CODE ASSESSMENT METHODOLOGY PROJECT (CAMP)

Comparative Evaluation:

Coverity Prevent™ 2.4.0

Fortify Source Code Analysis Suite™ 3.5.1

GrammaTech CodeSonar™ 1.0p3

Klocwork Development Edition 7.0

Ounce Labs Prexis 3.2

(Release 1.0)

Prepared for:
Maryland Procurement Office
Technical Task Order (TTO) Number: 2005

Prepared by:
Concurrent Technologies Corporation (*CTC*)
100 *CTC* Drive
Johnstown, PA 15904-1935
and
Carnegie Mellon's CyLab
CIC Building
5000 Forbes Avenue
Pittsburgh, PA 15213

October 2, 2006

0017-UU-DD-061717

CTC Concurrent Technologies Corporation

POVD PROTECTION OF VITAL DATA

APPENDIX B: EVALUATING TOOL PERFORMANCE

Larry Maccherone

The traditional way to talk about the performance of a bug finding tool is to talk about the FP (FP) rate along with either the TP (TP) rate or the FN (FN) rate (all defined below). While it is convenient to have a single pair of numbers to characterize the behavior of a particular tool, it is not a sufficiently rich way to effectively evaluate the tool's performance for a wide range of possible uses and for a wide range of the tools settings. This appendix describes the more sophisticated means of evaluating tool performance used in this report[1].

Before the approach can be explained, some definitions need to be provided:

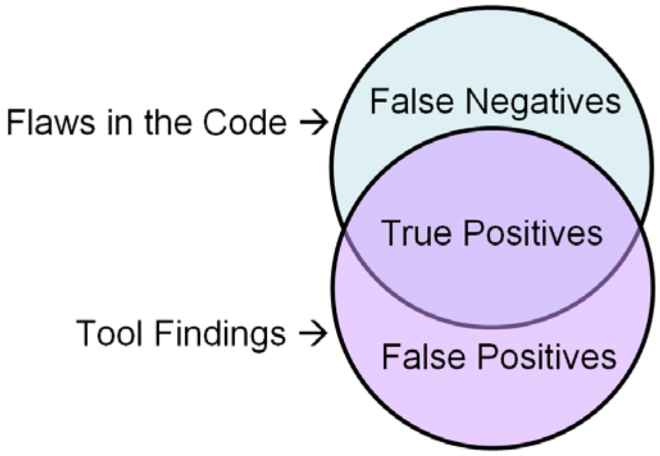

Figure 1 True vs. False Positives

False Positive (FP)

An FP, in this report, is a result that indicates one or more errors on a Line of Code (LOC) that does not have an error.

False Positive Rate (FP rate) = FPs for one tool / (TPs + FPs) for one tool

The FP rate is calculated by taking the number of FPs reported by a given tool and dividing it by the sum of the TPs and FPs reported by that same tool. This is typically displayed as a percentage.

True Positive (TP)

For this study, a result is considered a TP if the result indicates one or more flaws on a line that contains a legitimate flaw. For this report, the result is considered a TP even if

[1] The technique used in this report of trading off FP and FN/TP rates resembles the precision-recall tradeoff analysis used to evaluate the performance of fuzzy search algorithms like those used in web page search engines.

the tool mislabeled the type of the error or arrived at its conclusions through erroneous deduction.

True Positive Rate (TP rate) = TPs for one tool/All flaws in the code

The TP rate (TP rate) is calculated by taking the number of unique TPs reported by a given tool and dividing it by the total number of flaws in the code. Some codebases are fully analyzed by an expert and/or the flaws have been intentionally seeded into the code for tool evaluation purposes. For these code bases, there is a high degree of confidence that the vast majority of flaws have been identified. However, a complete expert analysis of the "natural" open source code bases in this study was impossible given the available resources. For these code bases, a proxy was needed for the denominator in the TP rate calculation. For these code bases, the union of all the unique flaws found by all the tools was used. This is typically displayed as a percentage.

False Negative (FN)

Any line of code that is known to have an error but is not flagged as having an error by the tool is a FN. Only the results from code and header files in the code base are counted. Results from system header files or libraries are ignored.

False Negative Rate (FN rate) = Flaws missed / All flaws in the code

The FN count (flaws missed) is calculated by subtracting the total number of unique flaws found by the tool from the total number of defects in the code. If the set of all flaws in the code is 100 percent, then the TP rate = 100 percent – FN rate and the FN rate = 100 percent - TP rate. Because of this, these two rates are frequently used interchangeably except that **a higher TP rate is desirable and a lower FN rate is desirable.**

Trading off FN rate with FP rate

In general, tuning a tool to make it more aggressive, in an effort to find a larger number of flaws will also increase the likelihood of mistakenly identifying flaws when no flaw exists. So, there is a tradeoff that occurs between finding the highest TP rate (lowest FN rate) and the lowest FP rate. Aggressive tools (and tool settings) find more TPs but they also find more FPs. Conservative tools (and tool settings) report much fewer FPs but miss more TPs. When tool vendors design their tools, they constantly evaluate new bug finding capability along this tradeoff continuum but many tools leave the tradeoff decision to the user.

Tool results are aggregation of checkers

The bug finding capability of a given tool is really the aggregation of the results of a number of checkers. Each checker is designed to identify potential flaws that follow a particular coding idiom. These checkers have varying degrees of success in finding TPs without producing FPs. Many tools provide the user with the ability to turn on and off checkers and thus leave the power in the hands of the user to tradeoff between maximizing the TP rate or minimizing the FP rate. With all the checkers turned on, the tool will find the maximum number of TPs, but it will also report its maximum number of FPs. Some tools also include checker setting sets as an aid for users who (at least initially) have little idea about how to make decisions along this tradeoff curve. A set may be considered, “Aggressive,” “Conservative,” “Optimal,” or “Moderate.” It is conceivable that sets may also be focused around a particular need like, “High Security.”

Micro-selection of checker sets is tedious and not always possible

The ideal picture of a tool’s performance is to capture the FP and FN rates along a continuum from the tool’s most aggressive setting to its most conservative setting. However, some tools have hundreds of checkers (which would make such an exercise tedious in the extreme), and other tools have no facility for making micro adjustments to the checkers that are used.

Filtering output by checker simulates micro-adjustment

To simulate the effect of manually adjusting checker settings and to enable micro-adjustment where no such user control is provided, the analysis in this report focused on the output of these tools. Each tool in this study provided output that identified the unique checker that was responsible for each potential flaw reported by the tool. By collecting each potential flaw into a database with an identifier for the checker that produced each finding and identifying which findings represented TPs as opposed to FPs, it was possible to design a computer program to filter the checkers one-by-one and produce all the data points for a continuum.

How are the curves generated?

To produce the curves on the graph below and used elsewhere in this report, the data set is first filtered by the criteria under interest (just C code, just C++ code, just Buffer Overflow findings, etc.) Then, the TP rate and FP rate are calculated using all the data remaining in the filtered set. This provides the first point on the curve. Then, the findings from each checker are filtered from the set one-by-one starting with the checker with the highest FP to TP ratio. As each checker is removed, a new TP and FP rate is calculated. By the time all the checkers in the set have been filtered out, the points for a continuous tradeoff curve has been gathered. Because humans like to see desirable results up and to the right, the data is manipulated and graphed so that the x-axis is really 100 percent - FP rate and the y-axis is the TP rate or 100 percent - FP rate. The point in the upper right hand corner of the graph represents the absolute best performance – the tool captured 100 percent of the flaws in the code without reporting any FPs. Conversely, the point in the lower left hand corner is the worst result – the tool captured no true flaws in the code and reported at least one FP.

Different usage scenarios determine which portion of the curve to focus upon

The data in these charts is not interpreted by simply looking for the highest curve. Different portions of the chart are useful for different purposes and different conclusions can be drawn for different needs. For instance, in the curve below, assume that the red line is the performance curve for Tool Red and the blue line a performance curve for Tool Blue. With all checkers turned on (the first point in each curve), Tool Blue finds roughly a third more TPs than Tool Red. However, Tool Blue reports almost twice as many FPs as Tool Red. Which is better depends upon the priorities of the user.

Also, depending upon the user's particular situation, the tool will not always be used with all checkers turned on. A third party evaluator using these tools might not have enough information about the code base under review to effectively discern the FPs from the TPs. In that case, the user might only want to look at the curves in the far right hand side of the graph where Tool Red has a clear advantage. The developer of the software might feel more comfortable dealing with FPs and may prefer a tool that finds the maximum number of TPs. Even then, the developer might start out using the tool with only a few number of checkers enabled and then slowly add checkers as TPs are removed and FPs are annotated away.

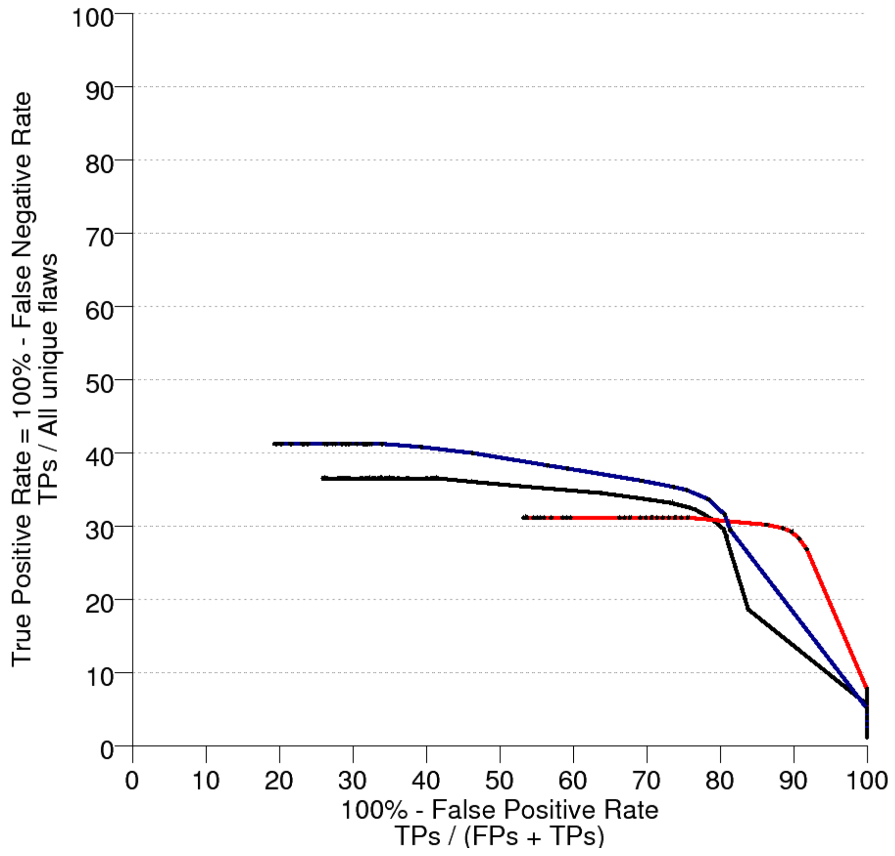

Figure 2 TPs Rate

Tool vendors design may emphasize one portion of the curve

History with simple bug finding tools like Lint has shown that developers will quickly stop using a tool if it reports a significantly high enough number of FPs. A tool vendor who is aware of this history, may make design decisions that would lower that tool's maximum TP rate but maintain a very low FP rate. At least one vendor in this study explicitly states that this is their design philosophy and the data supports this statement in that the tool generally has the highest TP rate when the FPs are zero.

It is important to look at more than one chart

The chart above might show the performance of three tools on a single code base. However, if the same three tools were run on a different code base, the results might look completely different. The data from this study clearly shows that the tools that do better on C code are not necessarily the tools that do better on C++. It might even be important to compare the performance of different tools on different flaw types and focus on the tools that do best on the flaw types of most interest to the situation.

Pareto principle applies to checkers

Each dot in the curve represents the input of a single checker. In the Tool Red curve, a large number of dots can be seen along the top flat part of the curve. In fact, it looks like there are quite a few dots where the curve does not trend downward at all. This means that the checkers represented by those dots provide no TPs – they only report FPs. However, once the curve starts to fall off and particularly after the "knee" there are relatively few dots. This means that a high number of TPs relative to the number of FPs were returned by the checkers represented by those dots. Typically, a relatively few number of checkers account for the vast majority of TPs.

Consistently high ratio checkers are necessary for third-party evaluation

If it holds true that certain checkers consistently produce a high ratio of TPs to FPs, the set of those checkers will be of particular interest for a third-party evaluation. The TP and FP rates can only be generated after all reported results are analyzed to discern the TPs from the FPs. A third party evaluator will not likely have enough knowledge of the code base to make those determinations. Having a set of checkers that is known to produce a consistently high number of TPs for a relatively low number of FPs, will enable the evaluator to make statements about the quality of the code without doing the deep analysis necessary to root out all FPs. Further work is needed to determine if such a set of checkers exists and under what conditions their high TP:FP ratio holds true.