Uncertainty of Software Requirements

Thomas Fehlmann, Luca Santillo

Abstract

Uncertainty is a measurable notion in statistics and measurement theory. Also, uncertainty typically propagates from one stage to the other in a measurement or estimation process. Software Requirements constitute an event space following standard distributions. Thus statistical concepts such as residuals, standard distribution, and uncertainty – and uncertainty propagation – are applicable from statistics and measurement theory.

This work presents preliminary results that allow evaluating the benefits of pursuing such a research direction for Six Sigma practice. We present the results of a field study where we compare the life cycles of new requirements to Personalized Customer Communications Software with the calculated uncertainties of these requirements.

A general approach to uncertainty propagation is applied in this case to quantify to what extent the uncertainty in input variables (requirements definition, for instance) can affect the outcomes of software analysis and development processes (project size, for instance), and finally our software estimation true capability. Uncertainty propagation is also known as "error propagation" in science (not to be confused with defects in software engineering).

1. Introduction

When we write "software", we use "processes", indeed. We have an input, usually customer requirements or – at least – expectations, we expect output such as code, a computer system running some new applications, or services, and we have a set of controls such as time, cost, and other quality attributes. Thus writing software resembles at least in theory a process, as we know from industry. Nevertheless, applying statistic methods to software processes seems more difficult than in many other industries.

1.1. The Nature of Software Development

Software development is knowledge acquisition – it is different from industrial engineering. Industrial engineering is a transition from an imagination into reality – first is the idea, then the detailed specification, the build and testing, improving the construction if needed until it works, then operation.

Software is different. Initially, we have but a rough idea of what we want, then we must think about it in more detail, find out what resources we need, what technology to use, what the constraints are in terms of time and cost. The more we know about the problem, the better we can define it and break into parts, the closer we get to the solution.

If we want to control software development, it is necessary to address and measure knowledge, and to develop metrics for knowledge acquisition. For this, see [6].

In short, knowledge is represented by transfer functions between possibly infinite sets of requirements, which we denote as knowledge terms $\{x_1, \ldots, x_m\} \rightarrow g$ where $x_1, \ldots, x_m \in X$ and $g \in G$, where $X \rightarrow G$ represent the knowledge about the transition between different knowledge domains, for instance between technical requirements X and business requirements G. The technical solution requirements $x_1, \ldots, x_m$ are the causes hat make the business requirement g happen; this is knowledge. Such knowledge domains are denoted as ***functional topics.***

Thus, formally, requirements management can be modelled using the same techniques as for statistical process control, where we try to understand processes by a finite sample of the infinite set of all possible input and output of that process.

1.2. Statistical Process Control

In production, statistical process control is a technique based on collecting data from the process' outputs, such as physical quality attributes like size, weight, or form, and measure variations. By comparing those measured variations with various other process parameters, root causes for variations become apparent. Eliminating root causes limits these variations. We iterate that until all variations encountered remain within tolerable limits. Such limits are denoted as ***tolerance interval***. Process results that lie outside of the tolerance interval are denoted as ***defects***. A "defect" in statistical process control is therefore a "strange behaviour" of the process results, affecting the process output(s) in some way or another.

1.3. The Metrics for Variance

From statistics, the Six Sigma approach uses the following metrics for variance:

$$\sigma = \sqrt{\frac{n\sum x_i^2 - \left(\sum x_i\right)^2}{n^2}}$$

where n is the size of the data sample, and x_i are the measured data components, for $i = 1, \ldots, n$. The variance σ ***("sigma")*** corresponds approximately to $^1/_6$ of the bell's curve base length, if the data distribution is statistically normal. This is why we say, if σ is small enough to fit "six times" into the tolerance interval, our process has Six Sigma[1].

1.4. Requirements Samples

As with any statistical data set, real requirements are represented by a finite set of explicitly formulated requirements. Typically, the formulation is done in natural language; sometimes a formal requirements language is used, allowing applying some mathematical logic to it for consistency and completeness checks. The best-known such logic is Quality Function Deployment, where representative sets of requirements between the various requirement areas are linked together using linear matrices and tensor algebra. Because of our inability to deal effectively with possibly infinite sets of requirements, we are interested in knowing whether the explicit requirements sample – be it from the customer or from the business analyst – is representative enough to start spending time and money on a software project. One possibility is analysing the transfer functions between the various kind of requirements on the different levels such as customer's needs, business requirements, technical, functional, and non-functional requirements, as outlined in [6]. However, it is yet unknown whether this transfer function allows for assessing the ***uncertainty*** of an individual requirement.

Individual requirements carry an uncertainty that originates from the requirements selection process. Depending upon the language skills and domain know-how of the requirements selector, the author of the requirements specification document might easily miss, skip, or hide some details on his/her requirements, that might be fundamental for a correct software understating, and further transformation into the other levels of requirements. Therefore, if we know the knowledge transfer functions $\{x_1, \ldots, x_m\} \rightarrow g$, we can look at the uncertainty transition for such requirements.

CMMI Level 4 organizations can track technical requirements back to the originating business requirements and therefore such an approach can be quite interesting and valid.

---

[1] In manufacturing, the Six Sigma practice assumes a shift of 1.5 σ in the target mean value. For a discussion of this shift, see for instance [5].

2. Error propagation theory – A summary

When the quantity to be determined is ***derived*** from other measured quantities, and if the measured quantities are ***independent*** of each other (that is, their contributions are ***not correlated***), the uncertainty "δ" of each contribution may be considered separately [1].

For example, suppose we measured the quantities time $t \pm \delta t$ and height $h \pm \delta h$ for a falling mass (the δ's refer to the relatively small uncertainties in t and h). We have determined that:

$$h = 5.32 \pm 0.02 \text{ cm},$$

$$t = 0.103 \pm 0.001 \text{ s}.$$

From physics, we can find the gravity acceleration g from the relation:

$$g = g(h,t) = 2\ h/t^2$$

which yields $g = 10.029\ldots$ m/s^2 (the well-known value for g is actually $9.80665\ldots$ m/s^2). To find the uncertainty in the derived value of g caused by the uncertainties in h and t, we consider separately the contribution due to the uncertainty in h and the contribution due to the uncertainty in t, combined in quadrature:

$$\delta_g = \sqrt{\delta_{g_t}^2 + \delta_{g_h}^2}$$

where for instance we denote the contribution due to the uncertainty in t by the symbol δ_{g_t} (read as "delta-g-t" or "uncertainty in g due to t"). The basis of the quadrature addition is an assumption that the measured quantities have a normal, or Gaussian, distribution about their mean values and that they are not correlated one with each other.

A typical method to calculate the contributions of direct measured independent variables in the dependent variable being derived is the ***derivative method***. The basic idea is to determine by how much the derived quantity would change if any of the independent variables were changed by its uncertainty. Let the function $f = f(x_1, \ldots, x_N)$ be the formula used to derive the dependent variable f from the N independent variables $x_i, \ldots, x_N$, and let δx_i be the estimated error on the value measured for each variable x_i. That is, the true value of x_i belongs to the interval $(x_i \pm \delta x_i)$ for each i from 1 to N. Then the true value of f belongs to the interval $[f(x_1, \ldots, x_N) \pm \delta f]$, where δf is given from the derivatives formula:

$$\delta f = \sqrt{\sum_{i=1}^{N} \left(\left| \frac{\partial f}{\partial x_i} \right| \delta x_i \right)^2} = \sqrt{\left(\left| \frac{\partial f}{\partial x_1} \right| \delta x_1 \right)^2 + \ldots + \left(\left| \frac{\partial f}{\partial x_N} \right| \delta x_N \right)^2}$$

The derivative $(\partial f/\partial x_i)$ is a partial derivative (simply put, when taking a partial derivative with respect to one variable, treat any other variable as constant). Since each uncertain variable will increase, not decrease the final uncertainty, we put the uncertainty in f due to the uncertainty in x_i as the absolute value. In the cited example for acceleration $g = g(h,t)$ we have:

$$\delta_{g_t} = \left|\partial g/\partial t\right|\delta_t = \left|-4h/t^3\right|\delta_t ,$$

$$\delta_{g_h} = \left|\partial g/\partial h\right|\delta_h = \left|2/t^2\right|\delta_h ,$$

so that:

$$\delta_g = \sqrt{\delta_{g_t}^2 + \delta_{g_h}^2} = \sqrt{\left(\frac{4h}{t^3}\delta_t\right)^2 + \left(\frac{2}{t^2}\delta_h\right)^2} = 19.8 \text{ cm/s}^2 \approx 0.2 \text{ m/s}^2$$

This is the uncertainty on the previously derived measure of the acceleration gravity g = 10.029… m/s^2, due to the uncertainties on the measured quantities height h and time t for the falling mass in our experiment.

For simple formulas, as addiction, subtraction, multiplication and division of two variables, one can easily find that the uncertainty is calculated as in the following:

if $f = f(x,y) = x + y$ then $\delta_f = \sqrt{\delta_x^2 + \delta_y^2}$

if $f = f(x,y) = x - y$ then also $\delta_f = \sqrt{\delta_x^2 + \delta_y^2}$ ("uncertainties always add")

if $f = f(x,y) = x \cdot y$ then $\delta_f = \sqrt{y^2\delta_x^2 + x^2\delta_y^2}$

if $f = f(x,y) = x / y$ then $\delta_f = \sqrt{\left(\frac{x}{y^2}\right)^2 \delta_y^2 + \left(\frac{1}{y}\right)^2 \delta_x^2}$

Notice how the variables values are mixed together in the uncertainty calculation for multiplication and division - it's worth pointing out that ***fractional*** (or ***percentage*****) *uncertainties*** in multiplication and division behave much like ***absolute uncertainties*** in addition: they sum up. In other words, for $f = f(x,y) = x \cdot y$ or x / y the fractional uncertainty is:

$$\frac{\delta_f}{f} = \frac{\sqrt{y^2\delta_x^2 + x^2\delta_y^2}}{xy} = \sqrt{\left(\frac{\delta_x}{x}\right)^2 + \left(\frac{\delta_y}{y}\right)^2} .$$

If either x or y is a ***constant*** or has a relatively small fractional uncertainty, then it can be ignored and the total uncertainty is just due to the remaining term. Furthermore, if one of the measured quantities is raised to a ***power***, the fractional or percentage uncertainty due to that quantity is merely ***multiplied by that power*** before adding the result in quadrature. For instance, from the original example for $g = 2h/t^2$:

$$\frac{\delta_g}{g} = \frac{\sqrt{\left(4h\delta_t/t^3\right)^2 + \left(2\delta_h/t^2\right)^2}}{2h/t^2} = \sqrt{\left(\frac{2\delta_t}{t}\right)^2 + \left(\frac{\delta_y}{y}\right)^2}$$

For the values in our example, δ_h/h = 0.5% and δ_t/t = 1% (so 2 δ_t/t = 2%), so we can see that the contribution from the uncertainty in h is negligible compared to the contribution from t. We can therefore conclude (as confirmed by previous calculation) that the fractional uncertainty in our measured result for g is about 2%: g = 10.0 ± 0.2 m/s^2.

As a reference, Tab. 1 shows the uncertainty of simple functions, resulting from independent variables A and B, with uncertainties respectively ΔA and ΔB, a precisely–known constant **c** (that is, with an uncertainty on it which is much less than the others, or negligible) and eventually a precisely-known power value **n** (the case where the uncertainties on **c** or **n** are not negligible are more complex, but they can be equally derived by the derivative method previously discussed).

Function	Function uncertainty
$X = A \pm B$	$(\Delta X)^2 = (\Delta A)^2 + (\Delta B)^2$
$X = \mathbf{c}A$	$\Delta X = \mathbf{c}\,\Delta A$
$X = \mathbf{c}(A \times B)$, or $X = \mathbf{c}(A/B)$	$(\Delta X/X)^2 = (\Delta A/A)^2 + (\Delta B/B)^2$
$X = \mathbf{c}A^{\mathbf{n}}$	$\Delta X/X = \lvert\mathbf{n}\rvert\,(\Delta A/A)$
$X = \ln(\mathbf{c}A)$	$\Delta X = \Delta A/A$
$X = \exp(A)$	$\Delta X/X = \Delta A$

Table 1: Error propagation example formulas.

3. Measuring the Uncertainty of Requirements

The obvious way of measuring the uncertainty of requirements "a posteriori" is to measure the amount of rework induced by changes to the requirements. This is quite easy, assuming our CMMI Level 3 organization has done its measurement homework right. The problem is that this rework must be distinguishable whether caused by requirements change or by bug fixes. Bugs in this sense refers to implementations not according to specifications, or ***B-Defects*** in the sense of [5] and [6]. The notion of ***A-Defects*** refers in contrary to wrong or missing requirements. The distinction between A-Defects and B-Defects requires the ability to trace back the need for rework to the phase of its origin; again this is no problem for CMMI Level 3. However, how to measure the uncertainty of requirements "a priori"?

3.1. The Deming Process Chain for Software Development

The clue to solve this problem is that requirements are not standing alone. Requirements are among a network of different topics. A Deming process chain connects different topics – or knowledge domains – using transfer functions that are represented by linear matrices between the finite set of representative requirements for the different topics concerned. Figure 1 represents a typical Deming chain for software development, including the Voice of the Customer decision area, the Capability Maturity process enabling area, and the product realization area with product feature deployment and testing.

Let G, X be functional topics, and $g \in G$, $x_1, \ldots, x_m \in X$ requirement samples for these topics. We characterize the knowledge term $\{x_1, \ldots, x_m\} \rightarrow g$ by assigning scalar weights $\alpha_1, \ldots, \alpha_m$ that describes the strength of relationship between the cause x_i and the effect g. About knowledge terms, see [2].

Let's assume we have a set of n requirements $\{g_1, \ldots, g_n\}$ from functional topic G, and for each representative g_i some knowledge terms $\{x_{i,1}, \ldots, x_{i,m}\} \rightarrow g_i$, assuming $i = 1, \ldots, n$. Let $\alpha_{i,1}, \ldots, \alpha_{i,m}$ be the corresponding relationship weights; they can be arranged in a $n \times m$–matrix:

$$\varphi = \begin{bmatrix} \alpha_{1,1} & \ldots & \alpha_{1,m} \\ \ldots & \alpha_{i,j} & \ldots \\ \alpha_{n,1} & \ldots & \alpha_{n,m} \end{bmatrix} \qquad i = 1, \ldots, n, \text{ and } j = 1, \ldots, m.$$

We call the matrix φ of relationship weights a ***knowledge combinator***. We know such matrices as the method of Quality Function Deployment (QFD) [1]. QFD is widely used in "Design for Six Sigma", see for instance [5].

Well-established techniques exist for characterizing functional topics with only a few requirements [7], [10], [10]. By choosing a comprehensive-enough set of requirements for functional topic *G*, we can keep the number of such requirements low for that functional topic and thus describe the Deming processes by just a few characteristic requirements on both the input and the output side.

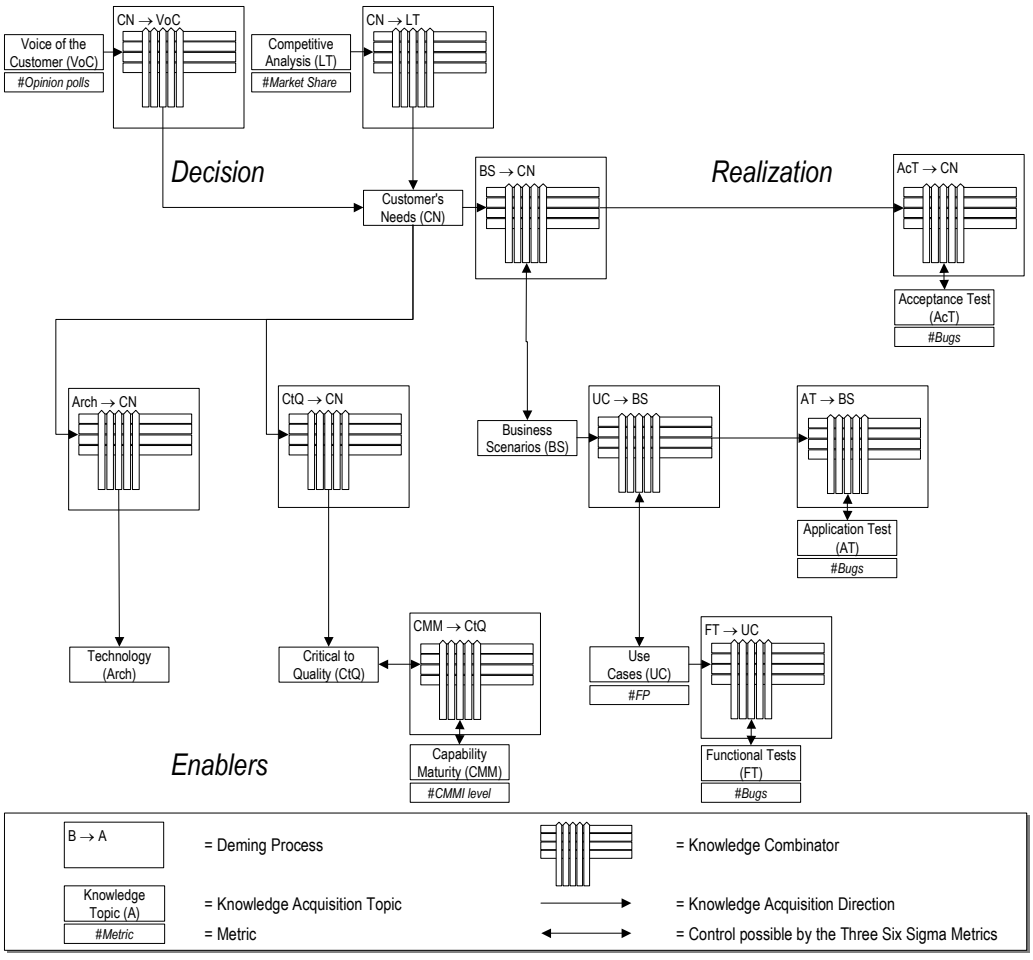

Figure 1. Deming process chain for software development with its knowledge combinators

3.2. Topic Profiles – Measuring Knowledge

With QFD, we have to possibility to measure knowledge acquisition and its variation along the Deming process chain. We do not need to count knowledge in some way. It is sufficient to study the process itself and its results, the requirements.

Let *G* be a functional topic as before. The topic *G* may represent the Customer's Needs that result from the decision processes in the Deming process chain. The requirements of *G* are not equally graded; there exist different weights. The set of these different weights is denoted as a ***(requirement) profile.*** If the weights for the effects are not equal, the solution

requirements that cause these effects cannot be either. Thus for the solution requirements X in the knowledge $X \rightarrow G$ there must also exist profiles that give suitable weights to the requirements of X. In practice, this means that the resulting effect on G depends both from the selection of requirements in X, and from their respective weights.

Usually you have a goal profile for the requirements $g_1, \ldots g_n \in G$; this is the vector $\langle g \rangle = \langle \gamma_1, \ldots \gamma_n \rangle$. This vector represents the profile of the desired effects. Given any solution requirements $x_1, \ldots x_m \in X$, we would like to know its solution profile that produces an effect as close as possible to the goal requirements (see Fig. 2). Assume the profile for the cause vector is $\langle x \rangle = \langle \xi_1, \ldots, \xi_m \rangle$, the ξ_j representing scalar values for the weights of the solution requirements $x_j, j = 1, \ldots, m$.

The matrix (2) allows for the calculation of the resulting profile when applying knowledge $X \rightarrow G$ to the functional topic X. Then we calculate the weights of the resulting effects as follows:

$$\varphi(\langle x \rangle) = \langle \varphi_1, \ldots, \varphi_n \rangle = \left\langle \sum_{j=1}^{m} \alpha_{1,j} * \zeta_j, \ldots, \sum_{j=1}^{m} \alpha_{n,j} * \zeta_j \right\rangle$$

component wise: $\varphi_i = \sum_{j=1}^{m} \alpha_{i,j}$, $i = 1, \ldots, n$

Again, the $\alpha_{i,j}$ denote the QFD matrix elements, which represent the weights of the relationship between solution requirements and effects ($i = 1, \ldots, n$; $j = 1, \ldots, m$); φ is a mapping from a vector with m components into another vector with n components. We call the resulting profile $\varphi(\langle x \rangle) = \langle \varphi_1, \ldots, \varphi_n \rangle$ the ***effective profile***. Unfortunately, the effective profile will not automatically match the original goal profile $\langle g \rangle$. With $\langle x \rangle$ selected arbitrarily, there is a gap between the vectors $\langle g \rangle$ and $\varphi(\langle x \rangle)$. The only commonality between the two vectors is that they both have the same number of components, namely n.

The knowledge combinators provide the metrics that define the statistical variance that we must expect when developing software. The Deming process chain yields metrics that let us control the deployment of requirements throughout the software development. see [6] for more on this.

3.3. Measuring Uncertainty

With this measurement framework for requirements, we can predict uncertainty at the beginning of the software development process by taking its derivative per component. Thus the uncertainty of a business requirement depends from the effect that a variation in the technical solution area produces.

Let $\varphi(\langle x \rangle) = \langle \varphi_1(\xi_1, \ldots, \xi_m), \ldots, \varphi_n(\xi_1, \ldots, \xi_m) \rangle = \langle \varphi_1, \ldots, \varphi_n \rangle$, where $\langle x \rangle = \langle \xi_1 \ldots, \xi_m \rangle$ and m, n some dimensional numbers. Then the formula:

$$\delta\varphi_i = \sqrt{\sum_{j=1}^{m} \left(\left| \frac{\partial \varphi}{\partial x_j} \right| \delta x_j \right)^2} = \sqrt{\left(\left| \frac{\partial \varphi}{\partial x_1} \right| \delta x_1 \right)^2 + \ldots + \left(\left| \frac{\partial \varphi}{\partial x_m} \right| \delta x_m \right)^2} \qquad i = 1, \ldots, n$$

expresses the uncertainty of the goal requirement φ_i, due to the uncertainties on the solution requirements $\langle x \rangle$.

Thus we can predict the uncertainty of a single requirement from nothing else than the requirements for the technical solution, and later compare it with the actually measured uncertainty.

4. Validation

From Figure 1, we'll use the knowledge chain FT → UC, UC → BS, and BS → CN, thus from Functional Tests (FT) via Use Cases (UC) and Business Scenarios (SC) back to Customer's Needs (CN), wishing to know the uncertainty of the requirements for Use Cases, Business Scenarios, and Customer's Needs.

Every knowledge combinator creates an uncertainty of its own, resulting from the variations induced by the solution approach. Uncertainties add up, therefore the uncertainty of the Customer's Needs (CN) is composed from the uncertainties induced from all of the selected solution approaches for the Business Scenarios (BS).

The actual measurements being considered in this work are taken from the Measurement Repository of the Software House GMC Software Technology[2]. As measurement data, we have effort and duration of actual project tasks that we can compare with the estimations on the level of Use Cases (UC). There is no "natural" metric for uncertainty in this data source, thus we used the following metrics as the actual uncertainty per project task in the dataset:

$$\text{uncertainty} = \sqrt{\frac{|p^2 - a^2|}{a^2}}$$

where p is the planned duration and a the actual duration of a project task. Since duration and effort are closely linked within GMC's software operations, we can assume that they provide the "same" metrics.

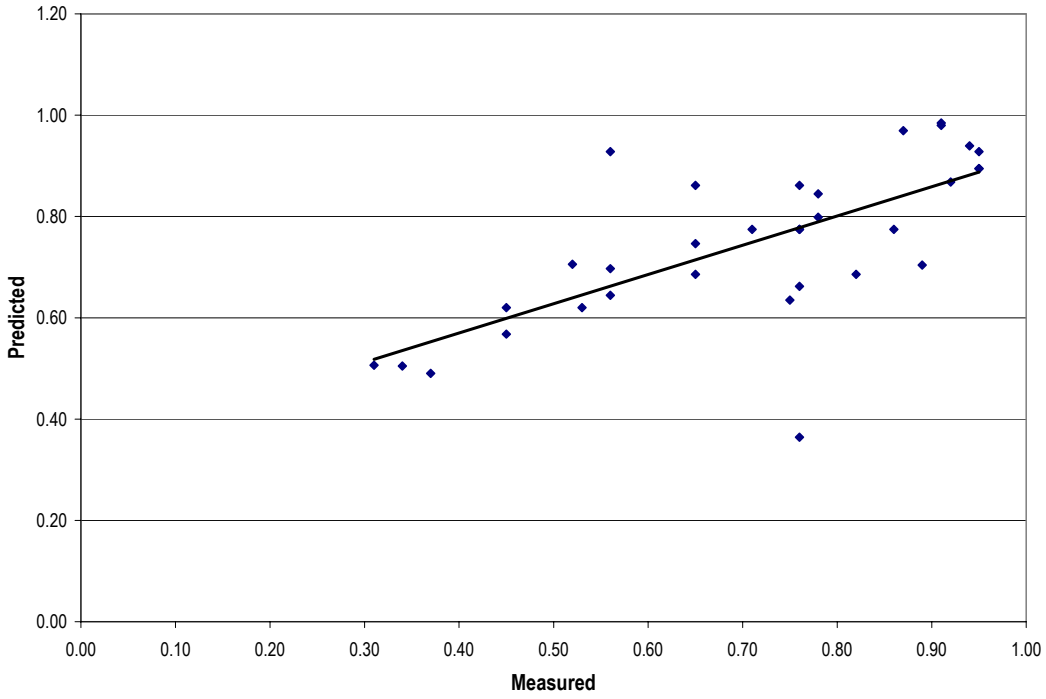

Figure 2. Predicted vs. Measured uncertainties (data source: GMC Software Group - Customer communications design software – r. 5).

[2] GMC Software Group is a Level 4 company that provides standard solutions for customer communications. The software development is located in Czech Republic; sales and support office are in Europe, Asia, and the Americas; the headquarters are in Switzerland. See www.gmc.net.

Since the derivative calculated from the knowledge combinators has no dimensions as well, we can scale both, the predicted uncertainty metrics as well as the measured uncertainty to the same range, for comparison. In the graph from Figure 2, we scaled them to a unit scale, which is easily understandable by management. If estimations and actual durations differ by a factor of 2, the uncertainty metrics would yield 0.77. For a perfect match, uncertainty would be equal to 0. If they are completely off, such as for seven days in estimation and one day in actuals, the uncertainty grows near to 1.0.

The correlation coefficient of the linear fit in Figure 2 is 0.70; such correlation is interesting enough to justify further studies on the usage of uncertainty estimation. Note that in this sample, we did not remove clear outliers such as these use cases that use only one day instead of seven.

Uncertainty metrics for use cases (UC) can be uplifted to business scenarios (BS) and Customer's Needs (CN), yielding uncertainty metrics for the respective requirements. No data exists that we can use to compare with; thus our uncertainty metrics reflect the uncertainty that we induce in the software process by selecting the validation strategy on the use case level. The metrics can be used to understand whether the chosen test strategy reflects the project's goal statement, reflected in customer's needs, or if our development strategy has induced certain risks that otherwise may have gone undetected.

5. Conclusions

The uncertainty metrics has now been incorporated into the measurement dashboard of GMC software development. For Six Sigma or CMMI Level 4 companies, the Uncertainty Estimation based on the uncertainty propagation theory, adds quite some value. First of all, it supports selecting test cases such that uncertainty on technical requirements or Use Cases is minimized. Furthermore, if software development is based on the Deming chain of requirements, it allows to get information out of knowledge combinators that's applicable for individual requirements, on all levels, including Customer's Needs. Risks with specific requirements can more easily be detected based on metrics than on experience and brainstorming only. It does not replace experience and domain expertise, it can effectively support it. When selecting new features to implement in software, uncertainty of requirements on all levels is a very valuable asset.

Companies neither doing QFD nor Six Sigma may find it difficult to set up knowledge combinators for development control. Those that do it get yet another advantage that is difficult to match for less advanced competitors.

Six Sigma and QFD are methods in the public domain, not protected by proprietary licences, methods or tools. They are not easy to master but results are overwhelmingly convincing. GMC Software Group that uses this approach is experiencing growth rates in the 50% range for many years now. Other companies can achieve this too, when they start to understand high-quality software business.

6. References

[1] Akao, Y. et. al. (1990), Quality Function Deployment (QFD); Productivity Press, University Park, IL.

[2] Engeler E. (1995), The Combinatory Programme, Birkhäuser, Basel, Schweiz.

[3] Fenton, N.; Krause, P.; Neil, M. (1999), "A Probabilistic Model for Software Defect Prediction", IEEE Transactions on Software Engineering, New York, NY.

[4] Fehlmann, Th. (2004), "The Impact of Linear Algebra on QFD", in: International Journal of Quality & Reliability Management, Vol. 21 No. 9, Emerald, Bradford, UK.

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

[6] Fehlmann, Th. (2006), "Six Sigma Revisited", EuroSPI^2 Conference Proceedings, 2006.

[7] Herzwurm G., Mellis, W., Schockert, S. (1997): Qualitätssoftware durch Kundenorientierung. Die Methode Quality Function Deployment (QFD). Grundlagen, Praxisleitfaden, SAP R/3 Fallbeispiel.

[8] Jensen, RW, Putnam, LH, Roetzheim, W, Software Estimating Models: Three View-points, in Crosstalk, The Journal of Defense Software Engineering, Feb. 2006. Web: http://www.stsc.hill.af.mil/crosstalk/2006/02/0602JensenPutnamRoetzheim.html.

[9] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, Ch. V. (1993), Capability Maturity Model for Software, Version 1.1, Software Engineering Institute, CMU/SEI-93-TR-24, DTIC Number ADA263403, Carnegie-Mellon University, Pittsburg, PA.

[10] Saaty, Th. (1999), „Fundamentals of the Analytic Network Process", in: ISAHP 1999, Kobe, Japan.

[11] Santillo, L., "Error Propagation in Software Measurement and Estimation", IWSM/MetriKon 2006 conference proceedings, Shaker Verlag, Aachen, Germany, 2006.

[12] Taylor, J.R., An Introduction to Error Analysis, The Study of Uncertainties in Physical Measurements, University Science Books, 1982.