Metrics Based Classification Trees for
Software Test Monitoring and Management

Raymond A. Paul
US Army Operational Test and Evaluation Command

Abstract
An important objective of software test programs is to identify "high-risk" components. This paper focuses on one method which can be applied to identify high-risk software components, the use of a classification tree with an established software metrics set. The selected examples of high-risk software components are those modules which are most likely to induce errors in the target operational system, and those software components which will require the most effort in the development process. The associated metrics are software reliability and productivity. This paper describes the methodology utilized by the US Army in the application of classification trees for analysis of software metrics data. A detailed example is provided with a step-by-step procedure for construction of a classification tree for software metrics analysis.

1: Introduction
Identification of software components which are potential risks in an operational system is an important management objective in any software development effort. If high-risk components can be identified early in a software development program, the manager can redirect resources to support preventative actions. Selby [1] demonstrated the utility of the application of classification trees for software risk management. Other practical applications of classification trees for risk management have been successfully applied in other areas, such as fuel enrichment [2]. This paper provides examples of the application of classification trees to evaluate two factors of risk in a software development environment. The two selected risk factors are software reliability and software development productivity. These high-level software risk factors can be measured by various software metrics, including software size and software complexity. This paper illustrates how classification trees can be applied to the sample metrics to identify those software components which present the highest risk during the software development effort.

Background of the Army Software Metrics Program. The Army Software Test and Evaluation Panel (STEP) was initiated in September 1989 to investigate and improve the state-of-the-art of test and evaluation (T&E) of US Army software. A primary recommendation of the STEP was to require reporting of a standard set of metrics on all Army software development and maintenance efforts. The mandatory set of 12 Army software metrics is illustrated in Table 1. The specific objective of the metrics set is to provide measurements which will allow an Army program manager to have visibility into the software management process. The Army metrics effort is supported by development of a central database for Army-wide metrics data collection.

| COST / SCHEDULE | Measures the total funding estimates for the software, compared with the actual expenditure schedule. |
| SCHEDULE | Measures the degree of adherence to planned major software development milestones. |
| COMPUTER RESOURCE UTILIZATION | Measures the degree to which target computer resources are changing and approaching the limits of resources available within the specified constraints. |
| SOFTWARE ENGINEERING ENVIRONMENT | Measures the capability of a contractor to use modern software engineering techniques in his development process. |
| REQUIREMENTS TRACEABILITY | Measures the adherence of software products to established requirements at various levels. It traces all Required Operational Capabilities (ROC) software-related requirements to the software specific. |
| REQUIREMENTS STABILITY | Measures the level of change in software requirements. |
| DESIGN STABILITY | Measures the level of change in the design of the software. |
| FAULT PROFILES | Plots the cumulative number of open and closed software anomalies at each priority level as a function of time. |
| BREADTH OF TESTING | Measures the extent to which required software functionality has been successfully demonstrated. |
| DEPTH OF TESTING | Measures the extent and success of testing for coverage of possible paths and conditions within the software. |
| RELIABILITY | Measures the number of observed faults in the software, and predicts the number of faults expected when the software is used in accordance with its OMS/MP. |
| COMPLEXITY | Measures complexity of software design and functions according to established industry models. |

Table 1: The US Army has established a minimum set of metrics for all software development programs.
This Army-wide database, designated the Software Metrics Management Information System (SMMIS), serves as the repository of lessons learned from the use of the Army software metrics. SMMIS output reports assist the Army manager to identify related deficiencies or anomalies in the software management program, based upon expected and actual values of the reported metrics.

The Army metrics data will allow software engineers and managers to identify the level of risk associated with significant issues in the software life cycle. However, software metric data is not of much use unless the program manager is able to extract useful information from the raw metrics data. Because Army program managers are generally not computer scientists, the Army has sponsored development of a series of software tools to support analysis and interpretation of metrics data. These tools are packaged as an overall analytical system, the Software Metrics Analysis and Reporting System (SMART). The SMART application programs provide a selection of analytical tools for metrics data; including classification trees, neural networks, and a knowledge-based system. This paper describes the classification tree component of SMART.

Classification Tree Overview. The classification tree is a powerful tool for analysis of metrics data to identify high-risk software components. The classification tree has a functional architecture which resembles a tree. The top branches represent the input statements, each of which defines a software module with specific characteristics. The trunk represents a series of decision nodes which perform further classification of the software modules using the metrics data which is reported for each module. The roots of the tree represent the subsequent classification of the software modules according to the level of risk associated with the management objectives. Software components are input through the top of the classification tree, classified at each inner node by the metrics values, and identified by level of risk at the bottom nodes. Because classification trees allow managers to orchestrate the use of several metrics, the trees may also serve as one type of metrics integration framework.

Classification Tree Terminology. The terms which a software manager may use to describe the risk that a software component has low reliability may include: "The component is error-prone" or "The component has a greater number of errors than the average". In terms of classification trees, these statements establish a "target class". The target class defines the conditions which the manager wants to identify in software components. In order to identify which software components are within the target class, a series of metrics must be applied which measure those characteristics which are related to the target class. These characteristics are defined as "contributing factors". The terms "metric" and "contributing factor" have the same meaning in this paper.

Once the target class and the contributing factors (metrics) have been identified, a corresponding classification tree can be generated. The "training set" is a baseline of historic data which has been collected on the contributing factors and the target classes. The training set provides an empirical basis to infer rules on the dependencies of contributing factors and target classes. The classification tree algorithm implements an evaluation function to build a tree in which every path from the root node leading to a leaf is a rule. Each node in the tree is a contributing factor (a software metric) and each branch of that node represents a value or possible range of values for the metric. Therefore, a path represents a condition, or a Boolean conjunction, of the values of the metrics at each node which the path may take. The conjunction is expressed in the form "metric X falls in range BX1 and metric Y falls in range RY2 and...". The leaf node represents the classification when the condition is satisfied. If the metric values of a new component satisfy the condition of any rule, then the leaf node classifies whether or not the component falls in the target class. In this way, once the tree has been generated from historic data, it can be used during the development program to identify components which may be error-prone. Based on such predictions, appropriate corrective measures can be taken well in advance. Similarly, knowing the development risks in advance can support a decision to change the project schedule to more efficiently achieve the project goals.

2: Example of the use of a classification tree

Consider a situation where a user wants the classification tree to predict whether a component might require extensive labor effort during development. Available metrics for this example include the size of the component, the number of decisions, and the number of Input-Output variables. For these selected metrics, a typical classification tree was constructed from the NASA data [1], as shown in Figure 1. This figure illustrates a classification tree constructed with the metrics of Size, Number of Decisions, and Number of Input-Output Variables. These metrics are respectively partitioned into the ranges (RX1, RX2, RX3), (RY1, RY2, RY3), and (RZ1, RZ2, RZ3). The "+" and "-" symbols at the leaf nodes indicate whether or not the component is expected to require a high level of effort during development. The tree can also be used for predicting trends of new components. For example, if a new component has a size (number of lines) of 300, number of decisions of 100, and the number of Input/Output variables is 130, then that component will require a high level of effort during development, according to the tree. This is shown by the path traced by broken lines in Figure 1. Similarly, other components fall under the categories which are shown in Table 2.
L = size of component
D = number of decisions
I = number of i/o variables

RX1: 27<L=500
RX2: 501<L=1500
RX3: 1501<L=2147

RY1: 0<D=50
RY2: 51<D=90
RY3: 91<D=162

RZ1: 0<I=100
RZ2: 101<I=120
RZ3: 121<I=238

New component (traced by dashed lines) with lines 300, Decisions 100, I/O variables 130, falls in target class (class‘+’)

Figure 1: A Sample Classification Tree

<table>
<thead>
<tr>
<th>Component</th>
<th>Size</th>
<th>Decisions</th>
<th>I/O Variables</th>
<th>Has high development effort?</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2000</td>
<td>20</td>
<td>35</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>1000</td>
<td>65</td>
<td>150</td>
<td>Yes</td>
</tr>
<tr>
<td>3</td>
<td>100</td>
<td>65</td>
<td>110</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 2: Categories of Components in Classification Tree

3: Methodology for constructing the classification tree

The method which is presented for constructing a classification tree is described in greater detail by Quinlan [2]. For this example, it should be assumed that the training set of components and a given set of contributing factors have already been determined. The contributing factors are the metrics which have been used to predict the high-risk software components.

Choosing the training set. An evaluation is performed on the complete set of software components for which historic data exists. The objective is to identify and eliminate those components that do not belong to a meaningful representative sample. For example, if two different components have the same metric values, but belong to opposite target classes, one or both of the components may be discarded. The resulting subset of components is the baseline training set.

Partitioning into ranges. The possible range of metric values for the training set of components is partitioned into a set of ranges. Partitioning can be done in different ways. One way is to first sort the values of a metric, then partition the metric into continuous ranges in such a way that each partition covers an approximately equal number of components. For example, the Size metric (number of lines of code) is partitioned into ranges: (27,500), (501,1500), (1501,2147) such that each range consists of approximately equal number of components.

Choosing a metric at a node. As the tree is built, at each node of the tree a metric is selected from the given set of metrics and assigned to the node. The metric which is selected is that metric (contributing factor) which incurs the minimum “cost”. Cost is a measure of the homogeneity of the range of metric values within the component partitions. The cost factor measures the homogeneity of the different range values of the metrics between the components. Values of cost are computed by an evaluation function that measures the value of the “partial tree” formed when a metric is assigned, and each partition of its values as branches of the node. This process of choosing the metric with minimum cost is repeated for every node. The set of metrics at a node is obtained by discarding the metric that was assigned to the parent from the set of metrics that were available at the parent.

Partial tree construction. Beginning with the root node, a partial tree is constructed for each metric. The partial tree construction procedures are the same for every metric. All the components are separated according to the partitions of
a metric's range of values, each of which denotes a branch. The selection criteria is whether or not the corresponding metric value of the component falls within the range of a partition. The next step is to derive the values of "P" and "N" for each set of components corresponding to each branch. The value of P is the number of components which belong to the target class (positive class) and N is the number of components which do not belong. The algorithm for evaluation of the partial tree is:

If P(i) and N(i) denote P and N for each branch 'i' of the partial tree, the cost is calculated as:

$$E = \sum E(i) = \sum \text{Weight}(i) \times F(P(i), N(i))$$

where

$$\text{Weight}(i) = \frac{S(i)}{\text{Number of components at the parent node}}$$

and

$$F(P(i), N(i)) = \left(\frac{P(i)}{S(i)}\right)^{\log 2(P(i)/S(i))} - \left(\frac{N(i)}{S(i)}\right)^{\log 2(N(i)/S(i))}$$

and

$$S(i) = P(i) + N(i)$$

Recursive metric assignment. Assign the metric with the least value of E (the least cost) to the current node. Once a metric is selected at a node, discard it from further consideration for all the children of the node. Apply this process recursively. The recursion stops when any of the "termination criteria" is satisfied. Termination criteria for the recursion may be divided into three cases which assist in deciding leaf nodes:

1. The primary termination case occurs when all the components at a node belong to the same target class. The node is marked as a leaf node. The leaf node is also marked as having either positive or negative target class.

2. Termination occurs if there are no components that can be filtered into a particular branch. This occurs when no components fall within a particular partition of a metric's range. The child node in that branch is marked as a leaf node with negative target class.

3. Termination also occurs if all components do not fall within the same target class, even though some components filter through a branch. In this termination case, all the metrics may have been assigned before arriving at the child node and there are no metrics which can be assigned to the child node. The child node is marked as a leaf node with negative target class.

In cases (2) and (3), the leaf nodes are assigned negative target class because nothing can be said in such a situation. An alternative may be to assign a new target class labeled "Unknown".

### 4. An Illustrative example for constructing a classification tree.

This example illustrates each step in constructing a classification tree. This example uses the subset of data from a NASA project which is shown in Table 3.

<table>
<thead>
<tr>
<th>Component Number</th>
<th>Number of Decisions</th>
<th>I/O Variables</th>
<th>Lines</th>
<th>Effort</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>92</td>
<td>20</td>
<td>8</td>
<td>121</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>93</td>
<td>15</td>
<td>15</td>
<td>148</td>
<td>87</td>
<td>0</td>
</tr>
<tr>
<td>94</td>
<td>9</td>
<td>11</td>
<td>95</td>
<td>162</td>
<td>1</td>
</tr>
<tr>
<td>95</td>
<td>11</td>
<td>7</td>
<td>90</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>96</td>
<td>34</td>
<td>31</td>
<td>229</td>
<td>313</td>
<td>1</td>
</tr>
<tr>
<td>97</td>
<td>8</td>
<td>14</td>
<td>150</td>
<td>115</td>
<td>0</td>
</tr>
<tr>
<td>98</td>
<td>7</td>
<td>5</td>
<td>86</td>
<td>62</td>
<td>0</td>
</tr>
<tr>
<td>101</td>
<td>11</td>
<td>21</td>
<td>108</td>
<td>145</td>
<td>1</td>
</tr>
<tr>
<td>102</td>
<td>12</td>
<td>11</td>
<td>124</td>
<td>162</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: NASA Data for Construction of a Classification Tree

### Partitioning into ranges.

In this example three ranges have been selected for every metric. The metric values are partitioned into the ranges by first sorting the data in increasing order. Since the software has 9 components, each partition gets 3 equal components. The LINES metric is assigned the first 3 components which are partitioned into the ranges 86-108, 109-124, and 148-229. Therefore, the continuous partitions are 86-107, 108-147, and 148-229. Similarly, the I/O VARIABLES metric values belong to the three partitions of 5-10, 11-14, and 15-31. The DECISIONS metric values belong to the partitions 7-10, 11-14, and 15-34.

### Assigning a metric at the root node.

A metric is assigned to the root node by constructing partial trees for each metric. The partial trees for the LINES, DECISIONS, and I/O VARIABLES metrics are shown in Figure 2. This figure shows the components that fall within the three ranges, the respective target class of each component, and the respective values of P, N, and E. These values are based on the partitions and the equations which were previously defined in Section 3.

For the partial tree with I/O VARIABLES metric at the root of the partial tree, the number of components is 9 and these values are observed: P(1)=0, N(1)=3, S(1)=3; P(2)=2, N(2)=1, S(2)=3; and P(3)=2, N(3)=1, S(3)=3. Since all components have negative class, E(1)=0. Using the formula from Section 3 of this paper for calculation of cost, the other values of cost for the I/O VARIABLES metric are calculated as: E(2)=0.3061 and E(3)=0.3061. Therefore,

$$\Sigma E(I/O \text{ VARIABLES}) = E(1)+E(2)+E(3)=0.6122.$$
Figure 2: This figure illustrates how a metric is assigned at the Root Node

as: \( E(1) = 0.3061; \ E(2) = 0.3061; \) and \( E(3) = 0.3061 \). Therefore,

\[
\Sigma E(\text{LINES}) = E(1) + E(2) + E(3) = 0.9183.
\]

For the partial tree with DECISIONS metric at the root of the partial tree, these values are observed: \( P(1) = 1, N(1) = 2, S(1) = 3; \ P(2) = 2, N(2) = 1, S(2) = 3; \) and \( P(3) = 1, N(3) = 2, S(3) = 3 \).

Using the formula from Section 3 of this paper for calculation of cost, the values of cost for the DECISIONS metric are calculated as: \( E(1) = 0.3061; \ E(2) = 0.3061; \) and \( E(3) = 0.3061 \). Therefore,

\[
\Sigma E(\text{DECISIONS}) = E(1) + E(2) + E(3) = 0.9183.
\]

Since \( \Sigma E(\text{I/O VARIABLES}) \) is less than \( \Sigma E(\text{LINES}) \) and \( \Sigma E(\text{DECISIONS}) \), the partial tree for I/O VARIABLES forms the root node as shown in Figure 3.

The next step is to assign metrics for each leaf of the partial tree for I/O VARIABLES. Metrics are assigned for each child node (for each range) of the partial tree for I/O VARIABLES shown in Figure 2. Since the first range contains all components with target class '+'
,' the target class of that leaf node is '+' and the process of building a tree at that leaf node stops.

 Metrics are then assigned to the second child node. The second range contains the components (94, 97, 102). Because all components do not fall under the same target class, another partial tree needs to be developed at this node. The metric I/O VARIABLES is removed from further consideration, since it has been assigned at the parent node. Each of the remaining two metrics LINES and DECISIONS form partial trees as shown in Figure 3, with the following details:

The LINES metric has 3 components with these observed values: \( P(1) = 1, N(1) = 0, S(1) = 1; E(1) = 0; \ P(2) = 2, N(2) = 1, S(2) = 1, E(2) = 0; \) and \( P(3) = 1, N(3) = 0, S(3) = 1, E(3) = 0 \). Therefore,

\[
\Sigma E(\text{LINES}) = E(1) + E(2) + E(3) = 0
\]

The DECISIONS metric has 3 components with these observed values: \( P(1) = 1, N(1) = 1, S(1) = 2; E(1) = 0.6667; \ P(2) = 0, N(2) = 0, S(2) = 1, E(2) = 0; \) and \( P(3) = 1, N(3) = 0, S(3) = 0, E(3) = 0 \). Therefore,

\[
\Sigma E(\text{DECISIONS}) = E(1) + E(2) + E(3) = 0.6667
\]

Figure 3: The I/O VARIABLE forms the Root Node
Since $\Sigma E(\text{LINES})$ is less than $\Sigma E(\text{DECISIONS})$, the partial tree for LINES is formed at the second child node of the partial tree for I/O VARIABLES as shown in Figure 3.

The LINES partial tree is checked to see if the child nodes can be assigned target classes. Target classes can be assigned only if all the components at a node are of the same class. As shown in Figure 3, this is true for all its children. The first child is a leaf node with target class '+' and the third leaf node is of target class '+'.

The next step is to continue the tree building process for the third child node of the partial tree for I/O VARIABLES. This process has been completed for the first and second child nodes. The metrics to be considered are the LINES and DECISIONS metrics, since we have already assigned the I/O VARIABLES metric. Details for each of the partial trees (shown in Figure 4) are:

The LINES metric has 3 components with these observed values: $P(1)=0$, $N(1)=0$, $S(1)=0$, $E(1)=0$; $P(2)=1$, $N(2)=0$, $S(2)=1$, $E(2)=0$; $P(3)=1$, $N(3)=1$, $S(3)=2$, $E(3)=0.6667$. Therefore,

$$\Sigma E(\text{LINES}) = E(1) + E(2) + E(3) = 0.6667$$

The DECISIONS metric has 3 components with these observed values: $P(1)=0$, $N(1)=0$, $S(1)=0$, $E(1)=0$; $P(2)=1$, $N(2)=0$, $S(2)=1$, $E(2)=0$; $P(3)=1$, $N(3)=1$, $S(3)=2$, $E(3)=0.6667$. Therefore,

$$\Sigma E(\text{DECISIONS}) = E(1) + E(2) + E(3) = 0.6667$$

In this case, both the partial trees have the same cost ($E$). Therefore, either one of the metrics can be chosen for assignment. The LINES metric is chosen and the process is continued.

After assigning the LINES metric, the only metric left is the DECISIONS metric. As shown in Figure 5, the partial tree details for the DECISIONS metric are: $P(1)=0$, $N(1)=0$, $S(1)=0$, $E(1)=0$; $P(2)=0$, $N(2)=1$, $S(2)=1$, $E(2)=0$; $P(3)=1$, $N(3)=1$, $S(3)=2$, $E(3)=0.6667$.

In this case, both the partial trees have the same cost ($E$). Therefore, either one of the metrics can be chosen for assignment. The LINES metric is chosen and the process is continued.
The first child node is assigned '-' target class and the second child node is assigned '+' target class. The third child node does not have all components of the same class and there are no more metrics to choose. Therefore, the target class '-' is assigned.

The final classification tree is shown in Figure 6 and the rules described by the classification tree are defined in Table 4.

5: Benefits from Classification Trees

Classification trees are considered superior to other classification methods because the resulting models are straightforward to build and interpret [3]. The generation process is extensible and new metrics can be added during construction of the tree. Classification trees can also be customized by using various metrics to classify different types of components in multiple development environments. Classification trees serve as metric integration frameworks and can be used to relate various high-risk properties of components to different software metrics. Likewise, classification trees can be used for inter-project evaluation. One of the most significant advantages of classification trees is that the tree-generation algorithm can be applied to large systems. For example, Quinlan [2] describes a project in which a training set incorporated 30,000 objects which were described in terms of 50 attributes.

6: References


<table>
<thead>
<tr>
<th>I/O VARIABLES</th>
<th>LINES</th>
<th>DECISIONS</th>
<th>Target Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>'-'</td>
</tr>
<tr>
<td>2</td>
<td>1 or 3</td>
<td></td>
<td>'+'</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td></td>
<td>'-'</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td>'+'</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td></td>
<td>'+'</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1 or 3</td>
<td>'+'</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>2</td>
<td>'-'</td>
</tr>
</tbody>
</table>

Table 4: This Example Illustrates How the Metrics Values Determine If a Component is