Nesma on productivity measurement

In this whitepaper Nesma presents information to enlarge your knowledge about productivity management and to support your activities in this area.

The information is structured around two subjects:

1. Challenges in productivity management
2. Software productivity management examples

1. Challenges in productivity measurement

Why is it hard to measure productivity in the software industry?

In other industries than the software industry, productivity measurement is a normal activity that drives the success of a company. Let's see for example a one man painting company. For a painter, it would be logical to measure his productivity in **effort hours per square meter**. Probably he wants to differentiate the measurement into some categories, like tools used (e.g. roller/brush/spray) and paint object characteristics (e.g. wall/stairs/door). When the painter builds up a database with productivity figures, he can easily quote for new painting jobs, simply by measuring the paint surface in square meters, multiplying with the proper productivity rate and multiply the result with the hour rate he asks. If there is a (international) database available with the productivity rates of paint jobs performed by other companies in the industry, the painter understands how he performs on average against the industry and in case he is not best-in-class already, he understands that to win new paint jobs he has to keep improving his productivity (as lowering the hour rate is usually not a very good idea).

He can do all this, but only when:

- He uses a ***standard measurement unit***, e.g. square meter. Only by using standards, productivity rates can be compared (benchmarked) and used for estimating;
- He uses a ***standard way to record the effort hours***. For example, is the lunch break included or excluded, is the time to talk to the customer to understand his requirements included/excluded?
- He uses ***meaningful categories*** that differentiate productivity. For a painter, it may not matter too much if the paint object (let's say a wall) is in a villa or in a fisherman's house. The type of house may not be a meaningful category. However, the tool he uses is probably a main productivity driver.

Now, let's take a look at the software industry.

The Software Industry and Productivity management

Unfortunately, the IT (and software) industry is still quite immature when it comes to using standards and when it comes to productivity (performance) measurement, benchmarking and continuous improvement. The software industry got away with that for a long time, because:

- It is difficult to measure ***output*** (software is not a physical thing, can't be touched and measured with conventional measurement instruments).
- Software projects are much more like an R&D project than manufacturing a product. R&D is incredibly hard to measure. It is relatively easy to measure the inputs, but the outputs are hard to measure and unpredictable by nature.

Now, slowly the industry is becoming more and more transparent and customer organizations ask potential suppliers more and more to quantify their performance based on historical data. This way, it becomes possible to select the best supplier for the job. Please note that the best choice is usually not the least expensive choice (often resulting in project failures…or even disasters).

Standards

While it may seem easy to implement a Productivity Measurement process in an organization, reality shows that it is more difficult than one may think. In principle, just like the painter, it is sufficient to measure **inputs** (usually effort hours) ***and outputs*** (Units of Measurement, UoM) per software project, while using meaningful categories to differentiate the projects, like technology (Java/.Net/Oracle/Etc.), project type (new development/enhancement) and/or implementation (Package implementation/modification/custom made software).

To be able to build up meaningful and comparable productivity metrics, it is critical that (international) standards are used. A number of choices have to be made:

Measuring Inputs

Some decisions that have to be made:

- Effort hours in/out scope of measurement, for instance
  - Technical design, coding, unit test, systems test, other supplier tests, overhead in scope;
  - Functional design, support acceptance test, implementation activities out of scope.
- Overtime in/out scope of measurement;
- Travel hours, meeting hours, overhead hours in/out scope;
- In case of packages, Portals/CMS or other configurable software, it may be necessary to have separate effort registration activities for customization, setting parameters and custom made software not part of the package.

To be able to analyze the productivity of a supplier, department or team, the effort registration system should be implemented in a standard way. If the choice is made that functional design hours are out of scope, all projects should register their effort of functional design separately from the other effort hours. It is strongly recommended to draw up a standard 'Work Breakdown Structure (WBS)' per project type and implement this WBS in the effort registration system.

Everybody who registers effort hours should be aware of the importance of booking his or her effort correctly in the system.

Measuring Outputs, methods that should be avoided

Measuring outputs is somewhat harder than measuring the inputs, due to the intangible nature of software. Many organizations measure the delivered source lines of code (slocs) of the software product delivered after completion and use productivity metrics like effort hours per 1000 slocs. This seems like a good way to go, but in fact there are many reasons why this is not a recommended practice:

- There is no (ISO or other) standard for Source Lines of Code. The result is that different automatic code counting tools produce (very) different results for the same code.
- It is not clear if more code is 'good' or 'bad'. Source lines of code are not of value for the customer organization. Functionality is of value. Customers never say: "Please give me 100000 source lines of code". No, it is functionality in terms of features that they require. More functionality is better and costs more; more slocs is maybe not better.
- Different programming languages (and mixes of these) result in very different source lines of code results.

American 'software metrics guru' Capers Jones wrote in a paper 'Software Defect Origins and Removal Methods (2013) that sloc measurements are so inaccurate, that using slocs in software measurement is in fact ***'professional malpractice'.***

Other size measures that are often used in the industry, but are also ***not recommended*** to use in productivity measurement:

- ***Story points (SP)*** in agile projects: A very subjective measure that only has value within one team. Comparison to other teams, departments and organizations is not possible. Please note that Story Points are useful to plan sprints and to track velocity for one team, but for productivity measurement SP are close to useless.
- ***Usecase Points (UCP)***: Only applicable when the documentation consists of usecases. UCP is also a highly subjective method, especially when it comes to establishing the Technical Complexity Factor and the Environmental Factor. Also, there is no standard way to write usecases.
- ***Complexity Points***: Subjective and not standardized method to measure the complexity of an application.
- ***IBRA Points***: Not standardized method to measure the business rules in an application. When applied according to the manual, the result is zero for all applications.
- ***Fast Function Points (FFPA)*** (by Gartner): A measurement method deployed by Gartner that cannot be compared to the ISO standardized function point analysis methods. FFPA is perceived to be a commercial method that lacks a theoretical base and is partly subjective. The method has not proven to be faster than the Nesma estimated method and has not proven to be more accurate. Unfortunately it is often pushed on higher management level without the support of the specialists who have to work with it.

Measuring Outputs – strongly recommended methods

It is a ***highly recommended practice*** to use an ISO/IEC standard for functional size measurement in Productivity Measurement of software projects. There are five functional size measurement methods that comply to the ISO/IEC standard:

- NESMA function points (ISO/IEC 24570);
- IFPUG function points (ISO/IEC 20926);
- COSMIC function points (ISO/IEC 19761);
- Mark II function points (ISO/IEC 20968);
- FiSMA function points (ISO/IEC 29881).

Advantages of using one of these functional size measurement methods for productivity measurement are:

- Objective, repeatable, verifiable, defensible way to determine the size of the software;
- A clear relation between functional size and effort needed to realize the application. This has been studied and verified many times;
- The measure is clear for both customer organizations and supplier organizations. More functionality means more value, more effort needed and a higher price;
- Functional size is independent of the technical solution and/or the non-functional requirements. An application of 500 NESMA function points realized in Java is just as big as a Wordpress website of 500 FP. This enables comparison and benchmarking over technical domains and the use of historical project data (when properly classified) in estimating new software projects.

Useful categories for data collection

To be able to compare and benchmark your productivity, it is important to use standard categories to collect data from your projects. Nesma highly recommends to use the definitions and categories that the International Software Benchmarking Standards Group (ISBSG) is using in their data collection activities.

The International Software Benchmarking Standards Group (ISBSG) is a 'not-for-profit' organization that collects software industry data and that grows, maintains and exploits two repositories: 'New developments and enhancements' and 'Maintenance & Support'. ISBSG can only do this when data is collected in a standard way. The 'Data Collection Questionnaires' can be downloaded from the ISBSG site and show already a lot of definitions and categories. Also the glossaries that are provided with the repositories are helpful.

Implementing software productivity measurement

So, just like the painter from the example above, it is possible to measure the productivity of software projects. See below for a few examples.

To successfully implement software productivity measurement, it is strongly recommended to use the document 'Basis of Measurements' that is listed in the list of Publications.

2. Software productivity management examples

Examples of productivity measurement

In this paragraph, two practical examples of software productivity measurement are given. The examples are based on effort hours, because productivity is expressed in hours. Multiplying effort by a relevant hour rate transposes the examples to cost.

Practical example of software productivity measurement 1

An organization requested its software supplier to realize a new application in the Java programming language. Although the contract was based on a time-and-material basis, an agreement was made that the supplier would carry out the project at least at a 'market average' productivity. Analyses of the ISBSG repository 'New Developments & Enhancements' made it clear that for this type of project a market average productivity of 11 hours per function point was applicable.

After project completion, the productivity was measured. The effort spent on the project by the supplier:

- Functional design: 2000 hours
- Technical design: 3000 hours
- Coding and Unit Test: 4000 hours
- Systems test:3000 hours
- Support Acceptance Test: 200 hours
- Implementation:200 hours
- Overhead (Project management, Quality Assurance): 1000

Total: 13.400 hours.

After project completion, the functional size was measured by a Nesma certified analyst of the supplier and reviewed by a Nesma certified analyst of the customer. The functional size measured was 1000 FP (Nesma 2.2).

The productivity realized in this project turned out to be 13400 / 1000 = 13,4 hours/FP.

The project was not carried out in an equal or better than market average productivity. Depending on the exact terms and conditions of the contract, the customer may be able to decide to pay only for the 1000 * 11 = 11.000 hours that are supposed to be a market average price.

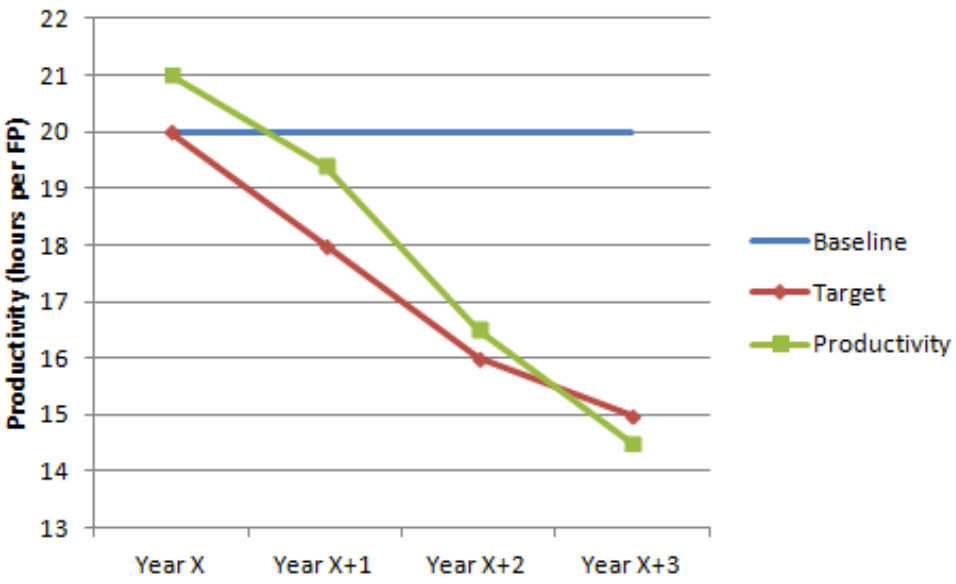
Practical example of software productivity measurement 2

An organization decided to outsource all the new releases for a specific Microsoft .Net application from their IT department to a specific external software supplier. To get a grip on the performance of this supplier, the supplier had to implement a productivity measurement process in order to show improvement over time. The following agreements were made:

- The first 3 releases of the application are measured and converted into a baseline productivity;
- All next releases are measured and productivity has to be reported by the supplier;
- The supplier has to conform to productivity improvement targets: 10% after 1 year, 20% after two years, and 25% after three years.
- The customer organization has the right to periodically review the size measurements (by an independent party) and audit the effort administration system of the supplier.

After three releases, the baseline productivity was set: 20 hours/FP.

In the next figure, the trend is shown. Please note that when it comes to productivity expressed in hours per FP, the lower the value, the better. A downward trend is therefore good!



Productivity measurement of supplier

Although the supplier did not make the agreed targets the first 2 years, productivity improved over time and after three years exceeded the target. Focus on measurement and productivity improvement resulted in faster and cheaper delivery of new functionality for the customer.