III. Evaluation

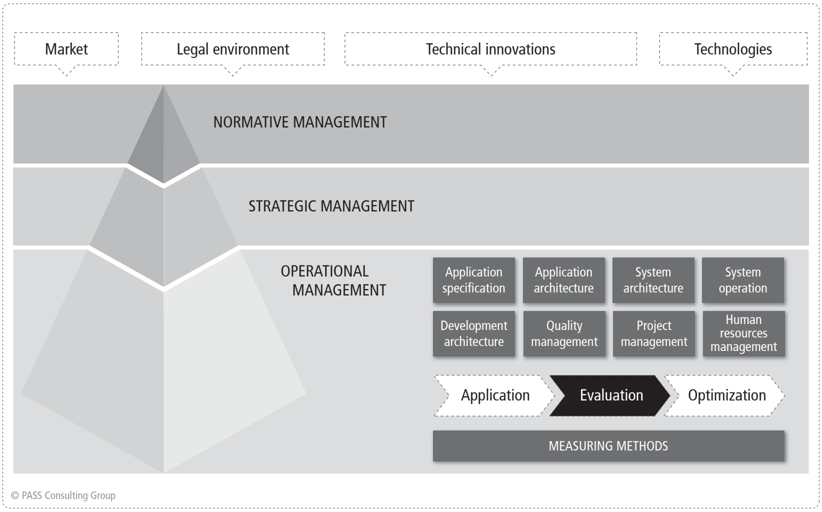

Figure 8: Evaluation of KPIs and root cause analysis

This chapter covers the evaluation of previously determined KPIs as well as a root cause analysis – as a prerequisite for purposeful improvements.

Analyzing the course of productivity over time

A first step in the evaluation of previously determined KPIs is the analysis of their time course. Usually, any continual deterioration is an indicator of underlying opportunities for saving time and costs and for quality improvements in the future. After the implementation of improvement measures, the development of the KPIs is an indicator for their effectiveness and sustainability.

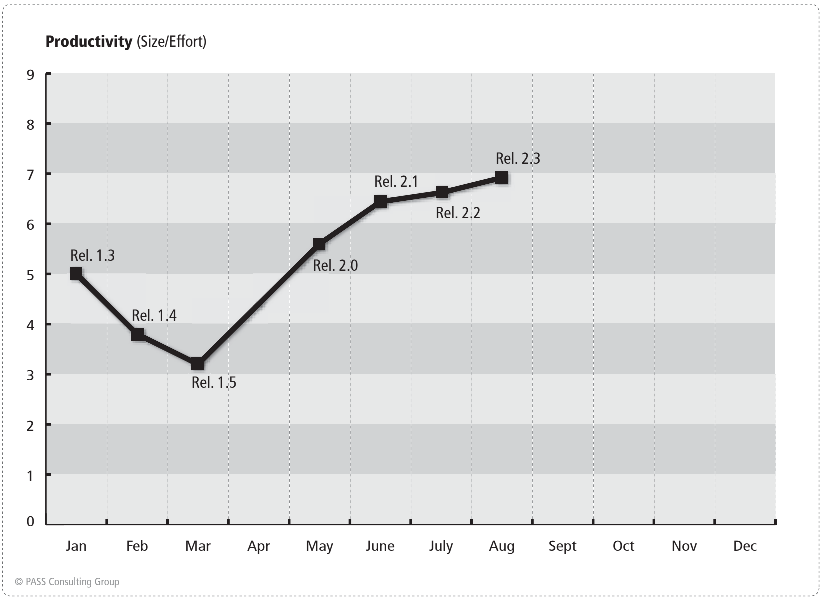

Figure 9: Course of productivity over several months (example)

Figure 9 shows the graphic representation of development productivity measured for every release of a sample system after the development process is completed and the release goes into production. The diagram clearly shows that, initially, productivity has decreased since the beginning of the year. In April and May a new major release was developed and went into production as release 2.0 in May, which included sustainable improvement measures.

It can often be observed that the accuracy of single measurements of the further development productivity is proportional to the size of the development: The smaller the functional size is, the more questionable is the measuring accuracy seems. The reason for this is that, in practice, the complexity implemented requirements often is very different, as the consideration of complexity by functional measurement methods is limited. These methods count data elements crossing the system boundaries as well as related structures in the database. In any case, a correlation between complexity and development costs is undisputed.

A good example is a small, new release of an application in which only some reports have been implemented. Reports are usually characterized by a large number of data elements crossing the system boundary and therefore being counted by functional measurement methods, which results in a large functional size. Let's assume that reporting data can be read by simple queries to the database, which requires little effort for the implementation. Both, the large functional size and the small development effort, lead to a calculated high productivity.

Another example is the implementation of a complex algorithm without user interaction, which then displays the result as one single value in a dialog field of the user interface. Because there is only one data element crossing the system boundary , applying a functional measurement method results in a low value of the implemented size. However, the implementation effort is high. Both, the small functional size and the high development effort, lead to a calculated low productivity.

When calculating the productivity of a small development scope, there is a risk that the complexity of the few implemented requirements is almost entirely higher or lower compared to the average and, therefore, the calculated productivity is not accurate – as shown in the examples above. These inaccuracies can be levelled by calculating the productivity of larger development scopes, where the requirements' complexity is evenly distributed. In practice, it has proven itself to summarize functional size and effort of all new releases of the last months in order to calculate the average further development productivity of a system:

$$\bar{P} = \frac{\Sigma \text{ Functional Size}}{\Sigma \text{ Effort}}$$

Productivity improvements or degradations become apparent if they are sustainable and have a perceptible value in relation to the total functional size. A good example for this is shown in table 2, which lists the fundamental values for the functional size S_n and the effort E_n for each new release as shown before in figure 9 in the course of productivity over time.

		S_n (DIP)	E_n (MD)	$\bar{P}$ (DIP/MD)
Rel 1.0	Okt	125	23	
Rel 1.1	Nov	23	2	
Rel 1.2	Dez	540	123	
Rel 1.3	Jan	125	15	5.0
Rel 1.4	Feb	410	149	3.8
Rel 1.5	Mrz	90	77	3.2
Rel 2.0	Mai	1,210	79	5.6
Rel 2.1	Jun	435	115	6.4
Rel 2.2	Jul	140	75	6.6
Rel 2.3	Aug	995	132	6.9

Table 2: Further development productivity by summarizing a period of four months (example)

In this example, the average productivity $\bar{P}$ is calculated by the sum of the functional size (in data interaction points) and the sum of the effort (in man days) of all releases developed in a period of four months. The average productivity in January, when release 1.3 is completed, is therefore calculated as follows:

$$\bar{P} = \frac{125 + 23 + 540 + 125}{23 + 2 + 123 + 15} = 5$$

Average productivity values calculated this way have proven to be resistant against outliers as can be seen, for example, in release 1.1 which would have a productivity of 11.5 in case of an individual consideration (results from a functional size of 23 DIP divided by an effort of 2 MD). Another outlier would be release 2.2 with an isolated measured productivity of 1.9 (results from a functional size of 140 DIP divided by an effort of 75 MD). Not until multiple individual measurements have been summarized, does the sustainability of the improvement become apparent, which was in effect since release 2.0. This is not detectable when looking at the time course of individual measurements only.

The period of 4 months for the summarization of further developments, which has been used in the example above, is sufficient to illustrate the mentioned effect of levelling the inaccuracies of smaller development scopes. Depending on the release cycles, different periods such as a complete year (n = 12) can be useful.

Internal benchmarks

In addition to the informative value of a single system's course of productivity, the comparison of different systems or organizational units can be useful, as figure 10 shows.

The basis for the comparability of values measured with different systems is their independence from technical characteristics and the orientation of the measurement methods towards use cases. This is generally given in case of functional measurement methods as the Function Point Analysis, the COSMIC method and

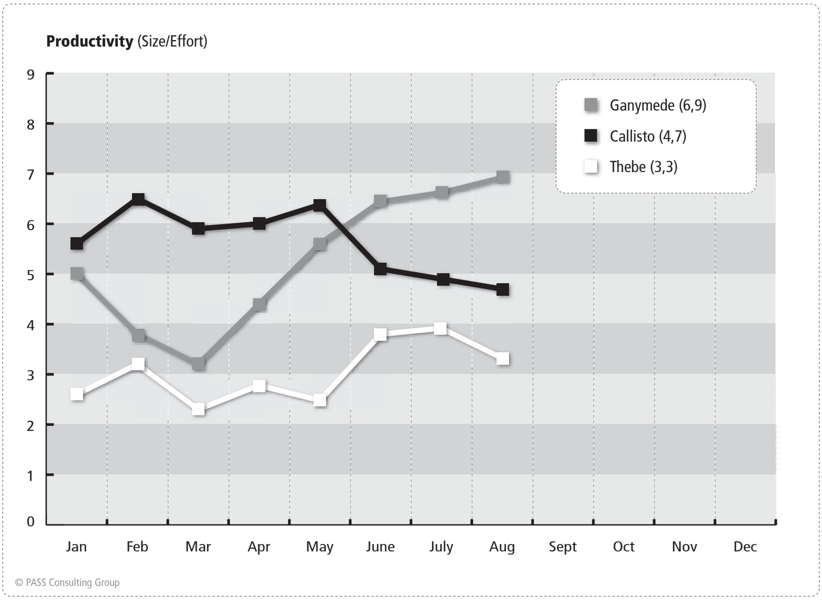

Figure 10: Internal productivity benchmark (example)

the Data Interaction Point method. Comparisons of systems or IT shops of the same organization, also called internal benchmarks in particular, help to identify the most productive teams or organizational units, also called „the best in class", others can learn from. This requires an analysis of their success factors and a check of the transferability to other teams or shops. If an organization is able to promote an internal improvement competition based on transparency and openness for their valid measurements and without assigning blame or punishments, this will result in high planning and scheduling reliability perceivable by the market, as well as in additional cost reduction effects.