Responsibly Reporting Performance Test Results: Trends, Noise, and Uncertainty

By **Michael Stahl** - December 24, 2018
https://www.stickyminds.com/article/responsibly-reporting-performance-test-results-trends-noise-and-uncertainty

Summary:
In order for performance test results to have value, you should report them in context. There are two main considerations: How do these compare to previous results? And how can we provide early reports on performance while emphasizing that these are preliminary results that may change significantly as we progress? Here are some ideas for responsible reporting.

Reporting performance test results is not a trivial undertaking. It's much more nuanced than simply reporting pass or fail conditions, and if everyone isn't on the same page about what the results mean and how to interpret them, it can end up causing confusion and minimizing value. That's why it's important to have **an effective method of reporting performance test results**.

But on top of reporting results, there are two additional aspects of performance test reporting that need to be addressed.

First, how do the current results compare to results from a previous drop? Is there an improvement or degradation? Putting it differently: What's the trend?

Second, some performance tests require preparation of large amounts of data, such as testing database performance or object recognition accuracy in images. At the start of the project we don't have all the needed data, but we usually already start performance testing on the partial data we do have. How can we provide early reports on performance while maintaining a clear message that these are preliminary results that may change significantly as we progress?

Trend Calculation

Let's assume we are testing the CPU utilization. A reduction in the utilization value indicates an improvement in the product's use of system resources. If the result we measure on the current release is a bit lower than the results on the previous release, can we confidently report that the new release improved the performance? It turns out this is not so straightforward.

If we run the same performance test, using the same version and running on the same platform, we won't get the same value with each test run. Instead, we will get some spread of the results. This is because parallel to our code, the system runs many other processes: OS utilities, antivirus software, network protocol that reacts to messages on the network, etc. (As I write this, there are about a hundred fifty services and processes running in the background on my PC.) Even the ambient temperature, the screen brightness, and the level of battery charge have some impact on the performance.

All this adds variance to the system, which in turn adds variance to the CPU performance. We sometime call this the measurement's "noise." As long as the difference between measurements are within the noise level, we can't say for sure that we have a clear trend.

If we execute the test many times, we can calculate the statistical distribution of the result, which in turn gives us a measure of **the process capability** of our platform. When the results' distribution is a known statistical distribution, we can be relatively confident in assessing if a new result indicates a real change. If the results are distributed according to the normal distribution, any result that falls within plus or minus three standard deviations (STDev) is within the "noise" level and can't be immediately interpreted as a real change.

There are a few cases when we can say confidently that there is a change:

- When the result clearly deviates from the regular distribution (for normal distribution, this will be a deviation of more than three standard deviations from the average)
- If the result is within the distribution range, but additional measurements on new versions of the product consistently show a deviation in the same direction

This second case needs further explanation. In a normal distribution, if nothing changed in the code that impacts CPU utilization, measurements on new versions of the code will usually fall within one standard deviation from the average. Every now and then we may get a measurement that is between one and two STDevs. In rare cases we will have a measurement that falls between two and three STDevs. In all cases, the results will spread more or less evenly above or below the average. But if measurements on new versions of the code result in measurements that are consistently above (or below) the average, then apparently a real change took place, even if all results fall within the expected noise range of three STDevs.

How many times must the measurements fall on one side of the average before we can say that the change is meaningful? This depends on the distribution and is related to the field of statistical process control.

The **Western Electric rules** define when a deviation of normally distributed parameter is meaningful:

- Eight consecutive drops of the program show a consistent deviation of less than one STDev ("consistent" means all measurements are above or below the average)
- Four out of five consecutive drops of the program show a consistent deviation between one and two STDevs
- Two out of three consecutive drops of the program show a consistent deviation between two and three STDevs
- A single case shows the measurement is over three STDevs form the average in any direction

This is all nice, but what if we have no idea what the process capability of our project is and don't have time to repeat tests dozens of times to generate this statistic?

Even if we do have results of many runs, it may be on different machines, so the machines' slight physical variabilities add additional noise. And in cases where we do know the distribution of the results, it may be an abnormal distribution, so it's not clear what rules to use to define a real change.

In such cases, we can draw on knowledge of the system and its use to decide what would be considered a significant change (e.g., "a change of 2 percent"). Once we make such a decision, it is possible to define simple rules to define what's a real trend.

My colleague Gal Steiner and I developed just such a set of rules:

1. The trend is decided by comparing the results on the current version to the results on the previous (N-1) version.
2. A trend can be no change, increasing, or decreasing.
3. If the change from N-1 is above 2 percent, the trend is increasing or decreasing (depending on the direction of the change)
4. If the change from N-1 is less than 2 percent, use the following table to decide the trend.

The trend value in version N-1	**Current version shows increase of less than 2% above N-1**	**Current version shows decrease of less than 2% below N-1**
Increasing	Increasing	No change
Decreasing	No change	Decreasing
No change	No change	No change

The basic approach that the above table implements is this: If the change of less than 2 percent continues in the same direction of a significant change in version N-1, then the change continues to be significant. If the change is in the opposite direction, then it's insignificant.

This is a simplistic algorithm and therefore imperfect. For example, it allows a small change of less than 2 percent in each progressive version to continue undetected for a long time. But the algorithm can be made better by looking also at N-2 results or by using an **EWMA chart**.

Once you have a trend metric, I recommend adding it to your presentation of the results.

Reporting Initial Results

When you are at the start of a project, it is possible that you did not yet implement all the tests by which you will determine the system's performance. However, you are already running some tests, so you would like to publish their results—but you're concerned about how they will be interpreted. You are aware that further testing may change the overall picture.

We need a method that allows for publishing the results while conveying a clear message that these are initial results and may change dramatically. Just saying it won't always help; people tend to quote the numbers you publish without the disclaimers associated with them.

To make this clearer, let's use an example.

You are developing an automation system for packaging green peppers. The system uses a camera and a computer vision module to identify peppers with defects and extract them before they are packaged. To test the system, you plan to record the video coming from the camera while a mix of good and bad peppers are passing through the packaging line. These videos will be fed into the defect-identification algorithm, and you will then check how many

of the defective peppers were identified correctly and how many good peppers were erroneously marked as defective.

These recordings take a long time, and so far you only have sixty out of the two hundred recordings you plan to do. Running the sixty videos through the system resulted in an average correct identification of 75 percent. Should you report this or not?

On one hand, it is possible that the next one hundred forty videos will have much better results, in which case you created a false alarm. On the other hand, the results on the remaining videos may be much worse, in which case you created a false sense of confidence—and an unpleasant surprise once you are done running all the tests.

The solution is to report the results but clearly articulate their inaccurate nature by using the **cone of uncertainty**.

First, let's look at the distribution of results for the sixty videos you already tested. Let's say you found that for the 10th percentile of the recordings, the system identified 58 percent or less of the defective peppers, and in the 90th percentile of the recordings, the system identified 93 percent or more of the defects. Let's assume that this range of 58 percent to 93 percent defines, more or less, our system defect-identification capability, and that in future videos this will continue to be the range of best and worst identification results.

Based on the results we already have and on the knowledge that we still have one hundred forty videos to record and test, we can give an estimation of the best and worst final results:

Best Case	Worst Case
$\frac{(140 \cdot 93\% + 60 \cdot 75\%)}{200} = 87\%$	$\frac{(140 \cdot 58\% + 60 \cdot 75\%)}{200} =$

Our report will therefore be:

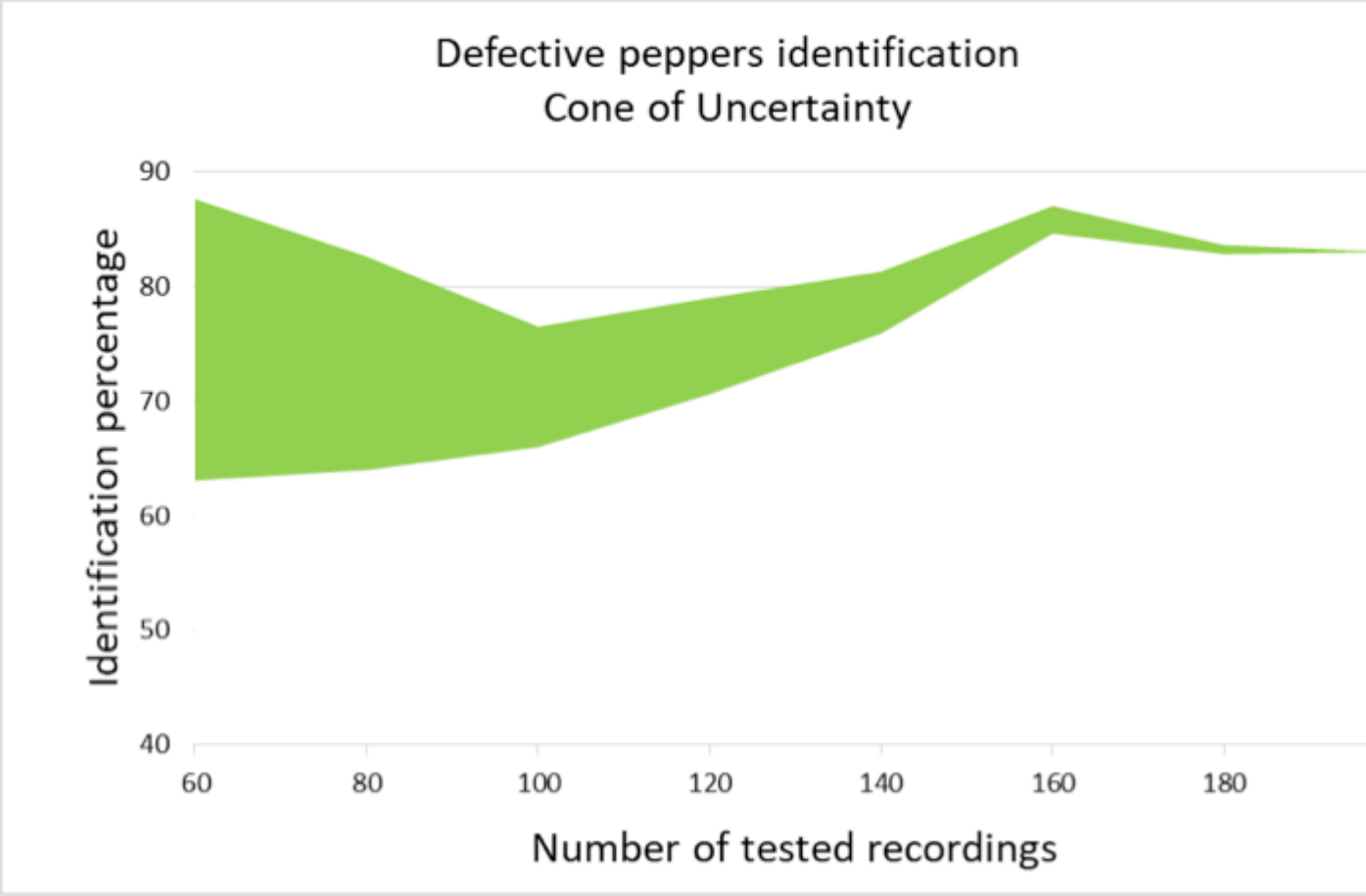
Parameter	Expected result range	Tests performed
Defective pepper identification	63% to 87%	30% [60 out of 200]

As we continue to record and test videos, we will have more real results and less uncertainty. For example, let's say that after testing one hundred twenty recordings, the 10th percentile of the results identified 67 percent or lower of the defective peppers, the 90th percentile identified 88 percent or higher and the average is now 73 percent. The resulting range estimation will change:

Best Case	Worst Case
$\frac{(80 \cdot 88\% + 120 \cdot 73\%)}{200} = 79.0\%$	$\frac{(80 \cdot 67\% + 120 \cdot 73\%)}{200} =$

Note how the uncertainty range shrunk: After sixty tests we had a range of 24 percentage points, and after doing sixty more tests, the range shrank to only 8.4 percentage points.

As we get closer to running the full set of planned tests, the uncertainty will continue to reduce. Representing the uncertainty range visually explains why this is called a “cone of uncertainty”—the graph looks like a cone that gets narrower as more information become available:



Even if identifying defective peppers is not your area of expertise, I assume that there are parameters you measure and report by taking an average of many tests. In these cases, the cone of uncertainty concept may be relevant when you are asked to report results before all the tests are executed.

Finally, a reality check. I used the idea of reporting trends. I did not use the cone of uncertainty idea, and I haven’t seen anyone else use it, either. So if you adopt it, do let me know how it works for you!