (Cq + Cre + Cae + Cae + Cae) \times f_e )</td>
<td></td>
</tr>
<tr>
<td>Node 2: ( Cae \times f_e + Cae \times f_i + Cae \times f_q + Cor \times f_r + Cae \times f_o )</td>
<td></td>
</tr>
<tr>
<td>Node 1: none</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>all ( C_{ij} = 1 )</th>
<th>( C_{ij} ) different</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>Expected Criticality</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>5</td>
</tr>
</tbody>
</table>

Figure 1—The triangle example
A possible testing approach would be to select cases of these three critical types in proportion to the criticality of those nodes, for example, three times as many cases of type one (two sides equal or almost equal) as of type two (sides ordered wrong) or type three (all sides equal or almost equal). Additional tests would be used to achieve C1 coverage of the program. This approach would emphasize testing for the errors that would be more costly.

ANOTHER EXAMPLE

Figure 1 has the flow graph of a program with a loop. The terminal nodes contain a letter that represents the proper case for that node. The terminal nodes also have potential computation errors. The cost of a potential computation error is represented by $C_{ij}$ for case $i$. The expressions for expected cost of potential domain errors are shown in Table II. Note that the expressions for the three nodes in the cycle are the same. The second part of Table II shows the numeric values for the expected cost of potential domain errors, the expected cost of potential computation errors, and the potential cost for each node. These were calculated with the $C_{ij}$ all equal to one. Note that the potential cost of the three nodes (2, 4, and 6) in the cycle is not identical.

IMPLEMENTATION

An implementation of this algorithm was written in PASCAL. The implementation is approximately 300 lines long. A two-dimensional array is used to represent the expression for the expected cost at a node. The combining and reduction operations involve logical and transform operations on the arrays. Documentation on the implementation is available from the author.

CONCLUSION

This model shows that the individual nodes in a flow graph can be analyzed for criticality using estimated costs of errors and estimated distribution of input cases. This analysis will be useful in deciding which nodes should be tested more thoroughly.

Once the criticality of each of the nodes in a flow graph is established, the testing effort can be distributed in proportion to the criticality of each node. For each node, the activities and decisions in that node will suggest what tests should be done for that node. The resulting sets of tests should evaluate the program in relation to the criticality of the possible errors.

This model also allows the evaluation of different software structures based on how the criticality of the nodes is spread throughout the flow graph. A flow graph in which the potential cost of individual nodes is minimized would seem to be preferable. Any node that has a very high potential cost should be suspect and a structure that causes a few nodes to have a high potential cost should be avoided.

This model is an initial attempt at approaching the problem.
of associating input distributions and knowledge of differences in the seriousness of errors with the criticality of nodes and the evaluation of software structures. Future research will attempt to refine this model.

REFERENCES


During 1983 and 1984, computer graphics emerged from the shadows and became omnipresent. More than 80% of the personal computers being purchased in 1984 include some form of graphics capability. A similar percentage of the information centers at large corporations are installing new graphics software tools. Hardly any office automation system without a graphics component is being announced. Thousands of graphics designers are switching from manual to automated methods. Engineers in ever larger numbers are using computer graphics work stations for design and drafting. Even word processors are gaining computer graphics capability.

On the entertainment front, computer animation is playing an increasingly large role in successes such as the film "Star Wars." Simultaneously, video games have brought entertainment-oriented computer graphics into more than three million homes in America. Two of the NCC sessions cover state-of-the-art technology and applications in the very visible entertainment and animation areas. The remainder of the NCC '84 graphics track focuses on the fastest-growing segment and the one that directly affects the most computer users: management and business graphics. It offers a state-of-the-art overview of the hardware and software and looks ahead at the coming year.

Four major classes of graphics products are covered in the sessions: displays, work stations, hard copy output devices, and software. The computer graphics display of 1984 can best be thought of as a traditional alphanumeric display in which every dot (or pixel) on the screen is addressable. Pictures with high levels of detail require much more memory than do pictures with less. Until 1982, terminals that could display more than 1 million pixels (1000-by-1000 resolution) cost more than $25,000. However, by 1984, those terminals had dropped in price to under $10,000; and for slightly more a buyer could have a terminal with extraordinary local intelligence for panning and zooming. Simultaneously, graphics terminals with lower resolutions—e.g., those with 300-by-400 addressable pixels—were dropping radically in price. Today it is common to find black-and-white graphics terminals at less than $1,000, and color graphics terminals at less than $3,000. During this year, even higher resolution terminals (up to 2000 by 2000) will begin to appear, first in black and white and later in color.

The session entitled "Graphics on Microcomputers" discusses the stand-alone graphics work stations that are a new phenomenon created by the low-cost microprocessor. These work stations come in two varieties: the personal computer variety and the high-performance professional work stations. Graphics on personal computers are generally low resolution. They appear to be better than they are when the screens are very small. The professional work stations, on the other hand, have higher resolution and more computer power than the PCs and are used in high-payoff applications, such as slide production and scientific data analysis. Today there is a larger price premium on the professional work stations, but during this year more powerful personal computers will begin to erode the difference between professional work stations and personal computer graphics systems.

The session entitled "Experts Look at the Future" is concerned in part with new developments in hard-copy paper charts, overhead transparencies, and 35-mm slides. New technology is revolutionizing all three. The digital plotter has been the workhorse of today's management graphics systems; but it is being challenged by inkjet printers, which provide more color more quickly on both paper and transparency. At the same time, new laser printers are cutting production time for charts from 10 minutes to 10 seconds. On personal computers, low-cost plotters and inkjet printers are adding color output,
while higher-resolution matrix printers (200 dots per inch) are making black-and-white output more presentable.

The 35-mm slide production market is a fast-growing application of computer graphics because computers cut the cost of slides from $35 to $7 each. New digital film recorders have brought the price of high-quality slidemaking equipment to $25,000. Later this year, even newer systems promise price reductions to $10,000 with quality similar to the output of equipment that cost $200,000 only two years ago. The same type of price erosion is occurring in laser printers: prices are expected to be in the $5,000-to-$10,000 range by 1985.

Nearly every session will have a software component, because no computer graphics can be produced without it. The graphics software industry has blossomed into a $100 million business, primarily for mainframes and minicomputers. Personal computer graphics software is also becoming important, but the vast majority is imbedded in integrated systems.

The largest vendors of graphics software have made major advances in the past year, including the following:

—New software for integrating text and graphics to produce technical documentation.
—A new standard for user friendliness.
—New database linkages.
—Predesigned chartbooks.
—Layout intelligence that automatically designs the best-looking chart for the target audience.

Application graphics software is another growth area, with both project management graphics and executive chartbook systems gaining broad acceptance.

Graphics software standards have finally arrived after seven years of effort. One session, entitled "Emerging Standards," will focus on these new developments.

During the coming year, personal computer graphics packages will continue to evolve into more powerful tools. And at the same time, the much larger development budgets of the mainframe graphics vendors will increase the distance between personal computer graphics and mainframe graphics software capabilities. Later in the year, however, the gulf between mainframes and micros will close as new desktop computers are announced that will run the software that works only on mainframes today.

New management techniques for computer graphics in large corporations will be the focus of the session called "Graphics in the Information Center." The principal keys to effective management that pioneering users of computer graphics have found are as follows:

1. Give all computer users access to computer graphics through shared plotters, laser printers, and film recorders.
2. Provide links to databases and to old application programs.
3. Offer chartbooks as the principal user interface.
4. Offer project management software.
5. Offer both microcomputer graphics and mainframe graphics.
6. Provide software that offers user friendliness, extreme quality and flexibility, and device independence.
7. Start with high-payoff applications for senior management.

The explosive growth being experienced by the computer graphics industry can be attributed in part to the price performance improvements of hardware and software. But equally important to this growth is the new realization that graphics work, that they are cost effective, and that the people who bring computer graphics into organizations are making computers more useful to management and therefore more valuable to the organization as a whole.
More than three million personal computers will be sold in 1984 to an increasingly segmented market. There are handheld, laptop, desktop, stand-alone, networked, and supermicro personal computers. You can buy personal computers for office workers, lab technicians, data processing managers, financial planners, and people with myriad other job descriptions. Applications software for personal computers runs the gamut of agricultural to zoological. The Personal Computing track attempts to cover this multifaceted market, its key issues, and new product innovations.

1984 may go down as a year of the multiuser computer, with AT&T introducing computers from micro to mainframe; but the greatest interest is still at the personal computer level. UNIX has not achieved much acceptance on personal computers in the past, but AT&T’s involvement may change things. UNIX provides a vital micro-to-mainframe link for companies connecting their personal computers to corporate mainframe equipment. The session entitled “Multi-User and Networked Personal Computers” explores this subject in depth.

The portable market has fragmented into several submarkets, including the laptop, the handheld, and the original portable: the “luggable.” Apple and IBM’s announcements of portables may be reducing “luggable” to an industry requirement for personal computers, not an option. More truly portable products like the Tandy 100 are gaining from new software developed for their special markets. The session “Portable Computers and their Software” reviews the latest in portables and the software being developed especially for them.

“Frontiers in Personal Computing: The User Interface” focuses on a number of recent innovations in user interfaces. The user interface continues to be a major industry issue as frustrated users try to master the intricacies of operating systems and applications, all of which use different commands. Along with menus and multiple windows, innovative devices for easing the user interface problem include the “mouse,” touch-sensitive screens, softkeys, voice, and touch pads.

What will the next generation of personal computers look like? Networking and friendly user interfaces are a part of the picture. IBM’s rumored “Popcorn” is said to use a 286 microprocessor. Apple’s MAC is certainly a pioneer in the 32-bit, easy-to-use personal computer sweepstakes. AT&T’s entry is sure to stir interest. Voice/data can’t be far behind AT&T’s current computer offerings. “Next Generation PCs” reviews the latest developments and presents the views of industry experts on what’s coming next.

Software distribution and marketing continue to evolve, but constrained distribution channels have limited the mass appeal of some software products. New approaches to software distribution and design are described in various sessions of the Personal Computing track, with various iconoclasts describing their ideas. A particularly important issue is the prospect of a universal standard, a topic to which a session by this name has been devoted.

Integrated operating environments are packages in which all applications use the same data formats and are designed to work with each other. This is one of the most exciting trends in the standardization arena. The session “Data Management in Integrated Operating Environments” explores the key issues here and presents the reactions of those with experience in using these environments.

As the market becomes more complicated, forecast data and industry analysis become more important for those who want to stay well informed. Key industry analysts share their sometimes controversial, always interesting ideas in “The Personal Computer Industry: The Experts Forecast the Future.”

The Personal Computing track gives computer users, data processing professionals, computer industry members, and educators an overview of the state of the personal computer and in-depth analyses of areas of special interest.