Using Six Sigma for Software Project Estimations
An Application of Statistical Methods for Software Metrics

Dr. Thomas Fehlmann

Euro Project Office AG, Zurich, Switzerland

thomas.fehlmann@e-p-o.com

Abstract:

Six Sigma is a proven way of removing defect cost in manufacturing and services. However, using Six Sigma for Software is not straightforward. Many metrics that are collected during software development are not following standard distributions and thus statistical methods won't apply easily.

However, estimations for software development projects follow statistical patterns. This is less surprising that it sounds. When developers estimate their future work, their mood and their most recent experiences have impact on their estimations. These factors follow random pattern but most people are not inclined towards admitting such influence. Thus statistical methods based on comparisons between effort predictions, or effort measured, and other measurable cost drivers can detect and reduce variations in estimations and avoid wrong predictions.

This paper describes experiences with a Six Sigma model for effort prediction.

Keywords

Six Sigma for Software, Project Estimation, Statistical Process Control, Functional Sizing, Function Points Analysis, COSMIC Full Function Points.

Zusammenfassung:

Six Sigma ist die weitverbreitete Methode für die Verbesserung von industriellen Fertigungsprozessen geworden und ist auch für die Software-Entwicklung von Interesse. Allerdings ist es schwierig, unter all den Metriken, die man während der Entwicklung von Software sammeln kann, solche zu finden, die einer Standard-verteilung gehorchen und damit für statistische Methoden in Frage kommen.

Allerdings gehorchen ausgerechnet Aufwandschätzungen statistischen Gesetzen. Das ist weniger erstaundlich als es scheint. Wenn Entwickler nach dem erwarteten Aufwand gefragt werden, folgen sie ihrer gegenwärtigen Stimmung und schätzen je nachdem zu hoch oder zu tief. Dies gehorcht den Gesetzen des Zufalls, was ungern zugegeben wird. Deswegen können statistische Methoden, welche auf Vergleichen zwischen Vorhersagen und Messungen für Aufwand und andere massgebliche Kostentreiber beruhen, Schwankungen in Aufwandschàtzungen aufdecken und vermindern.

Schlüsselbegriffe

Six Sigma für Software, Aufwandschätzung für Projekte, Statistische Prozesskontrolle, Funktionale Grösse, Function Points Analysis, COSMIC Full Function Points

1 The Prediction Model

1.1 Sizing and Estimations

Barry Boehm said in 2006 [3]: T*he two primary factors that influence software cost are the size of the project and the people involved in the project. [...]*

The people factor revolves around capability, experience, collaboration, and retention. You'll usually have a productivity factor difference of over 10 if you're doing the same job with qualified as opposed to unqualified people.

This is not only true for the qualification of developers, it even more holds for the qualification of customers, and organizations, to capture and formulate requirements, and able to acceptance testing of the project.

1.1.1 Functional Sizing

Several measurement methods exist that conform to the International Standard on Functional Size Measurement ISO/IEC 14143, among them ISO/IEC 20926, known as IFPUG Function Points (unadjusted), and ISO/IEC 19761 COSMIC Full Function Points 3.0. Not all functional size measurement methods measure the same; it depends on the selected Viewpoint (User Functionality, SW Architecture, algorithmic design, Data acquisition, …); under certain conditions it makes sense to combine several of the measurement methods to assess functional size under different viewpoints (e.g., see [5]).

There is an well-founded believe that functional size has an impact on effort. Thus, functional size is often understood as the primary cost driver for software development, and for software maintenance and operations as well. The more functionality is in use, the more load will be generated on Help Desk, Second Line Support, and Software Maintenance.

However, the exact impact factors are rather hard to estimate since there are many candidates.

1.1.2 The IFPUG General Systems Characteristics (GSC)

The IFPUG method transforms Unadjusted Function Points according ISO/IEC 20926 into another metric called 'Function Points'. This transform, called the 'Value Adjustment Factor', is a framework to accommodate added value originating from technical or performance constraints. For instance, if a software supports multiple users at once, this adds value to the software and this is reflected in the final Function Points Count. However, in practice it is not well defined when and how to determine the valid values for the General Systems Characteristics for some given project. It's guesswork. Therefore it has been exempt from the ISO standardization attempt.

No such transform is needed with the ISO/IEC 19761 COSMIC Full Function Points 3.0 method.

1.1.3 The Project Delivery Rate

The key metric needed to derive estimations out of functional size is called the ***Project Delivery Rate*** (PDR). This indicator means the number of hours needed to implement one Function Point. Typical values for the PDR are between 5 and 20 hours/FP. In practice, some guess for the PDR is the base for estimations. However, how is it possible to establish the true value?

1.2 Impact Factors on Estimations

1.2.1 Functional Size as Cost Factor

Functional Sizing is very important and useful for many purposes, but it is only the first step for estimations. Functional Size is the main cost driver for new developments but only after requirements are known. In commercial and customer-driven environments, requirements typically evolve during development, adding functional size at an unpredictable rate. For embedded and mission-critical software, requirements are usually better known but subject to Change Requests as well.

Moreover, Functional Size increases during the early inception phases of a software project, and decreases as a result of business decisions that for instance impose budget constraints on development ideas. Also, a profound analysis of business requirements also has the potential to decrease functional size. All this makes it quite difficult to establish the PDR generally valid for new software development, or re-use, since the complexity and the organizational environment factors typically are unknown.

Furthermore, for maintenance projects, functional size is not always the major cost driver for maintenance tasks. First of all, we must distinguish whether we measure the size of existing functionality, of changed functionality, or of added functionality. Second, it very much depends from the state of the code that needs being maintained. The actual coding activity may be such a slight fraction of the overall maintenance task that even the code size touched is not very indicative for the effort needed to agree on what needs to be done: analyze the requirements, identify the changes needed in the design, test the result and implement the changes in the production environment.

Especially predicting effort for testing is difficult. A single slight change that counts for almost no changed functionality may require adjusting a huge amount of regression tests. On the other hand, complicated business cases that extend an existing system may be easy to implement because of re-use .

For these reasons, neither IFPUG nor COSMIC Function Points are very well suited to predicting efforts for SW Development or Maintenance if not combined with some additional statistical methods that take care people's influences. Functional Sizing alone is more useful for scope management during such projects.

1.2.2 Estimation based on Benchmarking

The International Software Benchmarking Standards Group (ISBSG) collects effort data since a dozen years from leading software development organizations and publishes the data as a base for benchmarking. From this data, it becomes apparent how different the PDR can be, depending upon industry, application area, target project, architecture, re-use, and many more characteristics, including software process maturity levels, see [8]. However, the problem remains that fitting a project into these criteria is subject to individual evaluations and metrics; not to measurements.

1.3 Estimations based on Cost Drivers

1.3.1 Barry Boehm's COCOMO

To cope with such shortcomings Barry Boehm's introduced in 1981 the COCOMO range of prediction models. He identified a number of cost drivers.

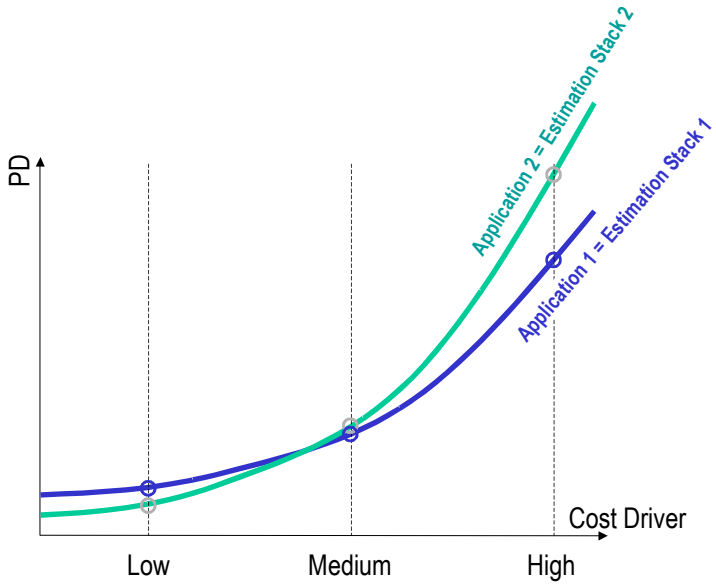

Figure 1: Cost Drivers with different slopes

Each of those cost driver functions has a different slope that models how the cost driver influences overall effort. Boehm used some kind of General System Characteristics such as: Requirements Volatility, Functional Sizing, Technical

Complexity, Impact on Current Application, or Communication Needs, and so one. These cost drivers are intuitively easy to understand but behave differently in the Design & Development development stage, or for Application Testing, or Documentation. Moreover, the cost drivers may behave differently among different products. Total effort prediction is a function of these cost drivers.

These cost drivers predict the effort needed to implement the project tasks. Boehm characterizes each criteria by a Cost Driver Profile. Users of the model rely on a discrete scale marked with "Low" – "Medium" – "High" as their limits that characterize the cost driving force. Medium always means average. Thus there is no need to give absolute values; as with functional sizing, we only need to compare.

The impact of cost drivers varies among products and even among the development stages. Thus, as Figure 1 shows, we may have different impact on the cost drivers, depending on the product.

1.3.2 Calculating Cost Driver Profiles

If reliable data is available, calculating Cost Driver Profiles is mathematically easy, using the statistical functions known from Six Sigma. If data is reliable, multiple regression analysis yields the cost driver functions. The data ideally consists of actual effort spent on the various phases in software development with known Cost Driver Profile. Such data allows calculating the *a*-parameter of the cost driver using multi-variant regression analysis with the Least Square method.

1.3.3 The Estimation Formula

Barry Boehm uses exponential functions for the impact of Cost Factors:

(i)
$$g(x) = e^{a*x}$$

where x is the cost driver and a is some parameter (later referred as *a*-parameter) that defines the impact of the cost driver x; represented as steepness of the slopes in Figure 1. The reason for taking an exponential function is that it nicely corresponds to the dimensions we experience in software development. The difference between low impact and medium impact is much less than between medium to high impact. High impact has almost no upper limit, whereas low impact always has a limit.

Barry Boehm combines those individual cost driver effects by multiplication with an exponential factor:

(ii)
$$PI(x) = \prod_{i=1}^{n} g_i(x) = e^{a_i * x_i}$$

where n is the number of cost drivers, and $PI(x)= PI(<x_1, x_2, ..., x_n>)$ is the total cost influence depending from people, dependent from the cost driver vector $x=<x_1, x_2, ..., x_n>$. We call $PI(x)$ the ***Impact Function***. The PIs therefore can be predicted based on the a-Parameter Vector $<a_1, a_2, ..., a_n>$.

1.3.4 Combining with Functional Size

If the model contains more than just one Cost Driver that depends from Functional Size, we cannot use the above formula. Since modules should not interfere with each other, an additive model is more appropriate than the multiplication of factors as proposed by Boehm. Let $y_i = f_i(x)$ where the index i runs over all functional cost drivers. Then the functional contributions y_i sum up:

(iii)
$$FD(x) = \sum_{i=1}^{n} f_i(x) = e^{a_i * x_i}$$

Note that we still use the exponential factors a_i*x_i that transform a functional size measurement into some Cost Driver. The ***Calculated Effort*** in Person Days (PD) for an estimation item x is therefore

(iv)
$$PD(x) = FD(x) * PI(x)$$

It means that the People-related cost factors behave similar to the Value Adjustment Factors (VAF) in IFPUG 4.2: They multiply with the original functional size based metrics.

If we use but one single functional cost driver, we don't need to distinguish the additive functional cost drivers from the non-functional cost drivers, and the estimation formula comes back to Boehm's original form (ii).

2 The Estimation Process

2.1 The Estimation Environment

2.1.1 Estimation Items

Estimations are needed at various points in the software development life cycle. Typical estimation items are Use Cases in the SW project. We do not benchmark as with IFPUG or COSMIC-based estimation but we separate into the work breakdown structure (WBS) that experts understand for their expert estimation. A functional measurement is a good step to find such a WBS.

Requirements Elicitation, Design & Implementation, Testing, Documentation, and Project Management activities are separate estimation items. The cost drivers

parameters for such activities can be different from cost drivers parameters that impact development of software for instance. Cost drivers for testing in a maintenance project can be completely independent from the amount of added code to the system – for instance with mission-critical software.

2.1.2 Estimation Points

Initial Estimation, Mid-Term Estimation, and Final Estimation are separate estimation points.

The goal of Initial Estimation is to get a quick and early estimate for the respective feature or Use Case. It is used to understand its impact on the applications and provide some guidance for gating the request, i.e., allowing the request to pass the next tollgate in the development process.

Mid–Term Estimations take synergies with other planned features or Use Cases in consideration. That might significantly impact overall development effort, as often additional features can be implemented "for free" when done together with other required features. Thus we expect that estimate being lower than the original Initial Estimate.

The goal of the final estimations is to establish a release plan containing the said NFR and eventually establishes the baseline for its implementation.

All *a*-Parameters are computed extra for each of the fifteen estimation points Analysis, Design, Implementation, Testing, and Project Management at Initial Estimation, Mid-Term Estimation, and Final Estimation.

2.2 Estimation Stacks

2.2.1 A Collection of Estimation Items that share the same Characteristics

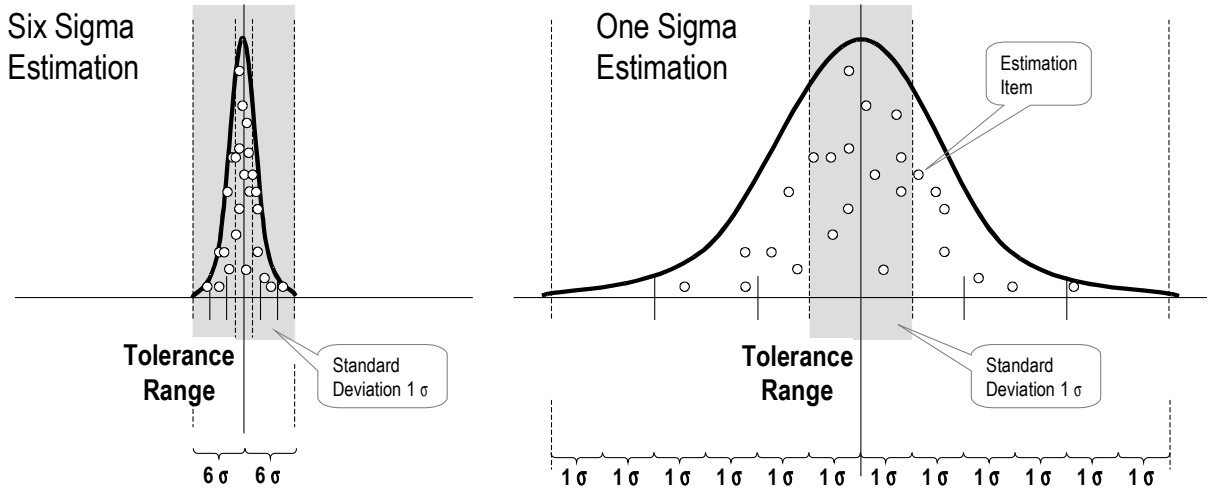

Figure 2: One Sigma and Six Sigma Estimations

A set of estimation items that belongs to the same product and has similar characteristics, as related to estimations, is called an ***Estimation Stack***. An estimation stack consists of a group of estimation items that belong to the same application, using the same technology, system environment, and development methodology. Within an estimation stack, we expect identical *a*-Parameters. Its cost drivers characterize an estimation stack.

Estimation Stacks can be identified by statistical means. If estimation items differ too much within an estimation stack, the variance increases, and the confidence interval shrinks, see Figure 2. In such case the model helps finding the criteria to split an estimation stack into two, see Figure 3. Such additional criteria must be recognizable by people who use the prediction model.

An estimation stack must contain sufficient estimation items with different impact for every estimation item. It must contain items both with high and low functional sizing, high and low technical complexity, and so on, for calibration.

2.2.2 Splitting Estimation Stacks

However, samples may not fit at all to standard distributions. If this is the case, we suspect a change in the process, be it introducing new technology, or different patterns reflecting a change in business.

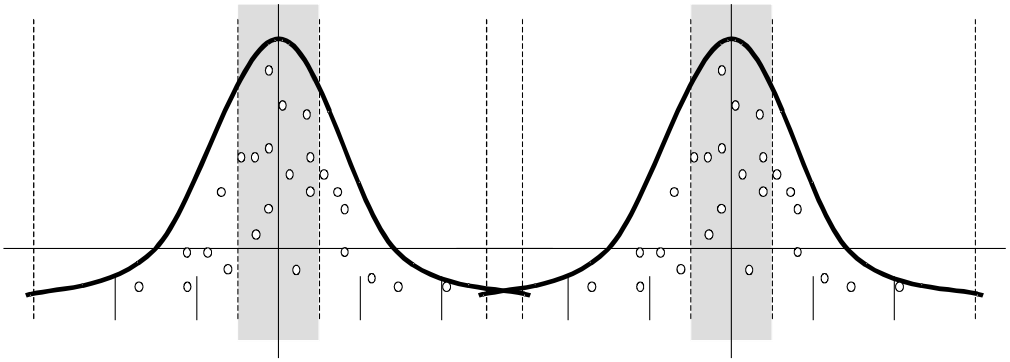

Figure 3: Split estimation stack in two – ideal case

In this case the deviations typically look as seen in Figure 3. Ideally, if enough data is available, we see two clusters in the deviation points, and can identify what characteristics correlate to these clusters.

If this happens, we must split the estimation stack in two or more. The model splits the clusters based on data only. The estimation expert has to identify the characteristics of the split.

2.3 Calibration

2.3.1 Variance in Estimations

Every estimation stack has its own set of cost driver parameters. These parameters characterize an estimation stack. Multiple Regression ensures that the differences between the Calculated Effort derived from the cost driver parameters a_i (i>0) and expert estimation is minimal.

The statistical variance σ is the square root of the sum of square differences between expert estimation and what the estimation model predicts [4].

This variance, if small, says that the cost driver parameters are clearly distinguishable from statistical noise[1]. However, if statistical variance is high, then this means that the cost driver parameters are all but well known. It might be in particular that several estimation items in the same stack follow different cost parameters or follow them in a different manner, thus making it advisable to separate the stack into several groups. The number of such groups can be found using regression analysis.

2.3.2 Trust in Estimations

The statistical variance σ is a trust indicator for estimates. If variance is high, estimates can only be trusted within a range of 6*σ, on both sides of the median. If variance is low, 6*σ is low too; and the estimation is trustworthy.

The expert doing an expert estimation, be it activity–based or be it based on some other estimation method, will see this picture that indicates where his new estimate fits into the previous sample. He can decide whether he wants to add his new estimation to the sample contained within his estimation stack or split the estimation stack until his new estimation better fits with the rest.

2.3.3 Calibration Method

Estimations are collected within an application. Thus several estimation items can be compared using their Cost Driver Profiles. If they correspond, we expect a similar estimate. Furthermore, we expect an estimate be higher within the same application, if it at least growth in one dimension against the other.

We can compare with other estimates. Having consistent estimates is already a big step forward to more transparency, and calibration does not need the life

1 Actually, we use the convention of taking twice the difference of the distances from the median. Thus

$$\sigma = \sqrt{\sum_{i=1..n} (e_i - \bar{e})^2}$$

where e_i are the individual estimates and $\bar{e}$ is its median.

cycle time of many releases. Consistent estimates are also much easier to compare with actuals, thus it adds value concentrating first on estimates and do the calibration with actuals in the second step only.

If estimation items are traceable until actual values have been collected, and those values are comparable with the original estimated items, then actual values may also be used for calibration. Actuals override estimations for calibration purposes.

Mathematically the procedure is simple: the n–tuple $<a_1, a_2, a_3, a_4, ..., a_n>$, called *a*-Parameters, is optimized such that the Calculated Efforts fit the sample. Any linear optimization algorithm can do the job.

2.3.4 Maintaining Estimation Stacks

The estimation process consists of two steps

- Calibration
- Use

Initially, an estimation stack must be calibrated. Initial calibrations are usually based on a set of expert estimations. If no historical data are available, the start is with one or several actual projects. We collect as many expert estimations as possible. It may even be needed to do estimations just in case to get initial calibrations for the estimation stack.

As experience growths with an estimation stack, more data, including actual data may become available that helps to improve initial calibrations. Because the threshold values for the Low/Medium/High scale do not change, previous Calculated Efforts may change when re-calculating with new calibration data.

2.4 The Calibration Algorithm

2.4.1 The Least Square Method

The Least Square Method is the usual one that brings best results for regression analysis. It uses linear optimization based on Runge-Kutta's original method stating the minimum function as the total vector difference between the Macro and the Expert Estimations. We must find optimum values for the *a*-Parameters $<a_1, a_2, a_3, a_4, ..., a_n>$ such that the following difference is minimal:

$$\sqrt{\sum_{j=1}^{m} \frac{\left(a_j - PD(x_j)\right)^2}{m^2}} = Minimum!$$

where j = 1…m ranges over all NFR samples that have both a Expert Estimation a_j, and a Calculated Effort $PD\ (x_j)$ for each corresponding impact analysis vector

$x_j = <x_1, x_2, ..., x_n>$. Obviously, we don't need taking the square root for the minimum, as all values are positive; it's only stated for theoretical reasons here.

2.4.2 Multiple Regression

The calibration algorithm uses multiple regressions to find the optimum solution. The n–tuple $<a_1, a_2, a_3, a_4, ..., a_n>$ are the variables for the regression analysis; the goals is to minimize the total weighted difference between the Calculated Effort predicted with the model, and the expert estimations. The algorithm normally used for this is to analyze the effect of adding/subtracting deltas from the a_i, and to retain them when the result is better in the sense of the optimization goal. Therefore it always results a better approximation of the parameters than before; only when adding more sample to the estimation stack, you may have to re-iterate calibration.

2.4.3 Limitations of Multiple Regression

However, with multiple regressions we only can find local minimum spots in our *m*-dimensional vector space spanned by the samples from the estimation stack. It cannot be guaranteed that such a minimum is a global minimum. Thus, a better solution for the minimum set of *a*-Parameters may exist, and sometimes the system is not stable but oscillates between two solutions.

2.4.4 Finding the Optimum

The "*a*"–parameter are not intuitive, they just model the effectiveness of the respective cost driver. However, it helps to know that the higher the value the stronger the effect. Thus you can perceive which cost driver cause most in which development phase. The target values, that must be minimized to find the optimum, are intuitive numbers neither; they are functions of the "*a*"–parameters and the cost driver profiles and serve as metrics for the deviation of the sample estimates, as used for calibration, from the median of all estimations. For more intuitive data, see the histogram as in Figure 4: Estimation Stack Overview.

We find the optimum by increasing or decreasing the "*a*"–parameters and verifying in which direction the targets are affected. If decreasing one of the "*a*"–parameters also decreases the target value, we continue doing so until it no longer has any effect, then we go to the next parameter. We repeat doing this until we can no longer get any perceivable result. This algorithm finds a local optimum, however it may fail finding all possible solutions in case the estimation stack has more than one peak on the deviation histogram, as shown in Figure 3.

3 The Selection of Cost Drivers

3.1 A ICT Service Maintenance Example

3.1.1 The Environment

A typical set of Cost Drivers for large organizations is

- Functional Size, summed up for all modules involved;
- Requirements Volatility, measurable by the number of change requests;
- Technical Complexity, measurable by impact on other modules;
- Communication Needs (number of internal and external stakeholders);
- Team Size;
- Time Constraints (usually the only metrics known);
- Availability Requirements, measurable by MTbF[1];
- Reliability Requirements, measurable by requested test coverage;
- Documentation Requirements.

It is not needed to establish a Base Count for the modules involved or to count the functional size of the changes; experience did show that with a very rough Low – Medium – High assessment of how much change is expected for each , results became quite good.

3.1.2 The Experiences

In an application for a large Telecom organization, the model turned out to be surprisingly useful and long living. However, it was not possible to use actuals for calibration. Estimation items seldom reached implementation unchanged. The model helped reducing variance in the expert estimations, and providing them easy access to previous estimations in turn did add quite some value. For this reason, the model is in use since five years .

3.2 A High-Maturity Software Development Example

3.2.1 The Environment

The environment is a high-maturity software development company that created a line of software products for personalized customer communications, mixing transactional (e.g., bills, statements,...), and promotional (“TransPromo”).

Actuals were collected over years according a stable process based on defined criteria, and the features actually being implemented were those planned. People worked under stable conditions and with known technologies. Very few short-term hot fixes or urgent feature requests did have impact on the development

1 MtbF = Mean Time between Failures

plans, and thus the implemented features were those once planned and the original estimate for the feature remained comparable with the implemented feature. We decided to use the model based only on functional size, using the formula (iii), without taking any multiplicative cost driver into consideration.

The Functional Sizing practice that was available was based on Lines Of Code. It was thus very simple to establish criteria for Low – Medium – High impact on each of the 30 constitutional modules units that made up the product. We had some 50 features with Low – Medium – High profile available.

3.2.2 The Experiences

The model was immediately capable of reproducing the expert estimations and within a few calibration runs provided much better estimates for implementation estimation points than the experts before could do.

We could verify the predictions with actual values and identified outliers from the beginning. This means, it became apparent which features have a high risk of suffering delays during implementation.

Other, more recently added products such as web applications had lees modules but did yield excellent results as well.

The model proved useless for predicting Quality Assurance activities or documentation. These activities simply don't depend from code size.

Despite its technical brilliance, the model is no longer in use. Developers were utterly reluctant to believe its predictions (statistical metrics didn't help creating trust), and management must rely on developer's commitment. The statements of a prediction tool, even if proved right, are less regarded.

3.3 Combining the experiences

Statistical methods allow not only predicting, but much more: understanding software development. Basically the multiple regression mechanism identifies the true dependencies between the various classes of influence methods. The model supports any combination of functional sizing (which must be additive), and cost drivers (which are used to multiply the size).

If measurement methods are available, as for instance for the functional sizing, we can replace the Low – Medium – High evaluation by measurable values, and get even more deeper insight into the true nature of software development.

Moreover, you can combine IFPUG and COSMIC measurements from different viewpoints and obtaining objective evidence which viewpoints contribute the most to total cost, using formula (iii).

We can use the IFPUG General System Characteristics (GSC) as multiplicative Cost Drivers, using formula (ii). Doing this, you no longer have to guess the

Value Adjustment Factors; the model calculates them from actuals, once he development process is stabilized enough, using multiple regression.

If you're not satisfied with the IFPUG GSC, you can try others and the model tells you whether your choice is better or not.

The statistical model supports analyzing cost factors for different influence sources. It can be used for determining what process area influences success in software development or maintenance projects; it supports organizational learning.

4 A few Notes about the Tool

The tool we used in both cases is implemented as an Excel prototype of size 261 UFP, which allows for some limited multi-user activities, since every Estimation Item uses a separate Workbook for the expert estimations and the Cost Driver Profile. Experts could thus work on their workbooks and results were collected only later for the prediction model.

For the first user, the large Telecom company, plans existed to make it a web-based application using Ajax technology, with a total of 640 UFP (included 264 UFP that were not yet well specified and thus only roughly estimated), but despite we offered everything at a fixed cost per Function Point (UFP) of € 559, it was not ordered.

4.1 The Estimation Stack Overview

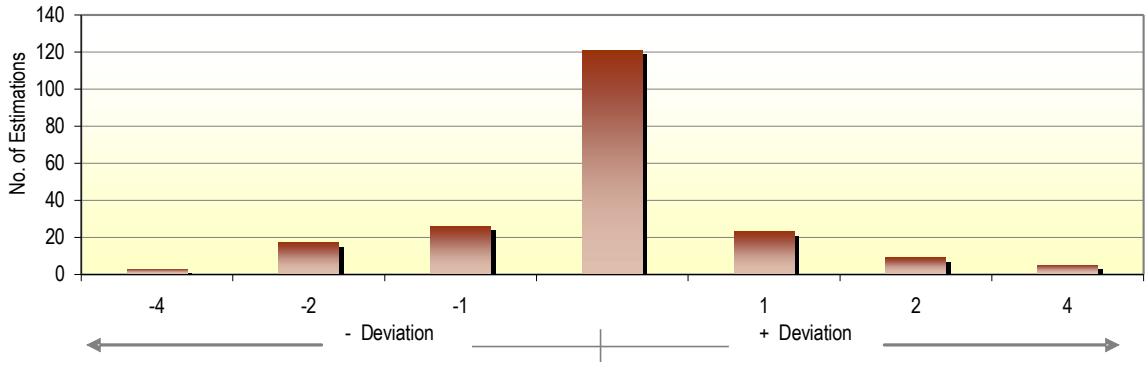

Figure 4: Estimation Stack Overview

The Estimation Stack Overview displays all estimation items used in the estimation stack. The graphics Figure 4 is a histogram that shows the deviations of the Calculated Efforts, based on the cost driver profiles, from the expert estimations used for calibration. The overview serves as a command panel for using the estimation tool; you can create a new estimation item sheet and add it to the stack, or remove an existing estimation item, or start calibration with all data that are approved for calibration.

Approval is also required when actuals are available. Actuals can be disapproved when evidence exists that the original cost driver profile is not valid for the particular estimation item under consideration.

Not all Cost Parameters have impact; if some modules are not touched at all by the requested new feature, they do not contribute to the predicted effort.

Typical success rates show significantly better results than for traditional benchmarking methods.

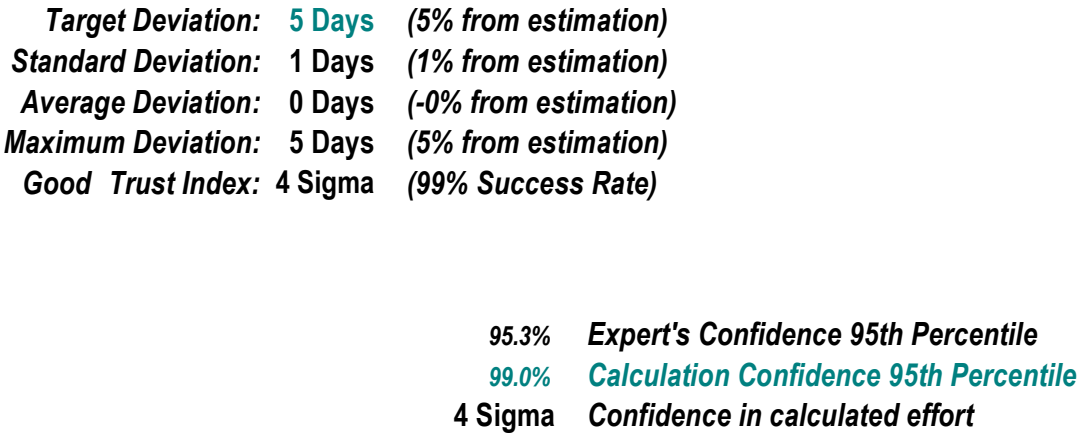

Figure 5: Typical success rate

4.2 The Estimation Dashboard

Experts using the tool have two tasks:

- Set the Cost Driver Profile for the Estimation Item
- Provide an expert estimation

Since the cost driver profile already yields some calculated effort from the prediction model, the expert does already know what the model expects from him. Nevertheless, he's free to overrule the model and give it another direction, because every new estimate added to the model, when approved, impacts the model predictions.

The expert records the Cost Driver Profile by activating buttons, where he has only the choice between No impact – Low – Medium – High impact. He then enters his detail estimates for the various phases of the estimation item: usually REQ – DEV – TST – DOC, and optionally PM. Next he evaluates the risks and enters probabilities and impact, following the standard risk catalog. The model then compares the detail estimation with the prediction and results are summed up for all estimation items in a stack..

4.3 Statistical Metrics

The statistical metrics calculated for the above example confirm that calibration was successful. We us the 95th percentile for confidence intervals, because multiple regression should be able to match actual data that good. The barrier for expert estimations is thus already quite high. In simple word, 95th percentile confidence means that 95% of our samples are within tolerance limits. We always reached values between 95% and 99%; initially, the model always performed significantly better than the experts. However, after some time, the experts learned to follow the advice of the model, and became as good as the model.

5 Conclusions

Statistical methods show significantly better results than other estimation methods. They are very useful for research and analysis, and indispensable for improving ICT performance using Six Sigma. Such studies provide deep insights in what the actual cost factors are that impact ICT. For the average user, commercial tools are available that implement similar statistical techniques to a large extend, and are easy to use..

References

[1] Abran, A. et. al. (2003), COSMIC-FFP Measurement Manual - The COSMIC Implementation Guide for ISO/IEC 19761: 2003, Version 2.2, The Common Software Measurement International Consortium (COSMIC), Montréal, Kanada

[2] Barry Boehm, COCOMO II, Addison-Wesley, 2002

[3] Barry Boehm, IT Metrics and Productivity Journal – Special Edition 2006, Computer Aid, Inc., Allentown, PA

[4] Fehlmann, Th. (2005), Six Sigma in der SW–Entwicklung, Vieweg–Verlag, Braunschweig–Wiesbaden

[5] Fehlmann, Th. (2006), When use COSMIC FFP? When use IFPUG FPA? - A Six Sigma View, IWSM/MetriKon 2006, Potsdam

[6] International Function Points User Group IFPUG (2004), IFPUG Function Point Counting Practices Manual, Release 4.2, Princeton, NJ

[7] International Software Benchmarking Standards ISBSG (2004), ISBSG Estimating, Benchmarking and Research Suite R9, Hawthorn, Victoria, Australia

[8] CMMI Product Team (2006), Capability Maturity Model Integration for Development, Version 1.2, Software Engineering Institute, CMU/SEI-2006-TR-008, ESC-TR-2006-008, Carnegie-Mellon University, Pittsburg, PA