Trade publications in the software field as well as the popular media are filled with articles concerned with productivity, its measurement and how to improve it. What is not often readily available is a benchmark for what level of productivity is expected for a particular type of project. There are questions left unanswered such as: What is included in a productivity calculation? What is a “good” level of productivity? We would expect that an easy project would have a higher level of productivity than a difficult complex project. Likewise, we would expect that maintenance of a legacy system would have a different level of productivity than the productivity associated with a new development. The problem is that it is not easy to determine quantify the difference in productivity between a complex project and an easy project, or the difference between development and maintenance.

In this paper we will review a case study where an organization was faced with this very same problem: was the subcontractor adequately productive for the type of project they were doing?

Background of the situation

There was a state government agency that had a subcontracted organization maintaining one of its software systems. The state wanted to evaluate the performance of the subcontractor, so the state brought in the independent consultants assess the
performance of the subcontractor on a number of criteria. Productivity, in its simplest form is output (software produced or maintained) divided by input (resources used). Considering the nature of the performance evaluation, it was necessary to size the software and analyze the maintenance effort (determine the amount of software produced).

Considering what the state was trying to accomplish, we, as the independent consultants, needed to analyze the maintenance changes and maintenance updates. In order to produce our analysis, we analyzed the productivity of the maintenance team based on actual effort data versus the effort and schedule estimated using a commercial software estimating model, SEER-SEM. To complete our productivity assessment we considered all other available documentation in conjunction with the parametric analysis.

**History of the project**

The software project was originally built in the early 1980s primarily as a database project with some data processing. At some point the maintenance of the software was turned over to the subcontractor. This subcontractor, as the sole maintainers of the software, was required to provide training to the system users (who are state employees) and staff the help desk. They were also required to make changes to the software in the event of a regulation change, fix defects, or if there was a change required in order to increase efficiency.
In the event that a user discovers a defect in the software, the procedure was that the user would fill out a “trouble ticket”. Once the trouble ticket is submitted, the maintenance staff would categorize the trouble ticket by the solution type in a database. These tickets could be categorized as a training issue, a quick fix (small defect that can be fixed in a short period of time), or a defect that can be remedied by a Maintenance Change Request (MCR). An MCR is for a defect or defects that will take more effort than a quick fix. When a trouble ticket was resolved it was noted in the database.

From an estimating point of view, MCRs more resemble new development than they do maintenance. MCRs were solutions to problems that required a significant amount of effort to remedy. This is consistent since many of these MCRs were necessary due to regulation changes or the need for an efficiency upgrade.

Overview of the work effort
The first step was to collect relevant data. This meant collecting all documentation on each of the Maintenance Change Requests, as well as all of the recorded number of hours spent on each MCR. We made the decision to analyze completed MCRs for the most fidelity of data. By analyzing actuals we eliminated any known uncertainty associated with analyzing a project that is still in progress. This uncertainty stems from requirements creep, code growth, as well as unexpected problems or challenges that the software team faces during the development and testing of the software. From the data that was available, it was impossible to collect the number of defects. Since the only metrics available on metrics was the number of tickets was collected, there was no
indication of how many problems or software bugs there were. For example, there could be a problem that is especially visible from the user's perspective, which could mean that there were many trouble tickets associated with the one problem. There could also be bugs that were less visible to users that would not yield any trouble tickets at all. There was also a definitional issue associated with the tickets that had a solution type designated as “training”. Some would argue that some of the “training” tickets were actually defects to the software that were resolved by instructing the users on how to work around the bugs.

Once we had collected the documentation on the MCRs and the number of hours expended on the MCR by labor category, we started to analyze the data and had some very curious initial findings. MCRs which appeared to have common attributes had very different actuals. We went back to the maintenance team to ask them questions and they suggested we look at some other reports that the maintenance team didn’t realize we would need. (And we were unaware that these reports existed). Once we had collected these additional reports it was easier to understand why we were seeing the curious results that we initially saw when we started to analyze the data. From the additional reports we were able to see the differences between the MCRs that formerly appeared to be quite similar.

Collecting and assessing the data
We performed a function count in order to obtain a size of the MCRs. We were able to get a physical line count, but not a logical line count so we chose to use function based
sizing since the documentation was set up in a manner that was especially conducive to counting functions. To cross-check the function count of the software size, we did a statistical sampling of a portion of the code which we hand counted the logical lines of code. From our sample where we had the physical line count as well as the logical line count, we used the ratio between the two to get a rough idea of the count of logical lines to verify that our function count was reasonable.

Once we had the size of the maintenance changes, we separated them by the type of maintenance change (regulatory, efficiency, etc.). The maintenance team had also provided us with their categorization of whether the maintenance change was on a scale ranging from easy to very complex. We then separated the maintenance changes accordingly. At this point we modeled each of these maintenance changes in SEER-SEM to get an objective estimate of how much effort would be expected to be
associated with each of the maintenance changes.
From our initial data collection and the subsequent data collection we had enough information in the documentation to adequately chose knowledge bases for our SEER-SEM estimate. We had already sized each of these maintenance changes using function points so we entered those sizes into SEER-SEM as well to get an estimate of the effort that it should take.
Once we had our estimates for each MCR, we compared the estimate against the actual then did a trend analysis of the results. Our results were very interesting. On the large maintenance changes, (where the software size and estimated effort was high), then the actual effort was even higher. On the other hand, on the small projects (where the software size and estimated effort was small) the actual effort was even smaller. This was rather curious so we asked some more questions. We asked more questions about how the effort actual data was collected. It turns out that each maintenance change was more or less a timesheet code, which might explain why the highs were high and the lows were lower. It is possible that if the maintenance staff was idle, they knew that if they charged an additional hour or two to the large project that it may go unnoticed while that is less likely to occur on the small projects.
We also performed trend analysis on the trouble tickets submitted and the trouble tickets resolved. For the most part this analysis was inconclusive. The only real conclusion was that larger projects had more tickets written against them, which is almost intuitive anyway.

To handle the problem of what software activities are included in the calculation of productivity, we used the Software Metrics report in SEER-SEM which shows the productivity calculated in several different ways.
Conclusions

It was outside the scope of our consulting contract to assess the validity of maintaining a twenty year old system. Instead, after performing the function count and the assessment of the maintenance team we suspected that there is a possibility that the maintenance team is a bit overstaffed since they were using the timekeeping system to hide idle time. This became apparent through using a tool like SEER-SEM. SEER is a parametric model, so it is based of a large amount of actual data. SEER is also an objective tool so it was easy to objectively estimate these completed programs to compare against the actual. In light of this, as part of our conclusions going forward we
suggested some strategies for better collection of metrics, including effort actuals.

Another major conclusion was that the total number of hours expended was extraordinary. They were expending approximately 19,000 hours per month to maintain the system and staff the help desk. Using the default 152 productive hours per month, 19,000 hours is equivalent to approximately one hundred full time people. When it was time to write our final report we made note of this extraordinary amount of effort that was expended every month on this system. By using SEER-SEM we were able to demonstrate just how much development can reasonably be achieved for that amount of effort using SEER-SEM’s Design-to-Size feature. The bottom line is that through the use of SEER-SEM this whole project was much more efficient and effective.

One of the valuable lessons learned was to question any results that appear inconsistent. Often there is an explanation for why there is the inconsistency, and even documentation for it. The other lesson is that a parametric software estimating model
can be used very effectively for a productivity analysis such as this analysis. Despite
the idea that a software estimating tool is usually applied at the beginning of a software
project, the objectivity of a parametric tool makes it ideal for this type of analysis.