

International Conference on Software Process and Product Measurement

**November 4 - 6, 2009
Amsterdam, Netherlands**

INDUSTRIAL TRACK PAPERS

Editors

**Alain Abran
René Braungarten
Reiner R. Dumke
Juan J. Cuadrado-Gallego
Jacob Brunekreef**

Editors

Alain Abran

Département de génie logiciel et des TI
École de technologie supérieure
1100, rue Notre-Dame Ouest
Montréal, Québec
Canada H3C 1K3
Alain.Abran@etsmtl.ca

René Braungarten

Bosch Rexroth Electric Drives and Controls GmbH
Quality Management and Methods, Processes DCC/QMP
Bürgermeister-Dr.-Nebel Str. 2
97816 Lohr am Main
Germany
rene.braungarten@boschrexroth.de

Reiner R. Dumke

Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Verteilte Systeme, AG Softwaretechnik
Universitätsplatz 2
39106 Magdeburg
Germany
dumke@ivs.cs.uni-magdeburg.de

Juan J. Cuadrado-Gallego

Universidad de Alcalá
Edificio Politécnico, O24. Autovía A2, Km. 31,7
28805 - Alcalá de Henares, Madrid
Spain
jjcg@uah.es

Jacob Brunekreef

University of Applied Sciences HvA
Media, Creation and Information
Weesperzijde 190
1097 DZ Amsterdam
The Netherlands
j.j.brunekreef@hva.nl

Preface

Since 1990 the International Workshop on Software Measurement (*IWSM*) has been held annually and is now in its 19th edition. The International Conference on Software Process and Product Measurement (*Mensura*) was initiated in 2006 and is now in its 3rd edition. The editions of *IWSM/Mensura* have been combined since 2007 to foster research, practice and exchange of experiences and best practices in software processes and products measurement. The 2009 editions were held during November 4-6, 2009 in Amsterdam, organized jointly with the Netherlands Association for Software Measurement (NESMA) and kindly hosted by the Hogeschool van Amsterdam.

The proceedings of *IWSM/Mensura* 2009 have been published in two separate volumes. The scientific papers have been published in the Springer *Lecture Notes in Computer Science* (LNCS) series. The industrial track papers are published in this booklet. The proceedings are testimonies of many of the software measurement concepts developed and of their related use in industry. These proceedings are of particular interest to software engineering researchers, as well as to practitioners, in the areas of project management and quality improvement programs, for both software development and software maintenance.

November 2009

Alain Abran
René Braungarten
Reiner R. Dumke
Juan J. Cuadrado-Gallego
Jacob Brunekreef

Table of contents

Practical experimentations with the COSMIC method in Automotive embedded software field <i>Sophie Stern</i>	1
Reliability of software metrics tools <i>Dennis Breuker, Jacob Brunekreef, Jan Derriks, Ahmed Nait Aicha</i>	10
Quality of process control in software projects <i>Yegor Bugayenko</i>	23
A Proposal for an Integrated Measurement and Feedback Environment “MIERUKA” for Software Projects Based on the Empirical Study of Three Measurement Domains <i>Yoshiki Mitani, Hiroshi Ohtaka, Noboru Higuchi, Ken-ichi Matsumoto</i>	29
Estimating the functional size of applications built with the Oracle eBS Package <i>Frank Vogelezang, Bert Veenstra, Cor van Dongen, Johan de Vries, Joyce Span, Hessel Bijlsma</i>	33
Implementing a Metrics Program MOUSE will help you <i>Ton Dekkers</i>	46

Practical experimentations with the COSMIC method in Automotive embedded software field

Sophie Stern

RENAULT, Embedded software group
TCR RUC T 65, 1 avenue du Golf, 78 288 Guyancourt cedex, France
sophie.stern@renault.com

Abstract

More and more functionalities are available in cars (to increase security, to reduce the CO2 emissions, to improve the connectivity with the outside world and so on) and the most part of these new features is realized by software. The automotive industry is used to manage very accurately the physical parts costs with its suppliers and has now to face to the software parts development costs management too. For that, it is necessary to construct effort estimation models based on a standard functional size measurement method which must be explicable, factual, simple and reproducible. The major goal of this paper is to give a feedback on the Renault practical experimentations with the COSMIC method on the embedded software field.

Keywords

The COSMIC functional size measurement method, Effort estimation models, Software development costs estimations, Software productivity management.

1 Introduction

During the last ten years, as the functionalities available for customers are more and more numerous, the cars' complexity has evolved increasingly. This complexity is mainly supported by calculators (Airbag, ABS,...) that we call ECU for Electronic Control Units. Historically, Renault subcontracts the development and the manufacturing of its ECU to many suppliers. An ECU presents the particularity to be part of hardware and part of embedded software. For several years, Renault has been used to pay each ECU as a whole. But with the increase of the software complexity, and on the same time the decrease of the electronic parts costs, the software development cost is less and less marginal and may be higher than the hardware one for major ECU.

If Renault, as the other cars manufacturers, is very used to estimate the development cost of physical parts such as harness or electronic components, it is quite lacking in for software development cost estimation.

Several departments but especially the Renault purchasing one asked to the Renault embedded software group to find a method to estimate the embedded software development cost.

The main goal of this short industry paper is to show how the COSMIC method has been used in the Renault experimentation and the learnt lessons.

After the presentation of the reasons why Renault chose to evaluate the COSMIC method for embedded software development effort management, I will present the goals assigned to the experimentation, first results, best practices and I will conclude on what next.

2 Why had Renault chosen to evaluate the COSMIC method for embedded software development effort management?

In 2008, the Renault purchasing department requested from the embedded software group a method to estimate the embedded software development cost in order to manage it more accurately.

The first point was to find a standard method to estimate the development effort which could be converted later in development cost with applying the suppliers' rates.

As everybody knows, it is not possible to measure a piece of software with a meter or a balance, and it appears that it is not possible anymore to measure one with the number of written lines code because it would have no sense with automatic coding. So we decided to interest ourselves to the Functional Size Measurement (FSM) methods.

The Renault embedded software group started by a state of the art of the FSM methods, two were studied particularly at first because of their past in Renault: the IFPUG and the COCOMO methods.

The COCOMO method had been experimented a few years ago in the ECU diagnostic department, nevertheless with unsuccessful results.

The IFPUG method has been used for several years in the Renault Information System department, and no new information system can be launched without its cost has been first evaluated by the IFPUG software effort estimation cell.

Nevertheless, the possible application of IFPUG on embedded software had to be checked with an evaluation.

In order to benchmark at least two FSM methods, we chose the COSMIC method as it was announced to be well adapted to real-time software as the embedded software in ECU is.

Our first experiments started on the Engine Control Unit at mid 2008 with the IFPUG and COSMIC methods. The Engine Control Unit is modular and each of its modules is a set of specifications under the Matlab/Simulink tool with also textual requirements. The effort supplier invoice is available for each module.

Functional size measurements were realized on the same nine modules with the two studied methods, and then results were compared.

In the Renault experimentation, the IFPUG functional sizes measurements were always higher than the COSMIC ones, the COSMIC method suited well for embedded software whereas the IFPUG method appeared not pertinent especially when the functional software size was increasing. Furthermore, the measurements with the COSMIC method seemed to us easier than the ones with the IFPUG method.

So we decided to pursue the experimentation for embedded software with the COSMIC method in a project way.

3 The Renault COSMIC project

In 2009, we decided to pursue in a project way and to take into account another ECU, the Body Control Module (BCM). The BCM is specified by Renault with the Statemate tool and textual requirements. The BCM is specified in software functions, the supplier effort invoice is available for each of them.

The first step was to measure functional sizes on BCM functions and on Engine Control Unit modules and to try to find linear regressions with the supplier effort invoices. For the Engine Control Module, it was on software evolutions, for the BCM, it was on software evolutions and on new developments.

In each case we found one or several linear regressions. At this moment, as the Renault implicated actors and managers were numerous, from different departments and with different interests in the COSMIC method, it appeared to me that it was necessary to define very clearly the goals of the COSMIC method application evaluation on Renault ECU.

We defined with purchasing departments, departments in charge of specifications, ECU project management, four major goals.

3.1 The four major goals of the COSMIC method application evaluation on Renault ECU

- Have indicators to manage the supplier productivity year after year.
- Predict the cost of the software functions development in order to negotiate with suppliers the right price.*
- Be able to estimate a function software development cost as soon as its specification is written to decide to implement or not the function.
- Benchmark the productivity of the different suppliers.

* This objective is a first milestone. The final goal is to be able to contract the price of a functional size in CFP (COSMIC Function Points) with suppliers. Notice that this process is already applied since several years for information systems.

These major goals are then declined in smaller goals for each of them.

I put in place steering committees for each ECU development sector to check regularly the goals and of course to check the advancement on work packages and usual indicators on projects: problems resolution, risks and planning.

3.2 A cell of measurers, COSMIC measurement textual guides

As the software functional size measurements are performed by several measurers, it is mandatory to define very clearly and without any ambiguity, the way of measuring functional sizes. We wrote COSMIC measurements textual guides. Several functional size measurements were performed independently by different persons, these experiments showed that even manual measurements are very reproducible with at the most 1% of difference.

3.3 The way of constructing COSMIC models

We made the choice to have one different COSMIC model for each ECU. We also constructed different models for each supplier. This is interesting to have standards of comparison and to benchmark suppliers.

Until know, we made the choice to split models when influent factors are different, but I intend to pursue our statistics studies with multi-factors approaches.

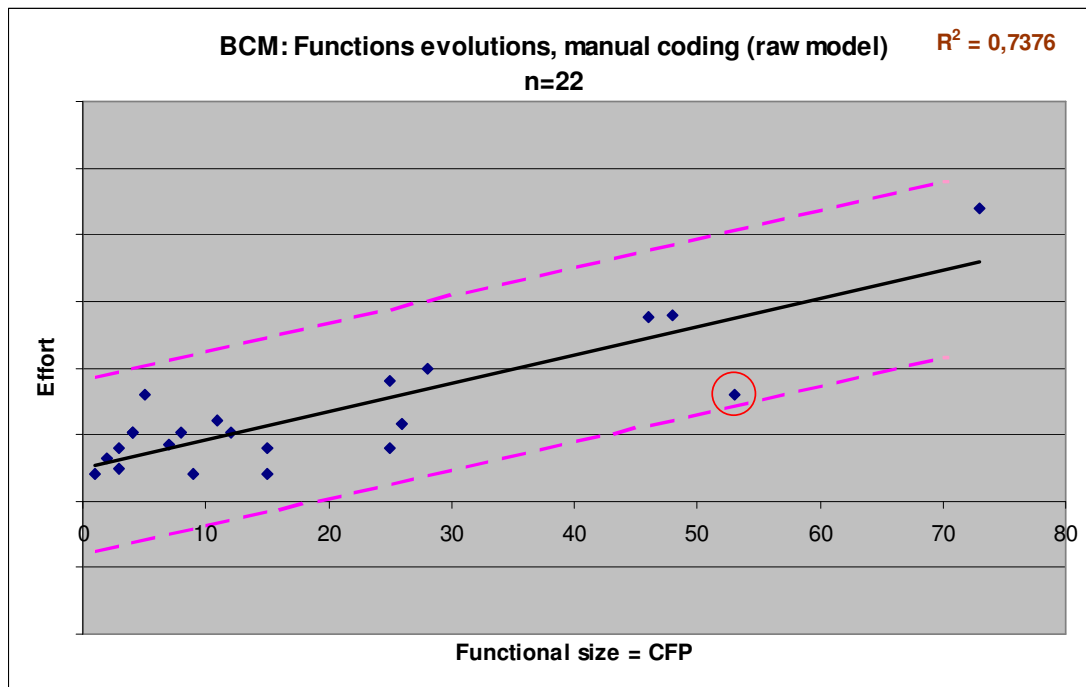
3.4 Obtained COSMIC results

We followed the COSMIC method and we constructed regression models with only one variable: the functional size in CFP.

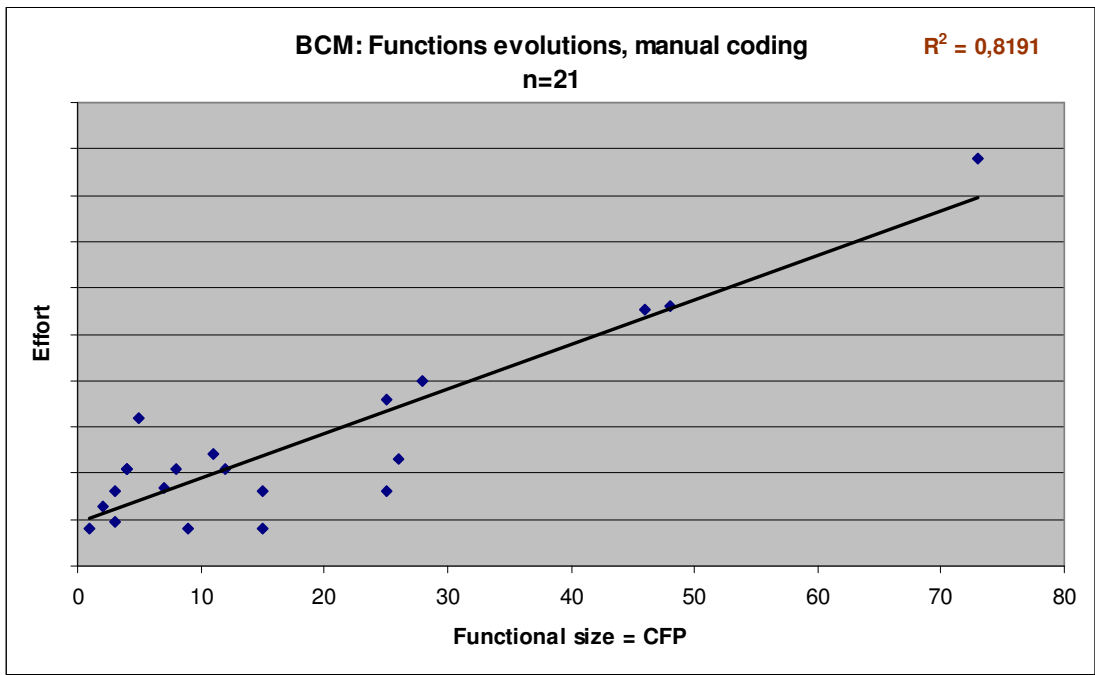
After our experimentations, my opinion is that it is necessary to find the balance between too many models and models easily understandable by many actors as the ECU development project, the purchasing, suppliers. The split of one COSMIC model in several models must always have a meaning.

BCM COSMIC models

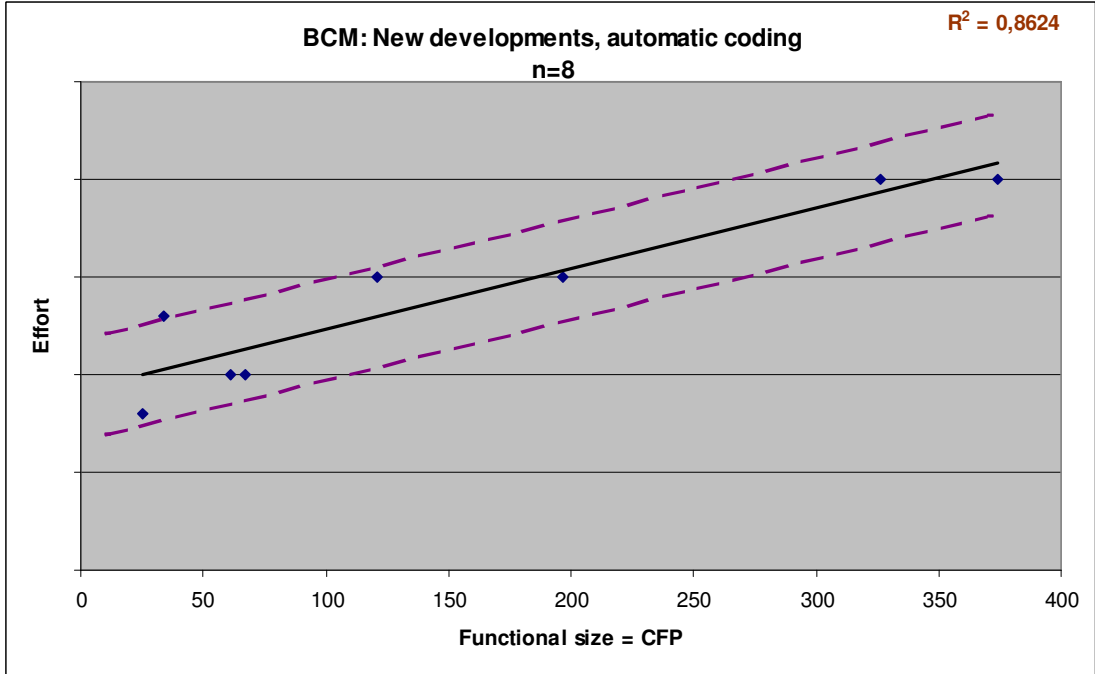
The BCM COSMIC model was first split between: new developments and functions evolutions. Then these two BCM models were split again between: manual developments and automatic ones. Four reference COSMIC models are defined at the moment for the Renault BCM.

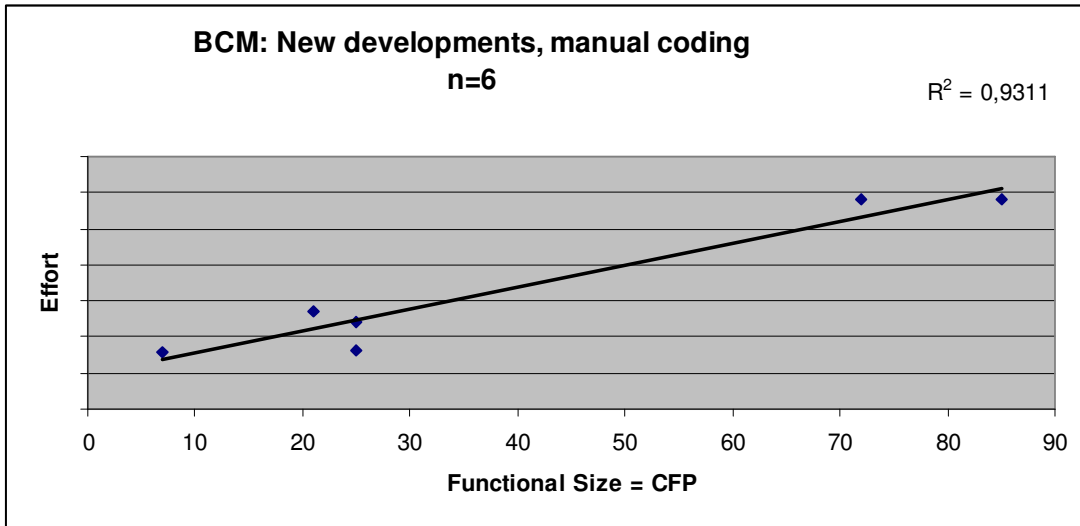


On this raw model, there is one point very close to the confidence interval limit. This BCM function was developed by an expert in the supplier team so we decided to remove this point from the regression curve and to capitalize in a checklist of particular functions. After this correction, we have the following reference model.



The number of points for the two BCM COSMIC models for new developments were low but the coefficients of determination were correct and these two models have been very helpful for us to realize predictions during a request for quotation with suppliers for a new BCM development.

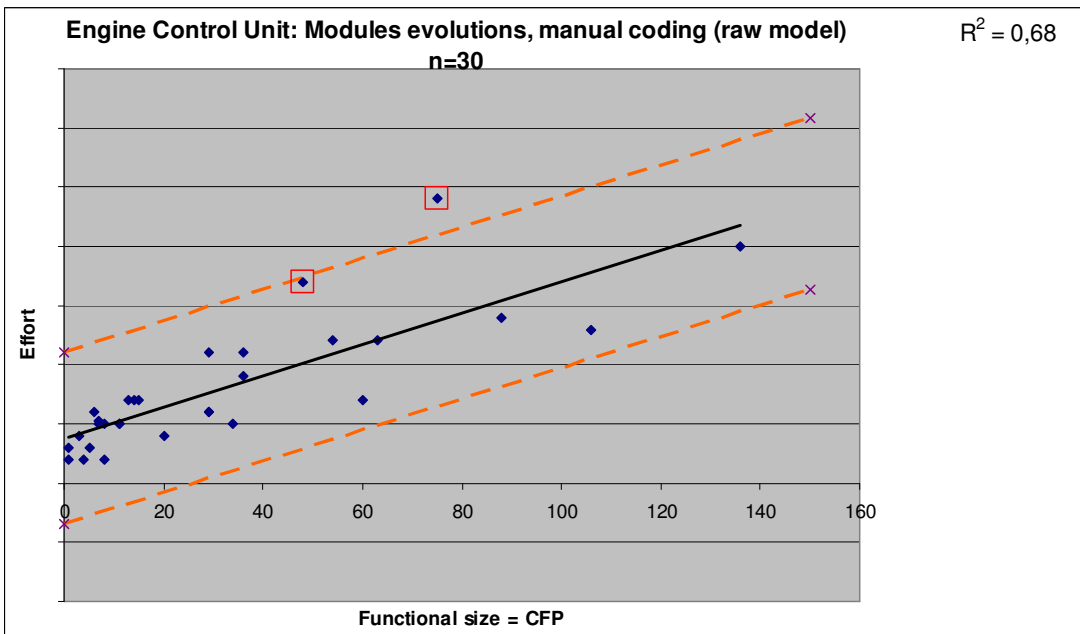




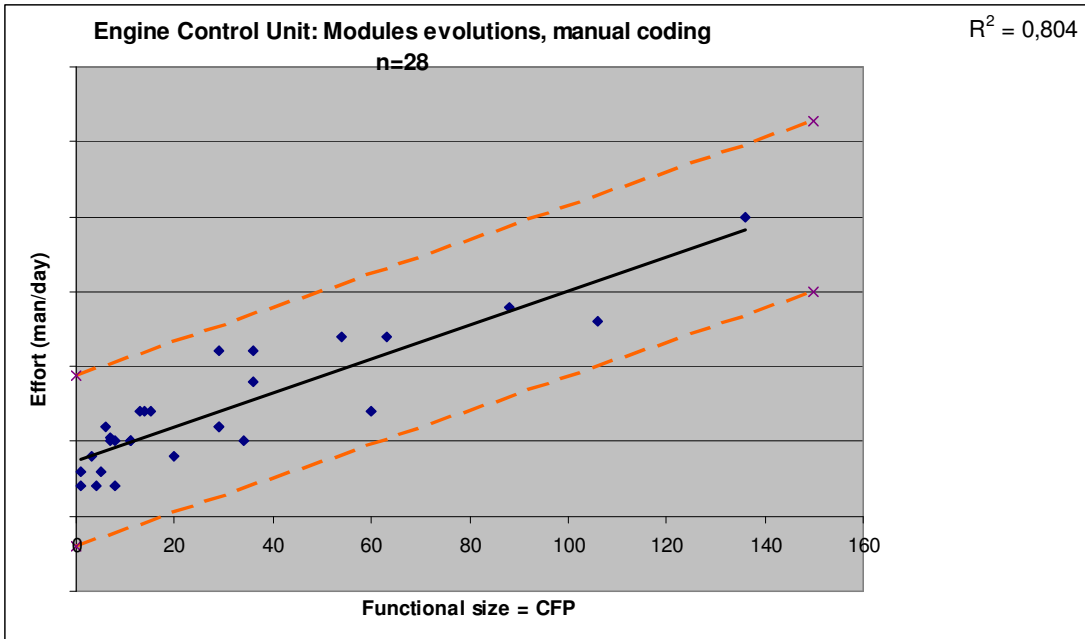
An important point will be to complete the reference COSMIC models with new data when available.

Engine Control Unit COSMIC models

For the Engine Control Unit, the COSMIC model was only split between the different suppliers, all considered developments are modules evolutions. One COSMIC model for one supplier is shown below.



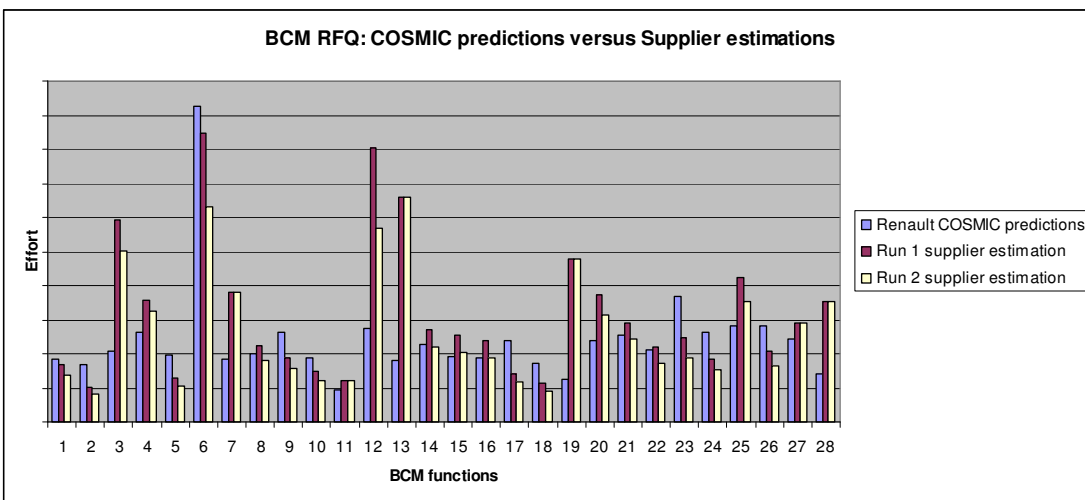
The two points in squares correspond to modules very different by their nature to the other ones. They have been removed from the COSMIC model and capitalized in a checklist until we can construct a COSMIC model for this kind of modules. After this correction, we have the following reference model.



A real experimentation on BCM: COSMIC predictions in a RFQ phase

We applied our BCM COSMIC new developments models during a Request For Quotation with suppliers for a new BCM in order to have a target software development cost on the applicative software. As soon as the specifications were written, we predicted the development effort for each BCM function within a prediction interval, we also predicted the development effort and the associated prediction interval for the whole applicative software. Then we negotiated with the supplier its estimations.

This is the comparison between our COSMIC predictions and the supplier estimations after the first round and after the second round with the supplier.



The negotiation is not finished but this example shows that factual measures realized with a standard method is a good lever to negotiate.

Predictions for the Engine Control Module

For the Engine Control Module, as there are regular software development invoices for modules evolutions, we did predictions on two modules. Each supplier invoice was in our prediction interval.

We have to pursue such experiments to make everybody, inside and outside the company, confident in the predictability of the COSMIC method.

3.5 The success keys for applying the COSMIC models

Capitalization

As we are not fortune-tellers, it is not possible to predict software development effort or cost without capitalization on ECU projects from the past. The storage must be precise and complete, in our case we need to store software specifications and the development cost for each specification. For some ECU, the first step will be to change the process of costs negotiation with the supplier to obtain the detailed software costs.

The implication of the ECU purchasers

The goals of the ECU purchasers are in line with the COSMIC method possibilities. It is important to explain them the method and to show some successful experiments in order to have their support because they may be a very good lever of change inside the company and outside with suppliers.

The help of the ECU project team

The COSMIC measurers are often embarrassed to explain the points outside the COSMIC regression curves, the help of the ECU project team is mandatory to interpret the modules intrinsic particularities or the ECU development process specificities.

Furthermore, the implication of the ECU project team is very important to deploy the method with the supplier.

Meeting points with the supplier

The final goal of using the COSMIC method is often to share it with suppliers and to contract agreements on the basis of factual software functional measures.

It is important to explain step by step the COSMIC method to suppliers, to make experiments with them to show the accuracy of the method.

Multidisciplinary team

The COSMIC core team needs people with knowledge and capabilities in computer science and in statistics. As it is very difficult to find people with these two competencies, the best way is first to mix people with different profiles in the same team and to trainee them.

Process change

To improve the way of buying software to suppliers, it is necessary to conduct a process change in the organization with a roadmap.

4 And now, what next?

As the Renault experimentations with COSMIC are very encouraging for embedded software cost management, we intend to continue with the major following actions:

- Pursue COSMIC experimentations on other ECU.
- Experiment statistical multi-factors models.
- Find one method to work upstream when simulation models are not available.
- Estimate other software development efforts than applicative: basic software development effort, ECU functional validation effort,...
- Automate the whole measurement process. At last but not least.

Reliability of software metrics tools

Dennis Breuker, Jacob Brunekreef, Jan Derriks, Ahmed Nait Aicha
{d.m.breuker, j.j.brunekreef, j.derriks, a.nait.aicha}@hva.nl

School of Design and Communication
Amsterdam University of Applied Sciences
(Hogeschool van Amsterdam)

The second author is also employed by DNV (www.dnv.com)

Abstract

Software metrics tools play a central role in acquiring information about the quality of software systems. In this paper it is shown that these tools are not reliable: too often different tools give different measured values for the same metric. In this paper the causes for the differences are investigated. For a restricted set of metrics the literature definitions are compared with the definition provided by the tool constructors and the values that have been measured. A small student project was used for measurement. This made it possible to compare measured values with values based on metrics definitions (from the literature or from a tool) and visual inspection of the source code.

Keywords

Software metrics, metrics tools, reliable tools.

1 Introduction

More and more, management decisions in IT-projects are (at least partially) based on measurement results extracted from source code. Therefore, from an industrial point of view it is remarkable that the reliability of software metrics tools does not seem to be a topic that draws much interest. At the Computer Science department of the Amsterdam University of Applied Sciences a big software quality measurement program for student projects is under construction. In the context of the development of this program, a set of software metrics tools has been inspected and tested for its usability. During these tests, a variation in measured values for the same metric on the same project appeared to be anything but an exception. The reliability of the metrics tools appeared to be questionable. Several causes can be identified for the observed lack of reliability. First, the definition of a metric in the literature may be imprecise, so a tool supplier will have to make some guesses while implementing the metric. Second, a variation in definitions exists and different tool suppliers choose different definitions. Finally, the implementation of a certain metric in a tool can be wrong.

This paper reports about a research project in which a small student project (about 4000 lines of Java code, consisting of 24 files) has been fed to a selection of six metrics tools. The restricted size of the student project made it possible to apply visual code inspection in order to validate some of the measured values.

The following questions were leading:

- Are the applied metrics well-defined in the literature and the tool documentation/help files?
- How big are the differences in values, measured with different tools for the same metric?
- Can an explanation be found for the observed differences?

We are aware of one study similar to ours, by Lincke *et al.* ([5]). They have measured the outcome of a number of metrics tools on several projects. They have found remarkable variations in the measured values. They also have shown that these variations influence the high level quality judgments that are based on the measured values. In their conclusions section they state that “an in-depth study should seek to explain the differences in measurement results.” Our paper can be regarded as at least a partial answer to this suggestion as it is presupposed that there will be differences in measured values, and from that starting point it is investigated where these differences come from.

Section 2 presents the selection of the software metrics and the software metrics tools that have been used in this research project. Section 3 deals with the definition of the selected metrics, both in the literature and in the documentation and help files that come with the metrics tools. Section 4 shows and analyses the values measured with the various tools for the various metrics. Conclusions (section 5) and references complete the paper.

2 Selection of metrics and metrics tools

Today a wide variety of metrics are in place to guide the development and deployment of software systems. Metrics provide information about dynamic behavior and static properties of a system. The experiments described in this paper focus on three subsets of metrics for static code properties: basic size metrics, class-oriented structure metrics and complexity metrics.

The first subset contained the following size metrics:

- NFL: Number of files in a project
- NCL: Number of classes
- NMT: Number of methods
- LOC: Number of lines of code. This metric comes in different flavors:
 - pLOC: physical lines of code
 - eLOC: effective lines of code
 - sLOC: statement lines of code
 - ILOC: logical lines of code
- CML: Number of comment lines

For the second subset with structure metrics, the well-known Chidamber-Kemerer metrics set for OO-code has been chosen. This set contains six metrics:

- WMC: Weighted methods per class
- DIT: Depth of inheritance tree
- NOC: Number of children
- CBO: Coupling between objects
- RFC: Response for a class
- LCOM: Lack of cohesion in methods

Finally, one of the oldest and most well-known metrics with respect to the static quality of source code has also been incorporated in the investigations: the McCabe Cyclomatic Complexity, referred to as MCC in this paper.

The output of a set of six metrics tools that were pre-selected for the measurement program has been inspected. As the small student project was a Java project, the selection was restricted to metrics tools able of processing Java code. Three tools (SourceMonitor, ckjm and RefactorIT) are free tools, one tool (RSM) is a low-priced commercial tool, and two tools (CMTJava and Essential Metrics) are mid-priced commercial tools. Evaluation copies of the last two tools have been used. Although RefactorIT is a refactoring tool and not primarily a metrics tool, it delivers an extensive set of software metrics. This tool has been incorporated in the experiments to see if it can compete with dedicated software metrics tools.

Not every tool is capable of producing all the metrics listed above. Therefore subsets of the tools have been used for the various metrics sets. ckjm operates on .jar files and hence does not provide line count metrics. SourceMonitor and RSM do not provide the Chidamber-Kemerer metrics set. ckjm does not produce complexity metrics.

The following versions of the tools have been used: ckjm v1.8, CMTJava v2.2, Essential Metrics v1.00.002, RefactorIT v2.7 beta, RSM v7.70 and SourceMonitor v2.5.0.2. More information about the tools can be found on the websites of the tool providers ([7]).

3 Definition of metrics

The widespread use of various metrics in many discussions about software quality suggests that everybody knows what we are talking about when referring to “the number of lines of code in a project” or “the McCabe cyclomatic complexity of a certain method”. However, a small inspection of the literature concerning software metrics already shows that this assumption is too optimistic. Even a metric that is very simple at first sight, like “Lines of code”, appears to be imprecise without further definition. Do we mean physical lines of code, executable lines of code, logical lines of code? And, how do we define these metrics?

This paragraph first discusses the definitions found in the literature, followed by the definitions given by the tools.

3.1 Metrics definitions in literature

Many textbooks and papers about software metrics provide metrics definitions. The question is to what extent these definitions are in line with each other: when referring to the same metric, is the same definition given?

3.1.1 Size metrics

Guidelines for counting the number of files in a project (NFL) seem to be absent in the literature. This is no surprise, as counting the number of files is not difficult as far as counting files is involved. It is a little bit more complicated when it comes to the decision what files have to be included in the count: only source code files, or also project files, include files, etc.

Guidelines for counting the number of classes and methods in a project (NCL, NMT) are also not found in the literature. This means that a question like “Is an inner class counted when counting the number of classes in a project?” is not answered.

In the literature much attention has been paid to counting the number of lines of source code (LOC). Many different line counts have been proposed in the past: physical lines of code, executable lines of code, statement lines of code, etc.

The definition of a physical line of code (pLOC) seems to be uniform: a string of characters, terminated by the Enter-character (see e.g., [3]). The same holds for the definition of a comment line: every line that holds a start-comment token (e.g., //), or is part of a comment block (e.g., the character string between the tokens /* and */), is counted as a comment line. Note that a line containing a statement followed by a comment is counted as a comment line. However, concepts like effective lines of code (eLOC), logical lines of code (lLOC) and statement counts (sLOC) are introduced without a precise definition. This gives way to multiple interpretations and hence different implementations.

3.1.2 Structure metrics

The Chidamber-Kemerer metrics suite contains six metrics. They are introduced and defined in their original paper from 1994 ([1]). For some of these metrics the definition is straightforward. E.g., the depth of inheritance (DIT) is defined as the (maximum) length from the node to the root of the inheritance tree. The coupling between object classes (CBO) is defined as a count of the number of classes to which a class is coupled, by method use or instance variables.

Other CK-metrics are defined in a more mathematical way. E.g., the lack of cohesion in methods (LCOM) is defined as $LCOM = |P| - |Q|$ if $|P| > |Q|$ and $LCOM = 0$ otherwise, where P is the set of non-intersecting methods and Q is the set of intersecting methods, based on the common use of variables of all methods in a given class. Other sources give merely textual definitions. Both Fenton ([2]) and Kahn ([4]) define LCOM as “the number of disjoint (that is: non-intersecting) sets of local methods.” This complies with the definition “ $LCOM = |P|$ ”, which is different from the definition given by Chidamber and Kemerer.

3.1.3 Complexity metrics

The definition of the McCabe cyclomatic complexity metric (MCC) is based on the control flow graph of a piece of code, e.g., a method: $MCC = e - n + 2p$, with e the number of edges in the graph, n the number of nodes and p the number of connected components / unconnected parts of the graph ([4], [6]). In some textbooks ([2], [3]) the definition $MCC = e - n + 2$ is given, assuming that p is equal to 1. An interesting observation is that nowhere in the cited literature it is specified how to extract a control flow graph from a piece of source code, related to the McCabe cyclomatic complexity. In his original paper, McCabe refers to other work for creating the program control flow graph.

3.2 Metrics definitions in tools

As (more or less) different definitions for metrics can be found in the literature, it is desirable that tools that implement a set of metrics make clear what definition has been used for implementation. All tools provide some kind of definitions in help-files, pop-up screens or supporting websites. In this paragraph examples of these definitions are shown. In some cases a difference between the definition given in a tool and the definition found in the literature is observed.

3.2.1 Size metrics

Not all metrics are explained in all tools: almost no definitions are found regarding how to count the number of files, classes or methods (NFL, NCL, NMT). Apparently this is supposed to be clear. RSM shows a very brief comment on counting files:

“RSM counts the files processed and generates metrics based on the file extension.”

SourceMonitor provides some kind of definition for counting classes and methods:

“Classes are counted on the basis of their declarations. That is, SourceMonitor looks for the "class <class name> {" or "interface <interface name> {" sequences and extracts the class names. Interfaces and classes are counted together. Anonymous inner classes are also counted. Once inside a class or interface, the "<method name>(...) {" construction identifies methods.”

The various line counts are more or less defined in the toolset. As expected, the definitions for the number of physical lines of code (pLOC) are clear – if present:

“LOCphy for a file is the number of physical lines in that file. LOCphy is equal to the number of new line characters in the file, or that amount + 1, if the file does not end with a new line. For a class, interface and method LOCphy is the number of lines that the construct occupies.” (CMTJava)

“Effectively, a count of the number of new lines in the project.” (Essential Metrics)

Counting comment lines seems straightforward and hence leads to short definitions:

“The lines that contain comments, either C style (*/*...*/*) or C++ style (*//...*) are counted.” (SourceMonitor)

“... counts all lines that contain regular comments and Javadoc comments where empty lines within both kinds of comments are also counted.” (RefactorIT)

“Comment lines ... include lines that have both code and comments on the same physical line.” (RSM)

The definitions for more specific line counts (eLOC, sLOC, ILOC) are often vague, or have a strong operational character.

“An effective line of code or eLOC is the measurement of all lines that are not comments, blanks or standalone braces or parenthesis.” (RSM)

“Logical lines of code represent a metrics for those lines of code which form code statements. These statements are terminated with a semi-colon. The control line for the "for" loop contains two semi-colons but accounts for only one semi colon.” (RSM)

“ILOC is a count of the total number of semicolons in the project excluding those within comments and string literals.” (Essential Metrics)

3.2.2 Structure metrics

The Chidamber-Kemerer metrics set is provided by a subset of the investigated metrics tools, ckjm, Essential Metrics and (5 out of 6) RefactorIT. For the WMC metric the definitions are completely in line with each other:

“A class's weighted methods per class WMC metric is simply the sum of the complexities of its methods.” (ckjm)

“WMC is the sum of the Cyclomatic complexity for all methods in the class.” (Essential Metrics)

“The WMC metric is the sum of the complexities of all class methods.” (RefactorIT)

The definitions for the coupling between object classes (CBO) differ. ckjm gives a straightforward definition, which is in line with the original definition. The definition provided by Essential Metrics is a little bit puzzling.

“The *coupling between object classes* (CBO) metric represents the number of classes coupled to a given class (efferent couplings, Ce). This coupling can occur through method calls, field accesses, inheritance, arguments, return types, and exceptions.” (ckjm)

“CBO is calculated by totalling the number of unique types of attribute within each class. The count only includes user-defined object types.” (Essential Metrics)

The following definitions can be found for the metric RFC (response for a class).

“The metric called the response for a class (RFC) measures the number of different methods that can be executed when an object of that class receives a message (when a method is invoked for that object). Ideally, we would want to find for each method of the class, the methods that class will call, and repeat this for each called method, calculating what is called the transitive closure of the method's call graph. This process can however be both expensive and quite inaccurate. In ckjm, we calculate a rough approximation to the response set by simply inspecting method calls within the class's method bodies. This simplification was also used in the 1994 Chidamber and Kemerer description of the metrics.” (ckjm)

“RFC is the number of methods in a class, plus the number of distinct methods called by those methods.” (Essential Metrics)

“The response set of a class is the set of all methods and constructors that can be invoked as a result of a message sent to an object of the class. This set includes the methods in the class, inheritance hierarchy, and methods that can be invoked on other objects.” (RefactorIT)

From the definition text it is clear that ckjm does not compute the complete transitive closure. The definition given by Essential Metrics suggests the same restriction. The RefactorIT definition is less clear. It at least suggests that the complete response set is calculated, without explicitly stating it. The definition given by Essential Metrics is not completely according to the definition given by Chidamber and Kemerer ([1]). If a method calls a method of the same class, this called method should not be counted.

For a more complex metric like LCOM the definitions differ significantly:

“LOCM¹ is calculated by building a list of member variables and a count of the number of references to each variable (usage) in all methods of this class. Then, a sum of the ratios of usage / TotalMethods is calculated. Finally, LOCM is returned as this SumOfRatios / TotalAttributes.” (Essential Metrics)

¹ Essential Metrics uses the slightly different abbreviation LOCM.

“A class's *lack of cohesion in methods* (LCOM) metric counts the sets of methods in a class that are not related through the sharing of some of the class's fields. The original definition of this metric (which is the one used in *ckjm*) considers all pairs of a class's methods. In some of these pairs both methods access at least one common field of the class, while in other pairs the two methods do not share any common field accesses. The lack of cohesion in methods is then calculated by subtracting from the number of method pairs that don't share a field access the number of method pairs that do. Note that subsequent definitions of this metric used as a measurement basis the number of disjoint graph components of the class's methods. Others modified the definition of connectedness to include calls between the methods of the class. The program *ckjm* follows the original (1994) definition by Chidamber and Kemerer.” (ckjm)

Essential Metrics calculates a ratio, whereas *ckjm* calculates the cardinality of a set. It should be noticed that the cited text from *ckjm* signals the same variation in the definitions of LCOM as we have found in the literature: $LCOM = |P| - |Q|$ vs. $LCOM = |P|$, see paragraph 3.1. Unfortunately, the tool RefactorIT does not provide a definition for LCOM.

3.2.3 Complexity metrics

The following “definitions” can be found for the McCabe cyclomatic complexity:

“The complexity metric is counted approximately as defined by Steve McConnell in his book *Code Complete*, Microsoft Press, 1993, p.395. The complexity metric measures the number of execution paths through a function or method. Each function or method has a complexity of one plus one for each branch statement such as **if**, **else**, **for**, **foreach**, or **while**. Arithmetic if statements (`MyBoolean ? ValueIfTrue : ValueIfFalse`) each add one count to the complexity total. A complexity count is added for each '&&' and '||' in the logic within **if**, **for**, **while** or similar logic statements. Switch statements add complexity counts for each exit from a case (due to a **break**, **goto**, **return**, **throw**, **continue**, or similar statement), and one count is added for a **default** case even if one is not present. Each **catch** or **except** statement in a try block (but not the **try** or **finally** statements) each add one count to the complexity as well.” (SourceMonitor)

“The cyclomatic complexity $V(G)$ counts the number of branches in the body of the method defined as:

- if statements
- conditions such as && and ||
- for statements
- while statements ” (RefactorIT)

“Cyclomatic complexity is the degree of logical branching within a function. Logical branching occurs when a "while", "for", "if", "case" and "goto" keywords appear within the function. Cyclomatic complexity is the count of these constructs.” (RSM)

“Cyclomatic Complexity ($v(G)$) is a measure of the complexity of the method's decision

structure. It is the number of linearly independent paths and therefore, the minimum number of paths that should be tested.” (Essential Metrics)

“The McCabe Cyclomatic number $v(G)$ describes the complexity of the control flow of the program. For a single method, $v(G)$ is one less than the number of conditional branching points in the method.” (CMTJava)

All these definitions focus on the way the cyclomatic complexity should be calculated: by counting the number of branching statements / decision points. In the literature, the program control flow graph is the basis for the cyclomatic complexity. The citations presented above at least suggest that no program control flow graph of the code is constructed. So, it is questionable *what* exactly is measured by the various tools when referring to the McCabe cyclomatic complexity.

Furthermore it should be noticed that the definitions are different with respect to counting conditional statements: SourceMonitor and RefactorIT separately count each atomic part of a composed condition, where the other tools only (seem to) count the conditional statement as a whole.

SourceMonitor is the only tool that mentions how switch statements and try-catch statements are counted.

Finally, CMTJava counts the number of conditional branching points and subtracts one from the result. This results in a cyclomatic complexity of -1 for a method without any branching point. However, in general such a method is considered to have a cyclomatic complexity of +1. It is not clear why this deviant definition has been chosen.

4 Measured values, analysis

A small student project has been fed to the six selected tools. In this paragraph the measurement results obtained are presented and analyzed. Size metrics, structure metrics and complexity metrics are handled in separate subparagraphs.

4.1 Size metrics

The tool ckjm operates on .jar files, not directly on source code. Therefore, this tool does not deliver size metrics. Feeding the small student project to the five other tools produced the results given in Table 1. The numbers given are aggregated values for the whole student project. The values in the rightmost column are derived from a visual inspection of the source code of the project, supported by basic Unix scripting (cat, grep, ls, wc). See paragraph 2 for an explanation of the abbreviations used for the various metrics.

Table 1. Measured values for basic size metrics

metrics	Source Monitor	RefactorIT	RSM	Essential Metrics	CMTJava	Visual Insp.
NFL	24	24	24	24	24	24
NCL	45	41	41	25	41 (24 top)	41
NMT	186	179	180	97	179	179
LOC - pLOC	3859	3860	3864	3860	3864	3860
LOC - eLOC	n/a	n/a	2316	n/a	2746	n/a
LOC - sLOC	2238	n/a	n/a	n/a	n/a	n/a
LOC - lLOC	n/a	n/a	1679	1721	1721	n/a
CML	556	556	556	556	556	556

Table 1 shows no differences between the tools in counting files (the NFL metric).

The differences in counting the number of classes and methods (NCL, NMT) were completely unexpected. A more detailed inspection of the measured values showed that Essential Metrics refused to process 8 out of the 24 source code files, due to the fact that these files contained Java version 1.5 constructions like generics. We have not found a restriction with respect to Java versions in The Essential Metrics documentation. The differences between the results of the other tools are small. SourceMonitor counts 4 classes and 7 methods that are not there (probably double counting), RSM counts one method too many in one class.

The measured values show small differences in counting physical lines of code (pLOC). A more detailed analysis shows that these differences are caused by different ways of counting “end-of-line” and “end-of-file” characters at the end of a file. The values measured with Essential Metrics are completely in line with the visual inspection results, for each separate file.

Measurement results for effective LOC, statement LOC and logical LOC show a certain variation. This apparently has its origin in the lack of a widely accepted definition of these metrics. They seem to be more or less tool-dependent.

Remarkably, the count of comment lines (CML) is the same for all tools, even in detail (for each separate file).

4.2 Structure metrics

Only three tools in the selected set (ckjm, Essential Metrics and RefactorIT) are capable of producing the six Chidamber-Kemerer metrics. RefactorIT shows five of them, the metric CBO (Coupling Between Objects) is absent. Table 2 shows the measurement results for four classes taken from the small student project. For each class also the size (physical lines of code) is indicated.

Table 2. Measured values for the Chidamber-Kemerer metrics set

metrics	Class 1 (16 pLOC)			Class 2 (120 pLOC)			Class 3 (226 pLOC)			Class 4 (234 pLOC)		
	ckjm	EM	RefIT	ckjm	EM	RefIT	ckjm	EM	RefIT	ckjm	EM	RefIT
WMC	3	3	3	17	17	17	13	43	68	14	32	38
DIT	1	0	1	1	0	1	2	0	2	0	1	3
NOC	0	0	0	0	0	0	1	1	1	0	0	0
CBO	0	0	n/a	2	0	n/a	15	0	n/a	7	0	n/a
RFC	4	3	0	28	18	16	23	23	19	27	25	22
LCOM	1	0	0	74	57	0,825	46	42	0,933	27	10	0,811

There is only one metric (number of children of a class, NOC) for which the measured values are identical. All other metrics show a (sometimes wide) variation.

This variation is partially expected for WMC (the weighted method complexity) as the definitions of Essential Metrics and RefactorIT differ (see previous paragraph). A short investigation of the measured values in combination with other metrics obtained from the student project shows that for Class 1 and Class 2 the WMC values are as expected. Class 1 holds three simple methods with complexity 1 each. Class 2 holds 15 methods, 14 have complexity 1, one has complexity 3. Class 3 has 13 methods, many methods having a complexity > 1. Class 4 has 14 methods, mostly with complexity > 1. It appears that for Class 3 and Class 4 ckjm only has counted the number of methods, not their complexity.

As Essential Metrics is not capable to process all files in the project, it can be expected that the values for the depth of the inheritance tree (DIT) are not accurate. For ckjm and RefactorIT, the measured values are the same for Class 1, 2 and 3, but not for Class 4. A

visual inspection of the source code shows that the value measured by RefactorIT is correct. No explanation has been found for the value measured by ckjm.

With ckjm the coupling between object classes (CBO) is measured according to the definition given by the tool. Essential Metrics measures a value of 0 for each of the classes. As the definition of this metric provided by the tool is unclear, it is impossible to say anything about the rationale of this value.

The response set for a class (RFC) gives different values for all three tools, for all four classes. We notice that ckjm always gives the highest value, and RefactorIT the lowest value. The value of Essential Metrics is in between these two values. Visual inspection shows that the values given by RefactorIT are too low for all four classes investigated. In one case (Class 2) the difference is nearly 50%. It is difficult to give an explanation for the differences. Class 2 and Class 4 contain inner classes, and the literature does not state anything about inner classes. Could it be that one tool counts the methods of the inner class, and the other does not? Some values are clearly wrong. For instance, Class 1 is a small class containing three methods that do not call other methods. Therefore, the RFC should be 3. RefactorIT seems to have problems with Class 1, since both RFC and LCOM are zero. Class 3 is the only class that ckjm and Essential Metrics agree upon. The RFC value for this class has been manually verified and is correct.

As stated in paragraph 3.2.2, the tool definitions for the Lack of Cohesion of Methods (LCOM) differ significantly. This explains why the three tools give different values, in some cases with large differences. The values given by RefactorIT are fractions. This is remarkable, because the definition in the literature states that LCOM is a count (the cardinality of a set), and a count is not the same as a fraction. No further conclusions can be given on the values of RefactorIT, because no tool definition is available. In Table 2 it is shown that the Essential Metrics values are whole numbers. This is surprising, since the definition of Essential Metrics should produce fractions. According to paragraph 3.2.2 ckjm follows the original definition of Chidamber and Kemerer. However, manual verification showed that the ckjm values are slightly off. This can be explained by the fact that ckjm does not count inner classes.

4.3 Complexity metrics

Five tools in the selected set (SourceMonitor, Essential Metrics, RSM, RefactorIT and CMTJava) measure the McCabe cyclomatic complexity. The table below shows the measurement results for 16 files from the reference student project. Eight files were left out because they could not be processed by Essential Metrics.

Table 3. Measured values for the McCabe cyclomatic complexity

File	Source Monitor	Essential Metrics	RSM	RefactorIT	CMTJava
1	52	39	50	50	43
2	50	40	50	50	36
3	75	43	68	68	56
4	7	7	7	7	1
5	4	4	4	4	1
6	4	4	4	4	1
7	24	23	24	24	9
8	12	19	18	18	12
9	3	3	3	3	1
10	9	5	5	9	9
11	5	5	5	5	1
12	6	6	6	6	1
13	2	2	2	2	1
14	4	3	3	3	2
15	1	1	1	1	1
16	3	3	3	3	1

The cyclomatic complexity has also been measured by visual inspection of the code. For all files minus one the value appeared to be equal to the value measured by Essential Metrics. In one file the difference was caused by counting default branches in switch statements. For RefactorIT all differences could be traced back to a different way of counting composed conditions, containing “&&” and “||” operators. This could be expected from the definition of cyclomatic complexity, given by RefactorIT. From the table it is very likely that RSM has implemented the complexity count in the same way, although this is not explicitly defined (see paragraph 3.2.3). The only difference between RSM and RefactorIT, for file number 10, is caused by RSM not counting a “try ... catch” statement, where RefactorIT does. It should be noticed that Essential Metrics does not count this type of statement either, where SourceMonitor and CMTJava seem to do.

SourceMonitor (in some cases) and CMTJava (in almost all cases) measure values for MCC that are different from the expected values. As noticed in paragraph 3, CMTJava has its own definition for MCC, so different values could be expected.

5 Conclusions

In this paper we have identified three different “worlds” for software metrics: a literature world with textbooks and papers, and a tool world which can be split in two: a metrics definition world (what the tool *says*) and a metrics implementation world (what the tool *does*). Unfortunately, the three worlds do not yield uniform results.

Variations in metrics definitions between metrics literature and metrics tools are caused by several reasons:

- The literature gives abstract definitions (using concepts like a control flow graph for measuring cyclomatic complexity) and/or vague definitions (e.g. for various line counts).
- Sometimes a metrics definition changes over time, e.g., the LCOM definition.
- The literature definitions lack language-specific clarifications (e.g., what to do with inner classes, try-catch statements).

- Sometimes a tool chooses its own definition for a metric (e.g., LCOM definition in Essential Metrics).

Variations in measured values between different tools are caused by various reasons:

- A tool is not capable of handling all constructs of a certain language (e.g., Java 1.5 and Essential Metrics)
- A tool uses its own definition of a metric, different from other tools and / or the literature (e.g., lack of cohesion of methods and Essential Metrics, cyclomatic complexity and CMTJava)
- A tool contains implementation errors (e.g., SourceMonitor for the number of classes and the number of methods)

The results reported in this paper show that the world of software metrics is far from mature. Measured values for software metrics are tool-dependent. Sometimes the differences with the expected values are small (less than 1%). However, sometimes the differences are more than 10% and cannot be neglected. This means that one has to be cautious when using values produced by software metrics tools. When comparing measurement values for different software products, the tools that have been used should be taken into account. One has to be sure that both tools measure a particular metric in the same way. To be on the safe side, it is recommended that only measurement values obtained from one and the same tool should be used for comparisons.

In line with the research reported by Lincke *et. al.* ([5]), in many cases an explanation could be found for a particular measured value. E.g., for most of the inspected tools measuring the cyclomatic complexity (MCC) gave results that could be expected from the definitions given in the tools. However, in other cases the results were merely unpredictable, e.g., for the response for a class (RFC) or the lack of cohesion of methods (LCOM).

Improvement of the situation described above requires actions in different directions.

First of all, in the literature about software metrics *accurate* and *operational* definitions are required that can be implemented in a straightforward way. The definitions have to be accurate, covering all different cases that will have to be handled when implementing a certain metric. This definition should be widely accepted.

Next, tool constructors/vendors have to be clear what definitions of metrics have been implemented in their tool. Most tools do provide some kind of definition, but in many cases additional effort will be required to improve the texts. These definitions should be accurate and operational as well. As each tool is geared to one or more programming languages, in many cases such a definition will contain a generic part and a language-specific part.

Finally, the implementation of a metric in a tool has to be correct with respect to the definition provided by the tool.

References

- [1] Chidamber, S.R., Kemerer, C.F.: A Metrics Suite for Object oriented Design. In: IEEE Transactions on Software Engineering, vol 20, pp 476 – 493 (1994)
- [2] Fenton, N.E, Pfleeger, S.L.: Software Metrics, A Rigorous & Practical Approach. PWS Publishing Company, second edition (1997)
- [3] Jones, C.: Applied Software Measurement. McGrawHill (1991)
- [4] Kan, S.H.: Metrics and Models in Software Quality Engineering. Addison Wesley (2003)
- [5] Lincke, R., Lundberg, J., Löwe, W.: Comparing Software Metrics Tools. In: Proceedings of the ISSTA'08, pp 131 – 141 (2008)

- [6] McCabe, T.: A Complexity Measure. In: IEEE Transactions on Software Engineering 2(4), pp 308-320 (1976)
- [7] Websites of the tool providers:
ckjm: <http://www.spinellis.gr/sw/ckjm/>
CMTJava: <http://www.verifysoft.com/en>
Essential Metrics: <http://www.powersoftware.com/em/>
RefactorIT: <http://sourceforge.net/projects/refactorit>
RSM: <http://msquaredtechnologies.com/m2rsm/>
SourceMonitor: <http://www.campwoodsw.com/sourcemonitor.html>

Quality of Process Control in Software Projects

Yegor Bugayenko

TechnoPark Corp.
568 Ninth Street South 202
Naples, Florida 34102
egor@tpc2.com

Abstract. The method described in the article consolidates and automates the mechanism of quality of process (QoP) measurement in software development projects. Quality policy is defined as a collection of QoP requirements, with respect to software modules, that automatically calculate the value of each QoP metric. The weighted average of all metrics is a QoP indicator, which is used for the project quality control and team motivation. An experimental list of QoP requirements based on CMMI recommendations is presented.

Keywords: Quality of Process, Process Improvement, Quality Control

1 Introduction and Problem Statement

Quality of Process (QoP) in software projects is measured as compliance to a set of pre-defined process standards and requirements. There are a number of industry-wide process standards; while some of them are more precise, others less. Good examples of such standards are CMMI, PMBOK, RUP, MSF, Scrum, PRINCE2, ISO-9001, IEEE standards, UML, and coding rules and conventions.

The selection of the right set of standards, applicable to a given project, is a complicated task for a project manager. However, a much more difficult task is to keep the project (team, artifacts, results, processes) compliant to the selected set of standards during the entire project life-cycle. What is almost impossible for the majority of projects is a regular reporting of the level of compliance. Senior management and project stakeholders want to know how compliant the given project is to the standards required. In other words, what is the QoP on a numeric scale.

Without such a consolidated QoP measurement, neither the project manager nor project stakeholders can get a clear understanding of how compliant the project is to the standards desired.

The heterogeneous nature of project artifacts and lack of formality in existing standards are the two biggest difficulties in the automation of QoP. The article suggests a method of automated QoP calculation and gives an example of QoP requirements for one particular CMMI process area.

2 The Solution, a Formal Approach

The instant value of QoP is defined as $Q(t)$ in $[0; 1]$ interval, calculated at any moment of time t as a weighed average:

$$Q(t) = \frac{\sum_i^n w_i \times R_i(t, A)}{\sum_i^n w_i}, \quad t_0 < t < t_1, \quad R_i \in R \quad (1)$$

Where n is the total number of QoP requirements defined in the project, and R_i is a function in $[0; 1]$ interval, indicating the compliance of project artifacts A to an i -th requirement at a given moment of time t in the project schedule. It is assumed that the project schedule is baselined, starts at t_0 , and the finish is planned on t_1 . w_i is the weight rank of an i -th requirement, indicating the importance of this particular requirement in the entire formula.

Higher values of $Q(t)$ and R_i mean better compliance to requirement(s), while lower values mean the opposite. The process completely complies with the given set of requirements R iff $Q(t)$ equals to 1.

During the project life-cycle, $Q(t)$ will have different values, as shown in the graph in Figure 1.

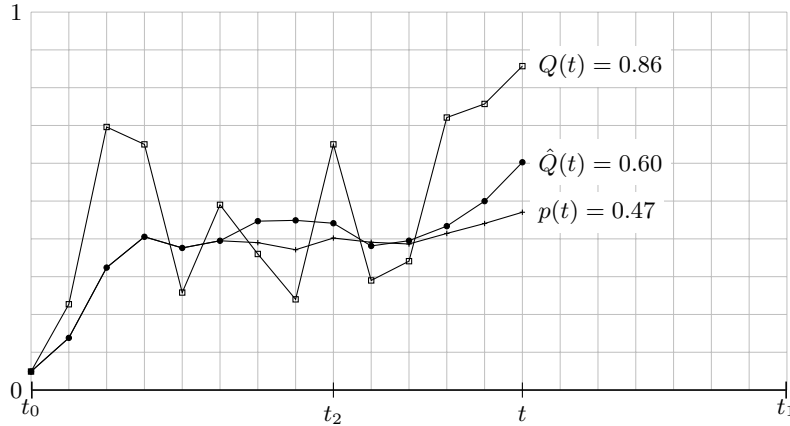


Fig. 1. Sample graph showing the dynamic of $Q(t)$ changing during the project life-cycle and the running average $\hat{Q}(t)$, which is a less pick-sensitive and more accurate indicator of the Quality of Process. $p(t)$ is the probability of the achievement of project objectives, according to the defined standard R .

The running average for the quarter of total schedule duration in equation (2) more accurately indicates the Quality of Process $\hat{Q}(t)$.

$$\hat{Q}(t) = \frac{\int_{t_2}^t Q(t)dt}{t - t_2}, \quad t_2 = t - \frac{t_1 - t_0}{4} \quad (2)$$

It is clear that $p(t)$ in equation (3) is an indicator of process compliance to a given quality standard, expressed by a set of requirements R , at any give moment of time t :

$$p(t) = \frac{\int_{t_0}^t Q(t)dt}{t - t_0} \in [0; 1] \quad (3)$$

If the quality standard R in its maximum application guarantees a successful achievement of project objectives¹, then $p(t)$ is an instant probability at the given moment of time t of such an event, as visually indicated in Figure 1.

The project will achieve its objectives if all of followings statements are true:

1. The quality standard guarantees success if applied in its maximum
2. The set of requirements R is derived from the quality standard
3. $\forall t \in [t_0; t_1] : p(t) = 1$

The definition and analysis of quality standards is out of scope of the article. There are a number of mature quality standards, which independently or combined give a guarantee of the project success, if being applied to their maximum.

The calculation of $p(t)$ is a technical task that could be easily accomplished, if all $R_i(t, A)$ are formally defined and automated.

It is necessary to define and automate a set of requirements R that will analyze project artifacts A from different points of view. Section 4 shows how artifacts could be automated in the project, and requirements could be defined on such a set of artifacts, according to CMMI verbal recommendations.

3 Technical Implementation

Technically, the method explained above is implemented as an endless cycle between the automatic “Calculator of QoP,” “QoP Rate Indicator,” and “Motivation System,” as shown in Figure 2. The Calculator uses the database of artifacts and a set of requirements as software functions.

Using the automated requirements, the Calculator collects measures from artifacts. Section 4 explains how automated artifacts provide such homogeneous measures and how automated requirements are defined in order to process the measures. The result of this process is a set of R_i required in equation (1). The Calculator produces $Q(t)$ and passes it to the QoP Rate Indicator.

¹ Achievement of project objective does not always mean the successful delivery of the product, but it always mean the closure of the project according to the baseline (scope, cost, risks, schedule, quality).

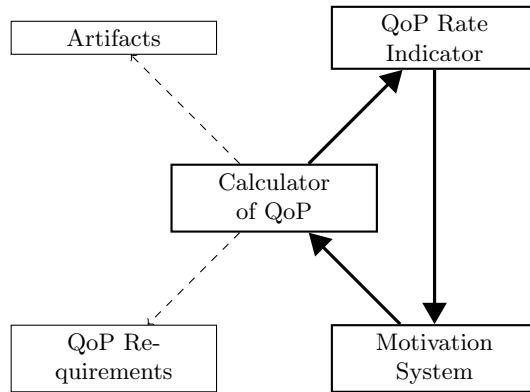


Fig. 2. Technical implementation of the method explained in Section 2, which includes three process components (Calculator of QoP, QoP Rate Indicator, and Motivation System) and two data components (Artifacts and QoP Requirements).

The QoP Rate Indicator distributes the calculated values to project stakeholders in a format convenient for reading and understanding (web page, document, e-mail, etc.) The indication could be either compact, like values of $Q(t)$, $\hat{Q}(t)$, and $p(t)$, or detailed with a full list of R_i along with their measures and explanations.

Values of QoP indicators should impact the motivation of project stakeholders, as defined in Motivation System. As shown above the value of $p(t)$ is a probability of project success.

The last and the most important process flow is between the Motivation System and the Calculator. Process engineers from Software Engineering Process Group (SEPG) or Project Management Office (PMO) should have a negative motivation for high values or QoP in each project.

4 Automated Artifacts and Their Metrics

CMMI, in its latest edition [1], breaks down the software development process into 22 process areas, and each of them may be automated with QoP requirements. The list of requirements below automates the “Configuration Management (CM)” process area. All of these requirements are validated by software modules in a fully automated mode.

“Repository Structure Is Baseline” (SP1.1) A structured list of configuration items exists and is approved. Every configuration item has a unique identifier (file name or URL), type, and an owner. The list is approved by the project manager and is not changed by anyone afterwards.

“ACL Is Baseline” (SP1.2) An Access Control List (ACL) of the repository exists and is approved, including roles and permissions. The ACL is approved by the project manager.

- “SCM Engine Is Live” (SP1.2) The Software Configuration Management (SCM) system is working and is in a consistent state. The consistency of SCM is validated by its built-in engine mechanisms. Most industry SCM systems have such a mechanism.
- “Change Requests Database Is Live” (SP1.2) The front-end for change request registration and the back-end for change requests storage are working and are in a consistent state.
- “Archived Copy Consistency” (SP1.2) A backup/archive copy of the repository content is available, fresh, and consistent.
- “Release Ownership Validity” (SP1.3) The owner of every release should be either a project manager or some other project team member with relative rights granted.
- “Mean Time to Produce a Release” (SP1.3) As suggested in [12] it means how often the project releases deliverables.
- “Justification of Changes” (SP2.1) The rate of changes that are back-traced to change requests. A high rate is desired.
- “Granularity of Changes” (SP2.1) The mean number of changes made per change request [13]. A low granularity is desired.
- “Changes are Made in Branches” (SP2.2) Tge ACL of the repository is configured to prevent changes to the `/trunk` by the project team, while only few people have rights to merge branches/tags with main thread in the repository.
- “Rate of Changes Documenting” (SP2.2) The portion of changes documented by their authors.
- “Quality of Comments” (SP3.1) A text template of change comments is used and all changes comply with this template.
- “Time to do a Configuration Audit” (SP3.2) Regularity of configuration management audits [12]

Applying, calculating and satisfying these requirements to their maximum will indicate that the project performs configuration management well enough to guarantee the success of the outcome.

5 Conclusion

QoP should be measured and controlled in modern software development projects. The long list of currently existing quality standards and development frameworks are not intended for a quantitative measurement of the QoP. The article showed the method of metrics deriving from major software development artifacts.

References

- [1] Software Engineering Institute, CMMI for Development, Version 1.2, CMU/SEI-2006-TR-008, August 2006.

- [2] TechnoPark Corp., Most Popular Software Development And Project Management Industry De-facto Standards (summary), <http://www.technoparkcorp.com/innovations/standards>
- [3] Berenbach,, B., Borotto, G.: Metrics for model driven requirements development, ICSE'06: Proceedings of the 28th international conference on Software engineering, pp. 445–451, Shanghai, China, ACM (2006)
- [4] Nuseibeh, B., Easterbrook, S.: Requirements engineering: a roadmap, ICSE'00: Proceedings of the Conference on The Future of Software Engineering, pp. 35–46, Limerick, Ireland, ACM, New York, NY, USA (2000)
- [5] Cheng, B.H.C., Atlee, J.M.: Research Directions in Requirements Engineering, FOSE'07: 2007 Future of Software Engineering, pp. 285–303, IEEE Computer Society, Washington, DC, USA (2007)
- [6] Kanjilal, A., Sengupta, S., Bhattacharya, S.: Analysis of complexity of requirements: a metrics based approach, ISEC'09: Proceeding of the 2nd annual conference on India software engineering conference, pp. 131–132, Pune, India, ACM (2009)
- [7] Lauenroth., K., Pohl., K.: Towards automated consistency checks of product line requirements specifications, ASE'07: Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, pp. 373–376, Atlanta, Georgia, USA, ACM (2007)
- [8] Kharb, L., Singh, R.: Complexity metrics for component-oriented software systems, SIGSOFT Software Engineering Notes, vol. 33, no. 2, pp. 1–3, ACM, New York, NY, USA (2008)
- [9] Yang., Q., Li., J.J., Weiss, D.: A survey of coverage based testing tools, AST'06: Proceedings of the 2006 international workshop on Automation of software test, pp. 99–103, Shanghai, China, ACM (2006)
- [10] Kim,, Y.W.: Efficient use of code coverage in large-scale software development, CASCON'03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research, pp. 145–155, Toronto, Ontario, Canada, IBM Press (2003)
- [11] Bonja., C., Kidanmariam., E.: Metrics for class cohesion and similarity between methods, ACM-SE 44: Proceedings of the 44th annual Southeast regional conference, pp. 91–95, Melbourne, Florida, ACM (2006)
- [12] Bendix, L., Borracc, L.: Towards a suite of software configuration management metrics, SCM'05: Proceedings of the 12th international workshop on Software configuration management, pp. 75–82, Lisbon, Portugal, ACM (2005)
- [13] Schackmann, H., Lichter, H.: Process assessment by evaluating configuration and change request management systems, WUP'09: Proceedings of the Warm Up Workshop for ACM/IEEE ICSE 2010, pp. 37–40, Cape Town, South Africa, ACM (2009)

A Proposal for an Integrated Measurement and Feedback Environment “*MIERUKA*” for Software Projects Based on the Empirical Study of Three Measurement Domains

Yoshiki Mitani^{1,2}, Hiroshi Ohtaka^{1,3}, Noboru Higuchi¹, Ken-ichi Matsumoto²

¹ Information Technology Promotion Agency, Japan /
Software Engineering Center (IPA/SEC)

²Nara Institute of Science & Technology (NAIST),

³Waseda University

{y-mitani ln-higu }@ipa.go.jp, otaka@fuji.waseda.jp, matumoto@is.naist.jp

Abstract

Various methods of project measurement have been tried to visualize software development project which originally hard to be visible and reflected them to the project management. Such experiments have evolved according to the purpose of measurement and came to constitute three domains; namely, “In-process measurement”, “Benchmarking”, and “Software tagging”. However, there is in actuality only one software development field, and it is unreasonable to complicate measurement demand and feedback activity from various viewpoints. This paper surveys and evaluates empirical studies in these three domains. Then reflecting the result, the authors propose an integrated environment for measurement and feedback.

Keywords

Measurement, empirical software engineering, project management

1 Introduction

Experiments in software project visualization are called Project Dashboard, and advanced trials have been carried out [1]. In particular “in-process measurement”, which measures on-going software development projects and is reflected in project management, opened a new area of research and the usefulness of the method has been reported [2][3]. Moreover, collection and analysis of post-process measurement data, or “Benchmarking”, also have begun by ISBSG, and became active even in Japan and China. And its various usages were considered. Then, a new concept, “Software tagging”, which utilizes such measurement for traceability and accountability was proposed [4]. However, for each project used as the target for measurement, the congestion of various measurement demands complicates the process. Until now, from the viewpoint of empirical software engineering, the authors conducted repeated experiments on various methods of project measurement. In this paper the authors survey and evaluate these experiments and reflecting the result, propose a measurement environment where convenience is felt in each development field with a low load in the actual development project. Additionally, consideration about required requirements is shown.

2 The verification situation of in-process measurement

Authors advanced the visualization project of the IT development project named, “*MIERTKA*”ⁱ, and have produced related tools [5]. This method consists of a "Qualitative approach", a "Quantitative approach", and an "Integrated approach". We divided the software development process into three processes, the upper-stream, the middle-stream, and the lower-stream, and considered these three approaches to each process. The method consists of books, downloadable data, and software tool suits.

There are three steps in the “Qualitative approach”:

1) Certain “Bird’s Eye Views” such as a “Stake Holder Relationships Chart” are described in order to grasp project situation and to extract “Dominant Items” from the target project.

2) The Project Risk is determined using two kinds of project check sheets. One is a questionnaire for self-checking by the project leader consisting of approximately 40 questions and requiring approximately 30 minutes to complete. The other is a questionnaire for a third observer to use to interview and investigate, such as PMO, which consists of approximately 70 questions and requires over two hours. The results of the two questionnaires will be compared.

3) Project failure is avoided by referring to the “Summary of Problem Project”.

In the “Quantitative approach”, it is recommended that the detailed measurement item list for quantitative project management be applied. This list includes over 70 measurement items for each process. The automatic measurement tool named Empirical Project Monitor (EPM) provided in the programming and testing phase of the project is recommended. Using the EPM, for example, source code line transition and bug quantity transition are traceable in real time.

In the "Integrated approach", some excellent link tables that connect related check sheet items, problem project summary data and measurement items are provided. For example, for unusual check results in the check sheet, related past problem project summaries and related measurement items are easy to retrieve. Related check items and past problem project summary are also easy to retrieve from unusual measurement results. The books in which we summarized these methods have sold several thousand copies, and they are referred to in industrial society. Over 70 companies participate in verification experiments using the EPM, and many requests have been forthcoming [6].

The usefulness of project measurement became clear from these activities, and some issues related to the advancement of such measurement became clear. For example, they include following subjects:

- 1) Operations for which the advantage of the measurement is understood well, not only by management personnel but by development field practitioners.
- 2) Discussion about the concentrated or distributed type environment.
- 3) Discussion about self-analysis and consultant type activity.
- 4) Discussion about feedback targets, and a feedback method for analyzed data.

3 Verification situation of benchmarking

IPA/SEC had begun collection and analysis of benchmark data since 2003, and published the results as a data white paper every year. In the 2008 edition of the white paper, over 2000 project data were collected, and the analysis was shown.

4 Verification situation of software tagging

The purpose of a software tag is the realization of accountability and traceability. It maintains the proceeding history of a software development project, in the form of a “Software Tag”, and it is used for explication of the cause of major accidents in respect to software, or conflict and law-suit processes. Moreover, the prior condition that such a history is maintained improves the transparency of a project and contributes indirectly to reliability and productivity. The authors’ reported the first draft standard of the information being saved in a software tag in their research project, StagE [7][8]. Some pilot projects designed to evaluate this are progressing in the industrial society, including some user companies.

5 Proposal for an integrated measurement environment

A proposal for an integrated measurement and feedback environment obtained from the above-mentioned verification activities is shown in Fig. 1. From the viewpoint of a single development field, it is created so that the measurement environment does not become complicated and the unified feedback is realized. The following requirements have become clear in the study.

The main purpose of project visualization through in-process project measurement is smooth project management. Key factors are measurement, analysis and feedback. Collecting process and product data for efficient project management with minimum measurement overhead work is necessary. The manner in which the collected data is analyzed and visualized is also important. The feedback of visualized data is an essential issue.

Benchmarking allows the collection of a great deal of project data, the statistical evaluation of a project, or looks at annual transitions. People who wish to evaluate and guarantee the quality of own project utilize this. Moreover, these data can be utilized for the estimation improvement in accuracy. Although in the past, it was used for post-process evaluation, today, practical in-process use is also considered by use in estimation support and similar past project retrieval.

While the purpose of maintaining traceability is the recording of an invisible software project, the use is analysis of serious software accidents, incidents, conflicts and long-term software maintenance. For all purposes, recorded data are used in the post-process. Key factors are also measurement, analysis and feedback; however, the content is quite different from project management. Measurement items must be useful for investigating serious accidents or resolving incidents and conflicts. The target personnel for information provision are not project managers.

6 Conclusion

Roughly speaking, project measurement came to constitute three domains, and among these, the usefulness of two domains became clear. Moreover, future development is expected also in the 3rd study domain “Software Tag”. In this paper, authors surveyed the verification situation of each domain and clarified issues extracted from it. And it was shown that harmonious environment and composition are required, and a unique structure was proposed from the viewpoint of practical software development that avoids complication.

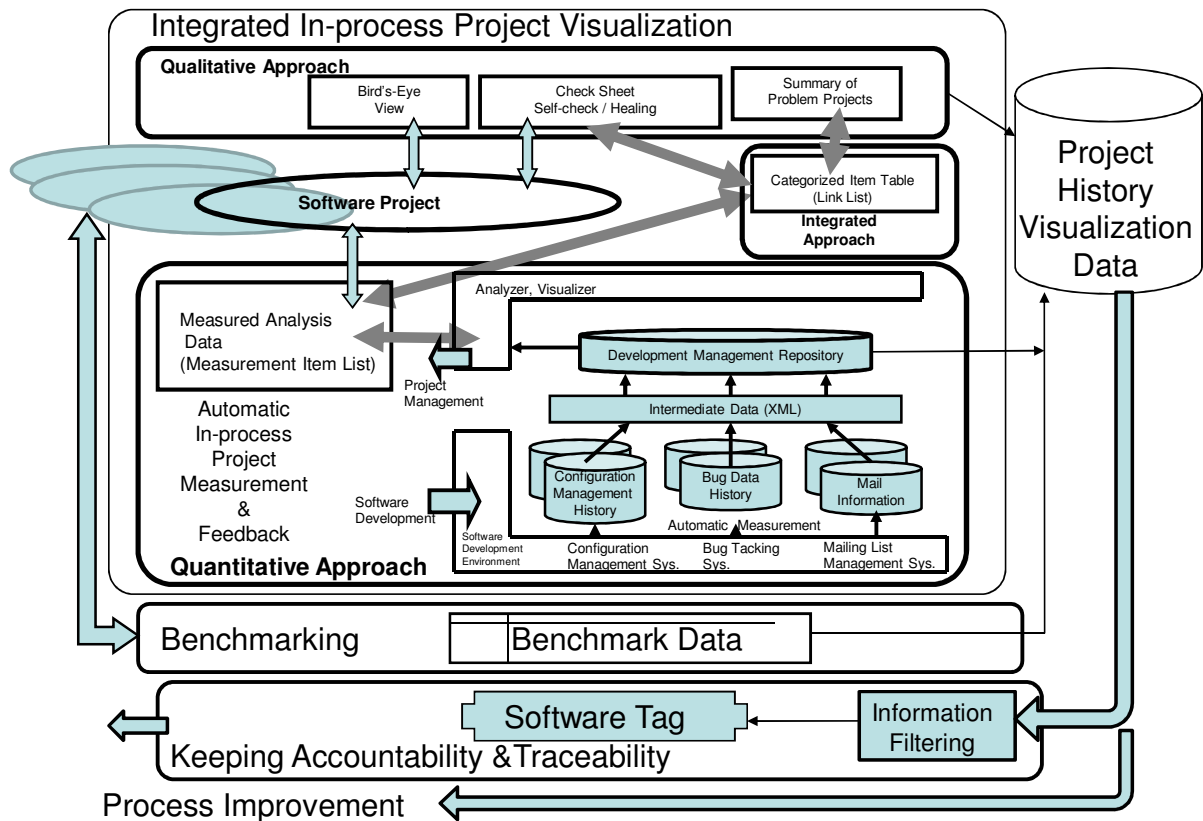


Fig. 1. Integrated Environment for Software Project Measurement and Feedback

Acknowledgments

This work was supported by IPA/SEC, METI and MEXT of Japan. We thank researchers in the SEC and StagE project.

References

- [1] C.Ebert, R.Dumke: Software Measurement, Springer, P.561, 2007
- [2] P. M. Johnson: Requirement and Design Trade-offs in Hackystat: An In-Process Software Engineering Measurement and Analysis System. ESEM 2007, pp.81-90, Madrid, 2007
- [3] M. Ciolkowski, J. Heidrich, F. Simon, M. Radicke: Empirical Results from Using Custom-Made Software Project Control Centers in Industrial Environments, ESEM 2008, pp.243-252, Kaiserslautern, 2008
- [4] K.Inoue: Software Tag: Empirical Software Engineering Data for Traceability and Transparency of Software Project, ATGSE 2007, pp.35-36, Nagoya, 2007
- [5] H. Ohtaka, R. Nagaoka, Visualization of IT Project – *MIERUKA* –, ProMAC2008, Anchorage, 2008
- [6] Y. Mitani, et.al: A Proposal for Analysis and Prediction for Software Projects using Collaborative Filtering, In-Process Measurements and a Benchmarks Database. MENSURA 2006, pp. 98-107 Cadiz, 2006
- [7] http://www.stage-project.jp/en_gaiyou_tag.php?
- [8] K.Inoue: Software Tag Standard 1.0 -Background and Definition-, ATGSE 2008, pp.31-32, Beijing, 2008

Estimating the functional size of applications built with the Oracle eBS Package

Frank Vogelesang, Sogeti Nederland
Bert Veenstra, Cor van Dongen, Johan de Vries, Joyce Span, Hessel Bijlsma
DICTU, Ministry of Agriculture, Nature and Food Quality
(Ministerie van Landbouw, Natuur en Voedselkwaliteit)

Abstract

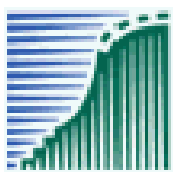
For most of the custom developed software the size estimation method as developed by Sogeti has proven to be very useful. Estimating the functional size of applications that were built in the Oracle e-Business Suite (eBS) package appeared to be difficult. Available eBS modules consist of a huge amount of functionality that can be used to configure or construct desired functionality for the end-users. For the purpose of the Ministry of agriculture, nature and food quality the functionality that is available for the end-user is what should be estimated, not the entire functionality of the available eBS modules.

With a group of eBS and FPA specialists we have been able to distinguish the eBS elements that constitute the available user functionality. Most of these elements can be determined by intelligent queries on the eBS repository. For only two of the characteristics that are used in the estimation method, queries could not give adequate enough results. For these characteristics system documentation had to be used.

This functional size estimation technique has been devised to work equally well in an environment with relatively pure eBS functionality as in environments with a lot of customization within eBS module functionality.

1 Introduction

1.1 Organisational setting : DICTU



DICTU is the IT service department for the Dutch Ministry of Agriculture, Nature and Food Quality. In 2005 DICTU started an Application Portfolio Management (APM) initiative to get a grip on the number of applications available within the Ministry[1]. With APM in place DICTU is taking the next step: charging the different departments on the amount of functionality that DICTU has to maintain for them. This mechanism is based on the functional size of applications, expressed in function points.

In May 2009 a project started to establish the functional size of over 400 applications. This project was headed by Sietse de Jonge, FPA coordinator of DICTU. To be able to finish the project in time for the 2010 budgets an estimation method was used, based on function point analysis and application characteristics, as developed by Sogeti. Over fifty people, mainly from the technical application maintenance departments, were trained to use the estimation method. To assure the quality of the size estimations a review team was trained and put in place.

1.2 Functional size estimation based on application characteristics

This fast FPA estimation technique counts a number of characteristics that can be distinguished in an installed application and translates this data into an estimate of the number of function points (IFPUG/NESMA) that would have been produced by a detailed analysis of functionality. The advantage of this fast FPA estimation technique is that it can produce a size estimate in cases where available documentation is insufficient to make a detailed analysis or when there is insufficient time to make a detailed analysis. To produce such a size estimate knowledge of the application is more important than functional sizing knowledge.



With this in mind, Sogeti developed an FPA estimation tool to assist personnel with good application knowledge and limited functional sizing experience to make a fairly accurate FPA estimation of the functional size[2]. This support tool uses the following application characteristics:

1. Data, divided into two categories:
 - logical tables owned by the application
 - logical tables read from another application
2. External inputs, divided into three categories:
 - screen maintenance functions
 - inbound interfaces
 - background processes initiated by the user
3. External outputs, divided into four categories:
 - screen inquiries
 - reports
 - outbound interfaces
 - unique controls to select field values
4. External inquiries, divided into two categories:
 - inquiries with a unique identifying key
 - help functionality

The estimation tool is enriched with help topics dealing with experience knowledge from earlier size estimations. We have also included several rules of thumb as a first sanity check for the estimator.

The rules of thumb that were incorporated are:

- expected size, based on the number of logical tables
- fraction of the size from logical tables
- expected number of logical tables
- expected number of maintenance functions
- expected number of EI, EO and EQ
- expected number of controls to select field values

These rules of thumb signal when parts of the size estimate falls outside of the expected range. They can assist the estimator to check whether his estimate is complete. They are

also valuable for the reviewer to spot irregular relations that need to be checked in a review of the size estimate. Examples of the output of the size estimation tool can be found in appendices A through E.

1.3 The need for size estimations of eBS applications

For most of the custom developed software the size estimation method has proven to be very useful. Estimating the functional size of applications that were built in the eBS package appeared to be difficult. The eBS modules that are available within the Ministry consist of a huge amount of functionality that can be used to configure or construct desired functionality for the end-users [3]. For the purpose of the APM project the functionality that is available for the end-user is what should be estimated, not the entire functionality of the available eBS modules.

We have been able to distinguish the eBS elements that constitute the available user functionality by combining our eBS and functional sizing experience. The functional size estimation has been devised to work equally well in an environment with relatively pure eBS functionality as in environments with a lot of customization within eBS module functionality. Establishing the functional size in both types of environment is very different [4].

Most of the elements that contribute to the functional size can be determined by intelligent queries on the eBS repository. For only two of the characteristics that are used in the estimation method, queries could not give adequate enough results. For these characteristics system documentation had to be used.

2 Size estimation for eBS applications

2.1 Internal Logical Files in eBS applications

The tables within eBS can roughly be divided into two types: one containing the setup data, describing the package configuration, the other containing the logical data that the users of the system want to store and maintain. In eBS these two categories cannot be distinguished with a query on the database. We have chosen to make the distinction based on the latest modification to the tables.

This means that the moment of distinction must be chosen in such a way that modifications to tables due to setup and implementation lie before this moment and regular mutations lie after this moment. The period between the distinction moment and the query moment must be long enough to allow for all possible types of table transactions due to the use of the functionality. To make the distinction easier all changes from people with exclusively setup or development roles were left out.

The result of this query is the number of physical tables that contain data that the users of the system want to store and maintain. To transform this to the amount of internal logical files we used the experience rule of thumb that the ratio between the number of physical tables and the number of logical files is approximately 3:2.

2.2 External Interface Files in eBS applications

External Interface Files can by definition not be found within Oracle eBS, unless more than one application would have been defined on a single eBS instance. Within the DICTU architecture direct access from eBS to data from other applications is not allowed. Data is always exchanged by means of “koppelvlakken”, logical coupling areas

where the application that owns the requested data puts that data at the disposal of one or more applications that request that data. The “koppelvlakken” are counted as External Inputs for data requests and as External Outputs for data deliveries. For eBS applications no External Interface Files are identified.

2.3 External Inputs in eBS applications

Screen Maintenance Functions

For the screen functions in eBS it is not possible to distinguish screen maintenance functions, screen inquiries and inquiries with a unique identifying key. Vanilla screen functions in eBS always facilitate the full CRUD: Create, Read, Update and Delete. This functionality can be limited by not authorising a part of that functionality for several or all roles. By definition Create, Update and Delete are separate screen maintenance functions. A query, based on used authorisations for all user roles, determines the amount of functions used within the eBS application. To make the query more distinct all authorisations from user roles with exclusive setup or development capabilities were left out. This resulted in the number of CRUD screen functions. The amount of screen maintenance functions for the FPA estimation was derived by multiplying this result by to allow for the assumption that for most of these functions the full CRUD is active. This assumption has been verified in a limited sample.

For the eBS application that contained a lot of customisation based on the QC-module also functions that were queried from tasks were added to the amount of screen maintenance functions. The amount of attributes involved could be queried from the quality plans underlying the tasks.

Inbound interfaces

The amount and classification of inbound interfaces is determined from the system documentation. Inbound interfaces can not easily be determined by querying the eBS repository. The query in the next section does not return all interfaces and cannot be used to differentiate between inbound and outbound interfaces. Due to the effort it would have cost to query this correctly, the limited amount of inbound interfaces and the good documentation of those interfaces we have decided to count and classify them from the interface documentation.

For each inbound interface one External Input is counted even if the user process is split up into multiple steps (for instance copy, data transfer, validate and process). The assumption for this decision is that for the user this is a single logical transaction that for technical reasons has been split up into multiple steps.

Background processes initiated by the user

To determine the amount of background processes the concurrent requests are queried for requests that have an authorisation which is linked to an active user. To make the query more distinct all users with exclusive setup or development roles were left out.

This query does not only return background processes, but also some of the reports, inbound and outbound interfaces are included in the results. These unwanted elements have to be filtered out based on the names of the concurrent requests:

- KV *Abbreviation for “koppelvlak”, logical coupling*
- Report
- Rapport *Dutch for report*
- Summary
- Overzicht *Dutch for summary*

Background processes are counted as external inputs because most of these processes are in this category.

2.4 External Outputs in eBS applications

Screen inquiries

Screen inquiries are the Read parts of the screen functions that implement the screen maintenance functions. The amount of screen inquiries is determined by the same query that is used to determine the amount of screen maintenance functions.

For this category the result of the query is used without multiplication.

Reports

The number of reports is determined by the same query that determines the amount of background processes. For this amount four of the name categories that were excluded for the background processes are the selection criteria for reports:

- Report
- Rapport *Dutch for report*
- Summary
- Overzicht *Dutch for summary*

To this result the number of reports from Oracle Discoverer and the Business Intelligence tool OBI are added.

Outbound interfaces

The amount and classification of outbound interfaces is determined from the system documentation. Outbound interfaces can not easily be determined by querying the eBS repository. The query to determine the amount of background processes does not return all interfaces and cannot be used to differentiate between outbound and inbound interfaces. Due to the effort it would have cost to query this correctly, the limited amount of outbound interfaces and the good documentation of those interfaces we have decided to count and classify them from the interface documentation.

For each outbound interface one External Output is counted even if the user process is split up into multiple steps (for instance validate, copy and data transfer). The assumption for this decision is that for the user this is a single logical transaction that for technical reasons has been split up into multiple steps.

Unique controls to select field values

These kind of controls are implemented in the eBS applications as Lists of Values. Most of the LoV's are not eligible to be counted, according to NESMA counting rules [5]. The assumption is that the LoV's that are based on a query get their data from a logical file and are eligible to be counted. This assumption appears to be valid for both generic eBS applications as for eBS applications that have a high degree of customisation within eBS modules [4].

2.5 External inQuiries in eBS applications

Inquiries with a unique identifying key

Within eBS it is impossible to query the difference between a screen inquiry (EO) and an inquiry with a unique identifying key (EQ). Based on the NESMA definitions the number of External Outputs is usually far greater than the number of External inQuiries.

Therefore we have decided not to differentiate between a screen inquiry and an inquiry with a unique identifying key. All these functions are counted as External Outputs.

Help functionality

The functional size estimation method from Sogeti counts the help functionality on three levels:

- application level
- function/screen level
- field level

To count this functionality three Yes-or-No questions have to be answered. In this respect eBS applications are not treated differently. There is some debate on whether these three levels is not limiting the amount of help functionality too much [6]. Since the contribution of the help functionality is usually very small, this is not considered an important issue for the functional size estimation method.

3 Results

3.1 eBS queries

The queries, as described in the previous chapter, were all created using the following pattern:

- First the used authorisations were selected with the following conditions:
 - a. Exclude all SETUP and PROJECT users
 - b. Select user-ID's $\geq 1,000$
 - c. Select customisation scheme-ID's $\geq 10,000$
- Subsequently select the used applications, based on the selected authorisations
- Subsequently select the tables belonging to the selected applications
- Per table the following information was retrieved:
 - d. Table owner
 - e. Table name
 - f. Number of rows
 - g. Number of columns
 - h. Last update date
 - i. Last updated by

The table owner and table name are used for documentation purposes. The owner and name of the table can be used for verification by the review team and facilitates quick comparison with future estimations of the same functionality.

The number of rows is used to confirm actual usage of the table. If the table is empty, this is a strong indication that it is not a part of the logical functionality.

The number of columns is used to assign the appropriate complexity level to the size estimation element.

The last update date is used to select only those tables that are part of the user functionality. This means that for the currently sized eBS applications the table should be updated within the last 30 days. This value may be different for other implementations. If the table was updated more than 30 days ago, it is likely that this table is part of the setup or is not part of the actual functionality.

The last updated by is used to double-check that only user functionality is counted and that setup and project activities are not counted as user functionality. The vast majority of setup and project activities is filtered out by the first selection step of the query.

3.2 Size estimates for eBS applications based on the queries

The full results of the queries are documented in appendices A through E. The size estimates are summarized in the following table:

	eBS Generic		BTR Generic		BTR 2006		BTR 2007		BTR 2008	
ILF	83	641	9	63	62	443	70	505	70	505
EIF	-	0	-	0	-	0	-	0	-	0
EI	842	2,777	38	142	145	513	188	698	185	685
EO	2,215	9,112	16	80	79	378	84	405	75	361
EQ	1	3	1	3	1	3	1	3	1	3
	12,533		288		1,337		1,611		1,554	

3.3 The size estimation project

In June 53 people mainly from the technical application maintenance departments of the Ministry, were trained to use the estimation method. In addition a review team was trained consisting of 6 people (3.5 FTE), mainly from the Quality department of the Ministry. They had to estimate and review the size of 413 applications before October 1st. Based on experience data from Sogeti this would require from each of them 1 day a week of sizing activity. Each week progress was monitored, both in number of projects sized and the estimated functional size of the portfolio in progress. Early August three full-time size estimators from Sogeti were added to the team and in September three more were added to the team.

The size estimation project was still in progress when this article had to be submitted. Based on the intermediate results the total size of the 413 applications will be around 210,000 FP to 220,000 FP. This means that the portfolio has an average application size of 525 FP, ranging from 40 FP to 12,500 FP.

4. Conclusions

By combining eBS and FPA expertise we have been able to estimate the functional size of a number of applications built with the Oracle eBS package. In this portfolio that would have been virtually impossible by using regular function point analysis.

In less than four months time an application portfolio of 413 applications of which the majority was poorly documented has been sized by using the size estimation approach. This could never have been done by using regular function point analysis.

Although the individual results may be off by 30% of a regular function point analysis we are confident that the end result on the portfolio level is accurate enough for the purposes of the Ministry.

References

- [1] Kees Hakvoort and Pim van der Kleij, APM – More or less the same, 13th Dutch ITSMF congress, October 27-28, 2007, Nijkerk (the Netherlands) www.itsmf.nl
- [2] Marcel Rispens and Frank Vogelezang, Application Portfolio Management basics – How much software do I have, Proceedings of the 4th Software Measurement European Forum (SMEF 2007), may 9-11, Roma (Italy) www.iir-italy.it/smef2007
- [3] Oracle E-Business Suite, www.oracle.com/applications/e-business-suite.html
- [4] Frank Vogelezang, Using COSMIC-FFP for sizing, estimating and planning in an ERP environment, Proceedings of the 16th International Workshop on Software Measurement (IWSM 2006), November 2-3, 2006, Potsdam (Germany) metrieken.sogeti.nl
- [5] NESMA, Definitions and counting guidelines for the application of function point analysis: NESMA Functional Size Measurement method compliant to ISO/IEC 24570, version 2.1, november 2004 www.nesma.nl/section/home
- [6] NESMA Forum, Discussion thread on help functionality, started May 28, 2009 forum.nesma.nl/viewtopic.php?p=64#64

APPENDIX A: eBS Generic functionality



Omvangsschatting FPA op basis van applicatiegegevens

Versie 1.3 LNV

applicatie: **eBS GLB Generiek**
 omvang: **12533 functiepunten**
 ingevuld door: **J.H. de Vries**

		onbekend				
> Gegevens	> Logische systeem eigen tabellen	Meer...	63	Logische tabellen met 50 of minder data elementen		
			20	Logische tabellen met meer dan 50 data elementen		
	> Logische systeem vreemde tabellen	Meer...	onbekend	Logische tabellen met 50 of minder data elementen		
			onbekend	Logische tabellen met meer dan 50 data elementen		
> Invoerfuncties	> Onderhoudsfuncties (toevoegen, wijzigen, verwijderen) <i>tel toevoegen, wijzigen en verwijderen als aparte functies</i>	Meer...	148	9	Schermen en/of functies met maximaal 4 velden	
				27	Schermen en/of functies met 5-15 velden	
				24	Schermen en/of functies met meer dan 15 velden	
	> Inkomende interfaces	Meer...	onbekend	4	Interfaces met maximaal 4 velden	
					Interfaces met 5-15 velden	
					Interfaces met meer dan 15 velden	
	> Batchverwerking / achtergrondprocessen <i>let op dubbelstellingen met de inkomende en uitgaande interfaces</i>	Meer...	onbekend	630	Processen met maximaal 4 data elementen	
					Processen met 5-15 data elementen	
					Processen met meer dan 15 data elementen	
	> Uitvoerfuncties	> Raadpleegfuncties	Meer...	onbekend	49	3
					9	Schermen en/of functies met 6-20 velden
					8	Schermen en/of functies met meer dan 20 velden
> Rapportages		Meer...	onbekend	151	2	Rapportages met maximaal 5 velden
					11	Rapportages met 6-20 velden
					2	Rapportages met meer dan 20 velden
> Uitgaande interfaces		Meer...	onbekend	2		Interfaces met maximaal 5 velden
						Interfaces met 6-20 velden
						Interfaces met meer dan 20 velden
> Unieke select controls om velden in te vullen		Meer...			1.978	Aantal unieke select controls binnen de applicatie
> Opvragingsfuncties	> Opvragingsfuncties op basis van een unieke sleutel	Meer...	onbekend			Functies met maximaal 5 velden
						Functies met 6-20 velden
						Functies met meer dan 20 velden
	> Helpfuncties	Meer...			N	Helpfunctie op applicatieniveau (J/N)
					N	Helpfunctie(s) op schermniveau (J/N)
					J	Helpfunctie(s) op veldniveau / tooltips (J/N)
> Vuistregels	> Deze vuistregels zijn bedoeld als hulpmiddel:					Hiermee kan nagegaan worden of de omvangsschatting compleet is.
	> Als de omvangsschatting niet aan de vuistregels voldoet:					Dit betekent niet zonder meer dat de omvangsschatting fout is. Wel is het verstandig om na te gaan waarom voor EBS GLB GEN deze vuistregel niet opgaat.
	> Omvang op basis van het aantal logische tabellen:	Meer...				Op basis van het aantal systeemeigen logische tabellen van en het aantal logische tabellen dat wordt gelezen wordt een totale omvang verwacht voor tussen de 2075 en de 3586 functiepunten. De schatting op dit moment is 12533 functiepunten.
	> Bijdrage van logische tabellen aan de omvang:	Meer...				De bijdrage van de logische tabellen aan de totale omvang is in de regel niet meer dan 40%. Die bijdrage is op dit moment 5%.
	> Het aantal logische tabellen:	Meer...				Op basis van het aantal onderhoudsfuncties en/of schermen wordt een aantal interne logische gegevensverzamelingen verwacht dat ligt tussen de 84 en 336. Op dit moment zijn er 83 ingevuld.
	> Het aantal onderhoudsfuncties:	Meer...				Op basis van het aantal interne logische gegevensverzamelingen wordt een aantal onderhoudsfuncties/schermen verwacht dat ligt tussen de 125 en 498. Op dit moment zijn er 208 ingevuld.
	> Het aantal invoerfuncties:	Meer...				Op basis van het aantal uitvoerfuncties en opvragingsfuncties worden ongeveer 1927 invoerfuncties verwacht. Op dit moment zijn er 842 ingevuld.
	> Het aantal uitvoerfuncties:	Meer...				Op basis van het aantal invoerfuncties en opvragingsfuncties worden ongeveer 967 uitvoerfuncties verwacht. Op basis van het aantal opvragingsfuncties worden ongeveer 11 uitvoerfuncties verwacht. Op dit moment zijn er 2215 ingevuld.
	> Het aantal opvragingsfuncties:	Meer...				Op basis van het aantal uitvoerfuncties worden ongeveer 201 opvragingsfuncties verwacht. Op dit moment zijn er 1 ingevuld.
	> Het aantal select controls:	Meer...				Op basis van het aantal onderhoudsschermen, onderhoudsfuncties en raadpleegfuncties wordt verwacht dat het aantal select controls niet hoger is dan 237. Op dit moment zijn er 1978 select controls ingevuld.

In the generic functionality the standard working process for the entire Ministry has been implemented.

APPENDIX B: eBS BTR Generic functionality



Omvangsschatting FPA
op basis van applicatiegegevens

Versie 1.3 LNV

applicatie: eBS GLB BTR Generiek

omvang: 288 functiepunten

ingevuld door: J.H. de Vries

		onbekend		
> Gegevens	> Logische systeem <i>eigen</i> tabellen	Meer...	9	Logische tabellen met 50 of minder data elementen Logische tabellen met meer dan 50 data elementen
	> Logische systeem <i>vremde</i> tabellen	Meer...		Logische tabellen met 50 of minder data elementen Logische tabellen met meer dan 50 data elementen
	> Onderhoudsfuncties (toevoegen, wijzigen, verwijderen) <i>tel toevoegen, wijzigen en verwijderen als aparte functies</i>	Meer...	3 24	Schermen en/of functies met maximaal 4 velden Schermen en/of functies met 5-15 velden Schermen en/of functies met meer dan 15 velden
	> Inkomende interfaces	Meer...	4	Interfaces met maximaal 4 velden Interfaces met 5-15 velden Interfaces met meer dan 15 velden
> Batchverwerking / achtergrondprocessen <i>let op dubbeltellingen met de inkomende en uitgaande interfaces</i>	Meer...	7	Processen met maximaal 4 data elementen Processen met 5-15 data elementen Processen met meer dan 15 data elementen	
> Uitvoervuncties	> Raadpleegfuncties	Meer...	1 8	Schermen en/of functies met maximaal 5 velden Schermen en/of functies met 6-20 velden Schermen en/of functies met meer dan 20 velden
	> Rapportages	Meer...	2 2 2	Rapportages met maximaal 5 velden Rapportages met 6-20 velden Rapportages met meer dan 20 velden
	> Uitgaande interfaces	Meer...		Interfaces met maximaal 5 velden Interfaces met 6-20 velden Interfaces met meer dan 20 velden
	> Unieke select controls om velden in te vullen	Meer...	1	Aantal unieke select controls binnen de applicatie
	> Opvragingsfuncties	Meer...		Functies met maximaal 5 velden Functies met 6-20 velden Functies met meer dan 20 velden
	> Helpfuncties	Meer...	N N J	Helpfunctie op applicatieniveau (J/N) Helpfunctie(s) op schermniveau (J/N) Helpfunctie(s) op veldniveau / tooltips (J/N)
> Vuistregels	> Deze vuistregels zijn bedoeld als hulpmiddel:			Hiermee kan nagegaan worden of de omvangsschatting compleet is.
	> Als de omvangsschatting niet aan de vuistregels voldoet:			Dit betekent niet zonder meer dat de omvangsschatting fout is. Wel is het verstandig om na te gaan waarom voor EBS GLB BTR deze vuistregel niet opgaat.
	> Omvang op basis van het aantal logische tabellen:	Meer...		Op basis van het aantal systeemeigen logische tabellen van en het aantal logische tabellen dat wordt gelezen wordt een totale omvang verwacht voor tussen de 225 en de 252 functiepunten. De schatting op dit moment is 288 functiepunten.
	> Bijdrage van logische tabellen aan de omvang:	Meer...		De bijdrage van de logische tabellen aan de totale omvang is in de regel niet meer dan 40%. Die bijdrage is op dit moment 22%.
	> Het aantal logische tabellen:	Meer...		Op basis van het aantal onderhoudsfuncties en/of schermen wordt een aantal interne logische gegevensverzamelingen verwacht dat ligt tussen de 5 en 18. Op dit moment zijn er 9 ingevuld.
	> Het aantal onderhoudsfuncties:	Meer...		Op basis van het aantal interne logische gegevensverzamelingen wordt een aantal onderhoudsfuncties/schermen verwacht dat ligt tussen de 14 en 54. Op dit moment zijn er 27 ingevuld.
	> Het aantal invoerfuncties:	Meer...		Op basis van het aantal uitvoerfuncties en opvragingsfuncties worden ongeveer 15 invoerfuncties verwacht. Op dit moment zijn er 38 ingevuld.
	> Het aantal uitvoerfuncties:	Meer...		Op basis van het aantal invoerfuncties en opvragingsfuncties worden ongeveer 43 uitvoerfuncties verwacht. Op basis van het aantal opvragingsfuncties worden ongeveer 11 uitvoerfuncties verwacht. Op dit moment zijn er 16 ingevuld.
> Het aantal opvragingsfuncties:	Meer...		Op basis van het aantal uitvoerfuncties worden ongeveer 1 opvragingsfuncties verwacht. Op dit moment zijn er 1 ingevuld.	
> Het aantal select controls:	Meer...		Op basis van het aantal onderhoudsschermen, onderhoudsfuncties en raadpleegfuncties wordt verwacht dat het aantal select controls niet hoger is dan 18. Op dit moment zijn er 1 select controls ingevuld.	

In this application the generic functionality for the income suppletion regulation has been implemented.

APPENDIX C: eBS BTR 2006



Omvangsschatting FPA op basis van applicatiegegevens

Versie 1.3 LNV

applicatie: eBS GLB BTR2006

omvang: 1337 functiepunten

ingevuld door: J.H. de Vries

		onbekend		
> Gegevens	> Logische systeem <i>eigen</i> tabellen	Meer...	59	Logische tabellen met 50 of minder data elementen
			3	Logische tabellen met meer dan 50 data elementen
	> Logische systeem <i>vremde</i> tabellen	Meer...	onbekend	Logische tabellen met 50 of minder data elementen
			onbekend	Logische tabellen met meer dan 50 data elementen
> Invoerfuncties	> Onderhoudsfuncties (toevoegen, wijzigen, verwijderen) <i>tel toevoegen, wijzigen en verwijderen als aparte functies</i>	Meer...	60	Schermen en/of functies met maximaal 4 velden
			54	Schermen en/of functies met 5-15 velden
			6	Schermen en/of functies met meer dan 15 velden
	> Inkomende interfaces	Meer...	onbekend	Interfaces met maximaal 4 velden
			6	Interfaces met 5-15 velden
			onbekend	Interfaces met meer dan 15 velden
	> Batchverwerking / achtergrondprocessen <i>let op dubbelleningen met de inkomende en uitgaande interfaces</i>	Meer...	onbekend	Processen met maximaal 4 data elementen
			19	Processen met 5-15 data elementen
			onbekend	Processen met meer dan 15 data elementen
			onbekend	
> Uitvoerfuncties	> Raadpleegfuncties	Meer...	onbekend	Schermen en/of functies met maximaal 5 velden
			20	Schermen en/of functies met 6-20 velden
			18	Schermen en/of functies met meer dan 20 velden
	> Rapportages	Meer...	onbekend	Rapportages met maximaal 5 velden
			15	Rapportages met 6-20 velden
			17	Rapportages met meer dan 20 velden
	> Uitgaande interfaces	Meer...	onbekend	Interfaces met maximaal 5 velden
			7	Interfaces met 6-20 velden
			onbekend	Interfaces met meer dan 20 velden
	> Unieke select controls om velden in te vullen	Meer...	onbekend	Aantal unieke select controls binnen de applicatie
> Opvragingsfuncties	> Opvragingsfuncties op basis van een unieke sleutel	Meer...	onbekend	Functies met maximaal 5 velden
			onbekend	Functies met 6-20 velden
			onbekend	Functies met meer dan 20 velden
	> Helpfuncties	Meer...	N	Helpfunctie op applicatieniveau (J/N)
		N	Helpfunctie(s) op schermniveau (J/N)	
		J	Helpfunctie(s) op veldniveau / tooltips (J/N)	
> Vuistregels	> Deze vuistregels zijn bedoeld als hulpmiddel:			Hiermee kan nagegaan worden of de omvangsschatting compleet is.
	> Als de omvangsschatting niet aan de vuistregels voldoet:			Dit betekent niet zonder meer dat de omvangsschatting fout is. Wel is het verstandig om na te gaan waarom voor EBS GLB BTR deze vuistregel niet opgaat.
	> Omvang op basis van het aantal logische tabellen:	Meer...		Op basis van het aantal systeemeigen logische tabellen van en het aantal logische tabellen dat wordt gelezen wordt een totale omvang verwacht voor tussen de 1550 en de 2619 functiepunten. De schatting op dit moment is 1337 functiepunten.
	> Bijdrage van logische tabellen aan de omvang:	Meer...		De bijdrage van de logische tabellen aan de totale omvang is in de regel niet meer dan 40%. Die bijdrage is op dit moment 33%.
	> Het aantal logische tabellen:	Meer...		Op basis van het aantal onderhoudsfuncties en/of schermen wordt een aantal interne logische gegevensverzamelingen verwacht dat ligt tussen de 20 en 80. Op dit moment zijn er 62 ingevuld.
	> Het aantal onderhoudsfuncties:	Meer...		Op basis van het aantal interne logische gegevensverzamelingen wordt een aantal onderhoudsfuncties/schermen verwacht dat ligt tussen de 93 en 372. Op dit moment zijn er 120 ingevuld.
	> Het aantal invoerfuncties:	Meer...		Op basis van het aantal uitvoerfuncties en opvragingsfuncties worden ongeveer 70 invoerfuncties verwacht. Op dit moment zijn er 145 ingevuld.
	> Het aantal uitvoerfuncties:	Meer...		Op basis van het aantal invoerfuncties en opvragingsfuncties worden ongeveer 166 uitvoerfuncties verwacht. Op basis van het aantal opvragingsfuncties worden ongeveer 11 uitvoerfuncties verwacht. Op dit moment zijn er 79 ingevuld.
	> Het aantal opvragingsfuncties:	Meer...		Op basis van het aantal uitvoerfuncties worden ongeveer 7 opvragingsfuncties verwacht. Op dit moment zijn er 1 ingevuld.
	> Het aantal select controls:	Meer...		Op basis van het aantal onderhoudsschermen, onderhoudsfuncties en raadpleegfuncties wordt verwacht dat het aantal select controls niet hoger is dan 80. Op dit moment zijn er geen select controls ingevuld.

In this application the variable functionality for the income suppletion regulation for the fiscal year 2006 has been implemented.

APPENDIX D: eBS BTR 2007



Omvangsschatting FPA op basis van applicatiegegevens

Versie 1.3 LNV

applicatie: eBS GLB BTR2007

omvang: 1611 functiepunten

ingevuld door: J.H. de Vries

		onbekend		
> Gegevens	> Logische systeem <i>eigen</i> tabellen	Meer...	<input type="text" value="65"/>	Logische tabellen met 50 of minder data elementen
			<input type="text" value="5"/>	Logische tabellen met meer dan 50 data elementen
	> Logische systeem <i>vremde</i> tabellen	Meer...	<input type="text" value=""/>	Logische tabellen met 50 of minder data elementen
			<input type="text" value=""/>	Logische tabellen met meer dan 50 data elementen
> Invoerfuncties	> Onderhoudsfuncties (toevoegen, wijzigen, verwijderen) <i>tel toevoegen, wijzigen en verwijderen als aparte functies</i>	Meer...	<input type="text" value="66"/>	Schermen en/of functies met maximaal 4 velden
			<input type="text" value="54"/>	Schermen en/of functies met 5-15 velden
			<input type="text" value="24"/>	Schermen en/of functies met meer dan 15 velden
	> Inkomende interfaces	Meer...	<input type="text" value="8"/>	Interfaces met maximaal 4 velden
			<input type="text" value=""/>	Interfaces met 5-15 velden
			<input type="text" value=""/>	Interfaces met meer dan 15 velden
	> Batchverwerking / achtergrondprocessen <i>let op dubbelleningen met de inkomende en uitgaande interfaces</i>	Meer...	<input type="text" value="36"/>	Processen met maximaal 4 data elementen
			<input type="text" value=""/>	Processen met 5-15 data elementen
			<input type="text" value=""/>	Processen met meer dan 15 data elementen
	> Uitvoerfuncties	> Raadpleegfuncties	Meer...	<input type="text" value="22"/>
			<input type="text" value="18"/>	Schermen en/of functies met 6-20 velden
			<input type="text" value="8"/>	Schermen en/of functies met meer dan 20 velden
> Rapportages		Meer...	<input type="text" value="13"/>	Rapportages met maximaal 5 velden
			<input type="text" value="21"/>	Rapportages met 6-20 velden
			<input type="text" value="2"/>	Rapportages met meer dan 20 velden
> Uitgaande interfaces		Meer...	<input type="text" value=""/>	Interfaces met maximaal 5 velden
			<input type="text" value=""/>	Interfaces met 6-20 velden
			<input type="text" value=""/>	Interfaces met meer dan 20 velden
> Unieke select controls om velden in te vullen		Meer...	<input type="text" value=""/>	Aantal unieke select controls binnen de applicatie
> Opvragingsfuncties	> Opvragingsfuncties op basis van een unieke sleutel	Meer...	<input type="text" value=""/>	Functies met maximaal 5 velden
			<input type="text" value=""/>	Functies met 6-20 velden
			<input type="text" value=""/>	Functies met meer dan 20 velden
	> Helpfuncties	Meer...	<input type="text" value="N"/>	Helpfunctie op applicatieniveau (J/N)
		<input type="text" value="N"/>	Helpfunctie(s) op schermniveau (J/N)	
		<input type="text" value="J"/>	Helpfunctie(s) op veldniveau / tooltips (J/N)	
> Vuistregels	> Deze vuistregels zijn bedoeld als hulpmiddel:			Hiermee kan nagegaan worden of de omvangsschatting compleet is.
	> Als de omvangsschatting niet aan de vuistregels voldoet:			Dit betekent niet zonder meer dat de omvangsschatting fout is. Wel is het verstandig om na te gaan waarom voor EBS GLB BTR deze vuistregel niet opgaat.
	> Omvang op basis van het aantal logische tabellen:	Meer...		Op basis van het aantal systeemeigen logische tabellen van en het aantal logische tabellen dat wordt gelezen wordt een totale omvang verwacht voor tussen de 1750 en de 2965 functiepunten. De schatting op dit moment is 1611 functiepunten.
	> Bijdrage van logische tabellen aan de omvang:	Meer...		De bijdrage van de logische tabellen aan de totale omvang is in de regel niet meer dan 40%. Die bijdrage is op dit moment 31%.
	> Het aantal logische tabellen:	Meer...		Op basis van het aantal onderhoudsfuncties en/of schermen wordt een aantal interne logische gegevensverzamelingen verwacht dat ligt tussen de 24 en 96. Op dit moment zijn er 70 ingevuld.
	> Het aantal onderhoudsfuncties:	Meer...		Op basis van het aantal interne logische gegevensverzamelingen wordt een aantal onderhoudsfuncties/schermen verwacht dat ligt tussen de 105 en 420. Op dit moment zijn er 144 ingevuld.
	> Het aantal invoerfuncties:	Meer...		Op basis van het aantal uitvoerfuncties en opvragingsfuncties worden ongeveer 74 invoerfuncties verwacht. Op dit moment zijn er 188 ingevuld.
	> Het aantal uitvoerfuncties:	Meer...		Op basis van het aantal invoerfuncties en opvragingsfuncties worden ongeveer 215 uitvoerfuncties verwacht. Op basis van het aantal opvragingsfuncties worden ongeveer 11 uitvoerfuncties verwacht. Op dit moment zijn er 84 ingevuld.
	> Het aantal opvragingsfuncties:	Meer...		Op basis van het aantal uitvoerfuncties worden ongeveer 8 opvragingsfuncties verwacht. Op dit moment zijn er 1 ingevuld.
	> Het aantal select controls:	Meer...		Op basis van het aantal onderhoudsschermen, onderhoudsfuncties en raadpleegfuncties wordt verwacht dat het aantal select controls niet hoger is dan 96. Op dit moment zijn er geen select controls ingevuld.

In this application the variable functionality for the income suppletion regulation for the fiscal year 2007 has been implemented.

APPENDIX E: eBS BTR 2008



Omvangsschatting FPA op basis van applicatiegegevens

Versie 1.3 LNV

applicatie: eBS GLB BTR2008

omvang: 1554 functiepunten

ingevuld door: J.H. de Vries

		onbekend		
> Gegevens	> Logische systeem <i>eigen</i> tabellen	Meer...	65	Logische tabellen met 50 of minder data elementen
			5	Logische tabellen met meer dan 50 data elementen
	> Logische systeem <i>vremde</i> tabellen	Meer...	onbekend	Logische tabellen met 50 of minder data elementen
				Logische tabellen met meer dan 50 data elementen
> Invoerfuncties	> Onderhoudsfuncties (toevoegen, wijzigen, verwijderen) <i>tel toevoegen, wijzigen en verwijderen als aparte functies</i>	Meer...	66	Schermen en/of functies met maximaal 4 velden
			51	Schermen en/of functies met 5-15 velden
			24	Schermen en/of functies met meer dan 15 velden
	> Inkomende interfaces	Meer...	7	Interfaces met maximaal 4 velden
				Interfaces met 5-15 velden
				Interfaces met meer dan 15 velden
	> Batchverwerking / achtergrondprocessen <i>let op dubbelleningen met de inkomende en uitgaande interfaces</i>	Meer...	37	Processen met maximaal 4 data elementen
				Processen met 5-15 data elementen
				Processen met meer dan 15 data elementen
	> Uitvoerfuncties	> Raadpleegfuncties	Meer...	22
			17	Schermen en/of functies met 6-20 velden
			8	Schermen en/of functies met meer dan 20 velden
> Rapportages		Meer...	12	Rapportages met maximaal 5 velden
			14	Rapportages met 6-20 velden
			2	Rapportages met meer dan 20 velden
> Uitgaande interfaces		Meer...		Interfaces met maximaal 5 velden
				Interfaces met 6-20 velden
				Interfaces met meer dan 20 velden
> Unieke select controls om velden in te vullen		Meer...		Aantal unieke select controls binnen de applicatie
> Opvragingsfuncties	> Opvragingsfuncties op basis van een unieke sleutel	Meer...		Functies met maximaal 5 velden
				Functies met 6-20 velden
				Functies met meer dan 20 velden
	> Helpfuncties	Meer...	N	Helpfunctie op applicatieniveau (J/N)
		N	Helpfunctie(s) op schermniveau (J/N)	
		J	Helpfunctie(s) op veldniveau / tooltips (J/N)	
> Vuistregels	> Deze vuistregels zijn bedoeld als hulpmiddel:			Hiermee kan nagegaan worden of de omvangsschatting compleet is.
	> Als de omvangsschatting niet aan de vuistregels voldoet:			Dit betekent niet zonder meer dat de omvangsschatting fout is. Wel is het verstandig om na te gaan waarom voor EBS GLB BTR deze vuistregel niet opgaat.
	> Omvang op basis van het aantal logische tabellen:	Meer...		Op basis van het aantal systeemeigen logische tabellen van en het aantal logische tabellen dat wordt gelezen wordt een totale omvang verwacht voor tussen de 1750 en de 2965 functiepunten. De schatting op dit moment is 1554 functiepunten.
	> Bijdrage van logische tabellen aan de omvang:	Meer...		De bijdrage van de logische tabellen aan de totale omvang is in de regel niet meer dan 40%. Die bijdrage is op dit moment 32%.
	> Het aantal logische tabellen:	Meer...		Op basis van het aantal onderhoudsfuncties en/of schermen wordt een aantal interne logische gegevensverzamelingen verwacht dat ligt tussen de 24 en 94. Op dit moment zijn er 70 ingevuld.
	> Het aantal onderhoudsfuncties:	Meer...		Op basis van het aantal interne logische gegevensverzamelingen wordt een aantal onderhoudsfuncties/schermen verwacht dat ligt tussen de 105 en 420. Op dit moment zijn er 141 ingevuld.
	> Het aantal invoerfuncties:	Meer...		Op basis van het aantal uitvoerfuncties en opvragingsfuncties worden ongeveer 66 invoerfuncties verwacht. Op dit moment zijn er 185 ingevuld.
	> Het aantal uitvoerfuncties:	Meer...		Op basis van het aantal invoerfuncties en opvragingsfuncties worden ongeveer 212 uitvoerfuncties verwacht. Op basis van het aantal opvragingsfuncties worden ongeveer 11 uitvoerfuncties verwacht. Op dit moment zijn er 75 ingevuld.
	> Het aantal opvragingsfuncties:	Meer...		Op basis van het aantal uitvoerfuncties worden ongeveer 7 opvragingsfuncties verwacht. Op dit moment zijn er 1 ingevuld.
	> Het aantal select controls:	Meer...		Op basis van het aantal onderhoudsschermen, onderhoudsfuncties en raadpleegfuncties wordt verwacht dat het aantal select controls niet hoger is dan 94. Op dit moment zijn er geen select controls ingevuld.

In this application the variable functionality for the income suppletion regulation for the fiscal year 2008 has been implemented.

Implementing a Metrics Program

MOUSE will help you

Ton Dekkers
tdekkers@galorath.com
Galorath International Ltd

Abstract

Just like an information system, a method, a technique, a tool or an approach is supporting the achievement of an objective. Following this line of thought, implementing a method, a technique and so on, should in many ways be comparable to the development of an information system. In this paper the implementation approach MOUSE is applied to implement a metrics program. The people, the process and the product get all the attention that is needed to make it success. It's also very helpful to asses the organisation's "readiness" to adopt the system.

1 A Metrics Program

It doesn't matter where a metrics or measurement program is initiated; implementing a metrics program can be seen as 'just another' staged IT project. The IT project brings together software, hardware, infrastructure, organisation and people. An IT project is structured with stages for development, transition to support, run & maintain and implementation. Why not apply the same structure to implementing a metrics program. All items from a 'real' IT projects have to be addressed too. However a metrics program is not equal to an information system, this requires different activities. Also the stages are somewhat different.

In figure 1, the baseline for the project lifecycle of the metric program implementation is given and in the following paragraph each stage is explained in detail.

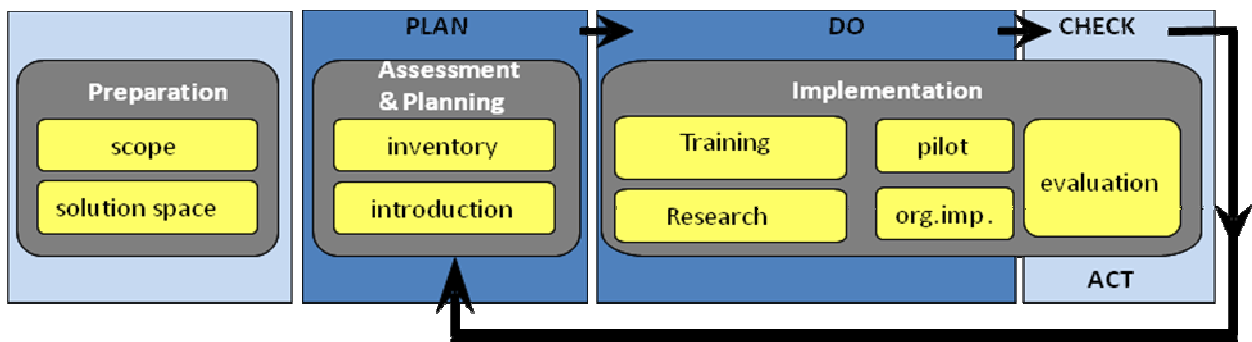


Figure 1

1.1 Preparation

Before the decision to implement a metrics program is made goals need to be defined clearly that should be served by the program. In the preparation phase, the scope and the boundaries of solution space are set. Most of the time the estimating process [1] of the software development and/or performance benchmarking is driving the metrics program

to be implemented. A good framework to decide which goals and the metrics are needed for the defined goals is the Goal-Question-(Indicator-)Metric (GQ-I-M) Method [2]. These goals are the basis for the organisation specific elements in the implementation of a metrics program.

1.2 Assessment & Planning

As showed in figure 1 this phase has two main activities: inventory and introduction. This phase can be compared with the requirements definition phase: the requirement has to be gathered and agreed with the stakeholders.

During this stage the current working methods and procedures are assessed together with all aspects that might have a relation with the metrics program to be implemented, such as:

- already implemented measurements: (functional) size, effort, defects, duration;
- software development methodology: development process(es), stages, milestones, activities, deliverables, guidelines and standards);
- software development environments (platform);
- measurement process(es) and related data collection procedures(s);
- common project organisation and related procedures;
- generic /organisation specific characteristics and the project specific ones;
- effort and work allocation breakdown;
- risk management procedures.

After the analysis of the current situation, the results have to map on the objectives of the measurement program. The assessment is executed using the agreed upon solution space and the implementation approach MOUSE that is described later in this document. This 'gap-analysis' is used to determine which activities and which stakeholders will be affected by the metrics program. Those stakeholders have to be informed or invited to participate and when necessary trained to work with the metrics program.

At the end of the Assessment & Planning stage there is a documented consensus about what the metrics program is expected to monitor, who will be involved and in what way (implementation plan vs. project plan). This implementation plan describes the transition from the current state to the wanted situation. In addition to the drawing of the current state (including findings and recommendations) and the wanted situation, the plan includes:

- the necessary changes
- per change the transformation, the steps to make the change
- the identified training needs
- an awareness/communication program
- the agreed upon base data collection set
- possible pilot and research projects
- required staff and roles in the implementation phase

When the approach is applied for assessment of the "readiness" the maturity or the presence of the assessed items is analysed in a similar way. Most common struggle from the "readiness" perspective is the addressing of the organisational structure, responsibilities and (management) support. Most of the time the methodical and operation setting is not the issue.

1.3 Implementation

1.3.1 Training

Employees in the organisation who are affected by the metrics program will have to be trained to use the metrics in a correct manner. Depending on the role this training can be adapted from an introduction presentation to a multiple day training course. For the introduction of a metrics program typically five categories of employees are identified:

- **Management:**
The management must have and convey commitment to the implementation of a metrics program. The management needs to be informed about the possibilities and impossibilities of the metrics program. They also must be aware of the possible consequences a metrics program might have on the organisation. It is well known that employees feel threatened by the introduction of a metrics program and in some cases react quite hostile. Such feelings can usually be prevented by open and correct communication of the management about the true purposes of the metrics program and the expected benefits.
- **Metrics analysts:**
The specialist who are responsible for analysing and reporting the metrics data. They are also responsible for measuring and improving the quality of the metrics program itself. They might have been trained in the preparation stage, otherwise they have to be trained and the basic and advanced topics. Depending on the implementation, additional tool training is required.
- **Metrics collectors:**
The employees that are actively involved in collecting or calculating the metrics have to know all the details and consequences of the metrics, to assure correct and consistent data. If the metrics that are used in the metrics program come from activities that are already common practice, the training may only take a couple of hours. If the metrics are not common practice or involve specialist training, for instance if functional sizes have to be derived from design documents, the training may take a substantial amount of time. In the last case this involves serious planning, especially in matrix organisations: It will not only consume time of the employee involved, but it will also affect his or her other activities. Depending on the implementation, additional tool training might be required.
- **Software developers:**
Usually most of the employees that are involved in the software development will be affected, directly or indirectly, by the metrics program, because they 'produce' the information the metrics program uses. They need to have understanding of the metrics and the corresponding vocabulary. For them the emphasis of the training needs to be on awareness, understanding the use and importance of a metrics program for the organisation, because they usually not expect any benefit from the program for their personal activities. However there are also benefits for them. In addition they may need to change some of their products to make measurement possible or consistent.
- **End-users or clients:**
Although a metrics program is set up primarily for the use of the implementing organisation, end-users or clients can also benefit from it. In particular sizing metrics are useful in the communication between the client and the supplier: how much functionality will the client get for the investment. Whether this audience will be part of the training stage for a metrics program depends on the openness of the implementing organisation: are they willing to share information about the performance of their projects?

At the end of the training stage everyone who will be affected directly or indirectly by the metrics program has sufficient knowledge about this program. It may seem obvious, but it is essential the training stage is finished before (but preferably not too long before) the actual implementation of the metrics program starts.

1.3.2 Research

In this stage the metrics to be implemented are mapped on the activities within the organisation that will have to provide the metrics data. The exact process of collecting the metrics data is determined and described in such detail that at the start of the implementation it is unambiguous which metrics data will be collected and how.

In this stage it is useful to determine what the range of the metrics data might be. An approach that is very helpful is Planguage [3]. The word Planguage is a combination of Planning and Language. The purpose of Planguage is to describe a concept (e.g. a metric), related to stakeholders, scope and qualifiers. In case of a metric the qualifiers are the type and type of measurement and the scale, the base measurement and the level to achieve. A wrong perception of the possible result of metrics data can kill a metrics program at the start. It is also important to establish at least an idea of the expected bandwidth of the metrics data beforehand to know what deviations can be considered acceptable and what deviations call for immediate action.

At the end of the research stage all procedures to collect metrics data are described, target values for each metric are known and allowable bandwidths are established for each metric in the metrics program.

1.3.3 Organisational Impact

Until now the metrics program has had little impact on the organisation, because only a limited number of employees have been involved in the pilot. The organisational implementation of a metrics program will have an impact on the organisation because the organisation has formulated goals which the metrics program will monitor. These goals may not have been communicated before or may not have been explicitly made visible. Metrics will have to be collected at specified moments or about specified products or processes. This could mean a change in procedures. For the employees involved this is a change process, which can trigger resisting or even quite hostile reactions. Over 10 years of experience show that the most suitable organisational structure for a metrics program is to concentrate expertise, knowledge and responsibilities in an independent body, a separate expertise group or incorporated in the Project Management Office or Audit group. An independent body has many advantages over other organisational structures. For example, when activities are assigned to individuals in projects, many additional measures have to be taken to control the quality of the measurements, the continuation of the measurement activities and the retention of expertise about the metrics program. When responsibilities for (parts of) the metrics program are assigned to projects, additional enforcing measures have to be taken to guarantee adequate attention from the project to metrics program assignments over other project activities. Installing an independent body to oversee and/or carry out the metrics program is essential for achieving the goals the metrics program was set up for. This independent body can be either a person or a group within or outside the organisation. How this body should operate is laid down in the MOUSE approach, which will be described in detail later on.

At the end of the implementation stage the metrics program is fully operational and available throughout the organisation.

1.3.4 Pilot

Unless the organisation is very confident that a metrics program will work properly from the start, it is best to start the implementation with a pilot. In a pilot metrics are collected from a limited number of activities and with a limited number of people involved. In a pilot all procedures are checked, experience is built up with these procedures and the first metrics data are recorded. In this way the assumptions about the metrics values and bandwidths from the research stage can be validated.

After the completion of the pilot all procedures and assumptions are evaluated and modified if necessary. When the modifications are substantial it may be necessary to test them in another pilot before the final organisational implementation of the metrics program can start.

The pilot and the evaluation of the pilot can be considered the technical implementation of the metrics program. After completion of this stage the metrics program is technically ready to be implemented.

1.4 Operation

This stage is actually not a stage anymore. The metrics program has been implemented and is now a part of the day-by-day operations. The metrics program is carried out conform the way it is defined and is producing information that helps the organisation to keep track of the way it is moving towards their goals.

A mature metrics program gives continuous insight in the effectiveness of current working procedures to reach the organisational goals. If the effectiveness is lower than desirable adjustments to the procedures should be made. The metrics program itself can then be used to track if the adjustments result in the expected improvement. If working procedures change it is also possible that adjustments have to be made to the metrics program.

Organisational goals tend to change over time. A mature metrics program contains regular checks to validate if it is still delivering useful metrics in relation to the organisational goals. All these aspects are covered in the MOUSE approach.

2 MOUSE

Implementing a metrics program is more than just training people and defining the use of metrics. All the lessons learned from implementations in (global operation) industry, banking and government were the basis for MOUSE, an approach to help to set-up the right implementation and to create the environment the method is fit for purpose. MOUSE describes all activities and services that need to be addressed to get a metrics program set up and to keep running.

MOUSE is a clustering of all the activities and services into groups of key issues, described in the table below:

Table 1: Key issues of MOUSE

Market View	Operation	Utilisation	Service	Exploitation
Communication	Application	Training	Helpdesk	Registration
Evaluation	Review	Procedures	Guidelines	Control
Improvement	Analysis	Organisation	Information	(Analysis)
Investigation	Advice		Promotion	

In the following paragraphs the five groups of MOUSE will be explained.

2.1 Market view

Communication in the MOUSE approach is an exchange of information about the metrics program both internally (the own organisation) and externally (metrics organisations, universities, interest groups, ...). Internal communication is essential to keep up the awareness about the goals for which the metrics program is set up. For example: Company publications (newsletters) and an intranet website are very useful to share information.

Communication with outside organisations is important to keep informed about the latest developments. Usually an important metric in a metrics program in an IT-environment is the functional size of software The International Function Point User Group (IFPUG [4]) and local organisations like Netherlands Software Measurement Association (NESMA [5]) and United Kingdom Software Measurement Association (Mark II [6]) the platforms for Function Point Analysis. Cosmicon is a platform for COSMIC Function Points [7]. In principle all SMA can help with the use of Functional Size Measurement (FSM) Methods. The requirements of this knowledge transfer depend on the organisational situation and demands. This interaction can be outsourced to a partner / supplier in case the partner has already contacts in these international networks and offers the possibilities to share. Collaboration with universities and specific interest groups (SIG) from other professional organisations (e.g. Metrics SIG of Project Management Institute – PMI [8]) is useful too.

If the independent body is located within the client's organisation, a direct and open communication is possible with stakeholders of the metrics program to evaluate whether the metrics program is still supporting the goals initially driving the program. When the independent body is positioned outside the client's organisation more formal ways to exchange information about the metrics program are required. Regular evaluations or some other form of assessment of the measurement process works well for an open communication about the metrics program.

The findings that the evaluations provide are direct input for continuous improvement of the metrics program. Depending upon the type of finding (operational, conceptual or managerial) further investigation may be required before a finding can be migrated to measurement process improvement.

Investigation can be both theoretical and empirical. Theoretical investigation consists of studying literature, visiting seminars and conferences or following workshops. Empirical investigation consists of evaluating selected tools for measurement and the analysis of experience data. Usually these two ways of investigation are used in combination.

2.2 Operation

Application includes all activities that are directly related to the application of the metrics program. This includes activities like executing measurements (for example functional size measurements, tallying hours spent and identifying project variables). Within MOUSE the organisation can choose to assign the functional sizing part of the operation either to the independent body or to members of the projects in the scope of the metrics program.

The best way to guarantee quality of the measurement data is to incorporate review steps into the metrics program. The purpose of reviewing is threefold:

- ensure correct use of the metrics (rules and concepts);
- keep track of applicability of the metrics program;
- to stay informed about developments in the organisation that might influence the metrics program.

During the research stage all procedures to collect metrics data are described for each metric in the metrics program. These procedures are described in a way that they support the organisational goals for which the metrics program was set up. Some metrics data can also be used to support project purposes. The expertise group can advise on and support the usage of the metrics for these purposes. For example an aspect of the metrics program can be the measurement of the scope creep of projects during their lifetime. Functional size is measured in various stages of the project to keep track of the size as the project is progressing. These functional size measures can also be used for checking the budget as a second opinion to the budget based on work breakdown structure for example. The independent body can give advice about the translation of the creep ratio in the functional size to a possible increase of the budget.

During the research stage target values and allowable bandwidth are established for each metric in the metrics program. The independent body will have to analyse if these target values were realistic at the beginning of the metrics program and if they are still realistic at present. One of the organisational goals might be to get improving values for certain metrics. In that case, the target values for those metrics and/or their allowed bandwidth will change over time. E.g. the effort estimate bandwidth should improve to 15% from 20%.

2.3 Utilisation

Next to the basic training at the start of a metrics program it is necessary to maintain knowledge about the metrics program at an appropriate level. The personnel of the independent body should have refreshment training on a regular basis, referring to new developments (rules, regulations) in the area of the applied methods. The independent body can then decide whether it is necessary to train or inform other people involved in the metrics program about these developments. In the case that the independent body is outsourced, the supplier can be made responsible for keeping the knowledge up-to-date.

To guarantee the correct use of a method, procedures related to measurement activities of the metrics program are necessary. They are usually initiated and established in the research stage of the implementation. Not only the measurement activities themselves need to be described, but also facilitating processes like:

- project management
- change management control
- project registration
- (project) evaluation
- performance analysis

After the initial definitions in the research stage the independent body should monitor that that all the relevant definitions are kept up-to-date.

As stated earlier the independent body can reside within or outside the organisation where the metrics program is carried out. The decision about this organisational aspect is usually combined with the number of people involved in the metrics program. If the metrics program is small enough to be carried out by one person in part-time the tasks of the independent body are usually assigned to an external supplier. If the metrics program is large enough to engage one or more persons full-time the tasks of the independent body are usually assigned to employees of the organisation. Depending on the type of organisation this might not always be the best solution for a metrics program. When the goals the organisation wants to achieve are of such a nature that it involves sensitive information, calling in external consultants might be a bad option, no matter how small the metrics program might be. If employees have to be trained to carry out the tasks of the independent body, they might perceive that as narrowing their options for a career within the organisation. In that case it might be wise to assign these tasks to an external party specialising in these kinds of assignments, no matter how large the metrics program is. Outsourcing these assignments to an external party has another advantage: it simplifies the processes within the client's organisation. Another advantage of outsourcing the independent body could be political: to have a really independent body to do the measurement or at least a counter measurement.

2.4 Service

To support the metrics program a helpdesk (e.g. focal point within the expertise group) needs to be instated. All questions regarding the metrics program should be directed to this helpdesk. The helpdesk should be able to answer questions with limited impact immediately and should be able to find the answers to more difficult questions within a reasonable timeframe. It is essential that the helpdesk reacts adequately to all kinds of requests related to the metrics program. In most cases the employees that staff the independent body constitute the helpdesk.

Decisions made regarding the applicability of a specific metric in the metrics program need to be recorded in order to incorporate such decisions into the 'corporate memory' and to be able to verify the validity of these decisions at a later date. Usually such decisions are documented in organisation specific guidelines for the use of that specific metric.

The success of a metrics program depends on the quality of the collected data. It is important that those who supply the data are prepared to provide this data. The best way to stimulate this is to give them information about the data in the form of analyses. This should provide answers to frequently asked questions, such-as: "What is the current project delivery rate for this specific platform?", "What is the reliability of the estimations?", "What is the effect of team size?". Usually the historical data can answer the questions related to internal performance per functional size unit. When this isn't

available (yet), historical data of third parties can be used, e.g. the repositories of the International Software Benchmarking Standards Group – ISBSG [9].

Promotion is the result of a proactive attitude of the independent body. The independent body should market the benefits of the metrics program and should ‘sell’ the services it can provide based on the collected metrics. Promotion is necessary for the continuation and extension of the metrics program.

2.5 Exploitation

The registration part of a metrics program consists of two components: the measurement results and the analysis results. In a metrics program in an IT-environment all metrics will be filed digitally without discussion. Here a proper registration usually deals with keeping the necessary data available and accessible for future analysis. For most metrics programs it is desirable that the analysis data is stored in some form of an experience database. In this way the results of the analyses can be used to inform or advise people in the organisation.

Control procedures are required to keep procedures, guidelines and the like up-to-date. If they do no longer serve the metrics program or the goals the organisation wants to achieve, they should be adjusted or discarded. Special attention needs to be given to the procedures for storing metrics data. That data should be available for as long as is necessary for the metrics program. This might be longer than the life of individual projects, so it is usually advisable to store data in a place that is independent of the projects the data comes from.

3 Tools

Although a tool is not required to implement a metrics program, it definitely supports the operation of the process and the acceptance. When the introduction of tool(s) is part of the implementation, some of the items in MOUSE need additional attention.

In addition the tool(s) may require additional data collection to maximise the benefits of the tooling. This will influence the scope and solution space.

3.1 Type of tools

The activities described in this document can be supported by tooling in a number of areas:

- (functional) sizing
- data collection
- historical data storage
- estimation & control
- (performance) analysis
- benchmarking

In practise, the most generic solutions are found in series of Excel sheets, sometimes linked with Access or SQL databases. This might be cheaper and flexible, however from a quality, maintenance, governance and consistency perspective this is not the most appropriate option.

In the sizing area, (effective) source lines of code (SLOC) is quite often the easiest choice for a size measure, in that case a tool for counting the lines of code is almost unavoidable. When selecting a tool, the definition of the SLOC counted should be according clear definitions and preferably according well respected public standards (e.g. the standard from the Software Engineering Institute – SEI).

My preference for sizing are the ISO certified functional sizing methods e.g. Function Point Analysis or COSMIC Functional Sizing Method. In the market space various tools are available for registering detailed counting results. There are good tools in the commercial domain as well as in the freeware domain.

Selecting the appropriate sizing approach is very important for support and acceptance. Especially in organisations where most development is linked with real time applications, control systems, embedded software, contemporary environments or applications with a strong data interdependency, COSMIC might be an interesting option.

The same applies for tools in the other areas. It will not be surprising that I would like to inform you that the Galorath SEER suite [10] is a professional solution for integrating the areas historical data storage (SEER HD – Historical Data), Estimating (SEER for Software, SEER IT – Information Technology), Control (SEER PPMC – Parametric Project Monitoring & Control) and benchmarking (SEER Metrics that also supports the ISBSG repository). SEER by comparison supports sizing based on analogy, whatever sizing unit is chosen when of course historical data is available.

3.2 Implementation of tools

When tools are selected to support the measurement program and to utilise the collected data, the groups and items in MOUSE that need special attention are:

- Market View (improvement, investigation)
- Operation (application, analysis)
- Utilisation (training, procedures, organisation)
- Service (helpdesk, guideline)
- Exploitation (registration, control)

Initially the basis requirements for the use of the tools has to be set (investigation) in line with the purpose and goals of the metrics program. Most of the tools offer more functionality than initially required that might become useful when the metric program get more mature (improvement).

A tool has to be integrated in the measurement program application. Part of this is that the reporting and analysis provided by the tool(s) needs to be tailored or translated to the definition and needs of the organisation.

It's obvious that a tool requires additional training. Especially when referring to the metrics program, the tools are mostly expert tools. (Advanced) Training is almost mandatory, at least for the expertise group that needs to support the organisation. It has to be embedded in the procedures and responsibilities have to get in line with other organisational structures.

A focal point for setting the guidelines for the usage of the tool(s) and support (e.g. helpdesk) has to be organised.

And last but not least, practical things like back-up, storage, access and security must be put in place (registration, control).

4 Conclusions

When starting the implementation from a processing perspective, it's not only easier to manage but also easier to address the items that are relevant to make the implementation a success. Already early in the "project" it's clear what's needs to be done and what is needed to make the environment (process, product and most of all people) ready to accept and adopt the new program.

5 References

- [1] Galorath, Evens, "Software Sizing, Estimation and Risk Management", 2006, Auerbach Publications, ISBN 0-8493-3593-0
- [2] Van Solingen, Berghout, "The Goal/Question/Metric Method", 1999. Mc Graw Hill, ISBN 0-07-709553-7
- [3] Gilb, "Competitive Engineering", 2005, Elsevier, ISBN 0-7506-6507-6
- [4] International Function Point User Group, "Function Point Counting Practices Manual version 4.2.1", 2005, IFPUG, www.ifpug.org
- [5] Netherlands Software Metrics Association, "Definitions and counting guidelines for the application of function point analysis, NESMA Functional Size Measurement method conform ISO/IEC 24570, version 2.1.", 2004, NESMA, ISBN 978-90-76258-19-5, www.nesma.org
- [6] United Kingdom Software Metrics Association, "MK II Function Point Analysis Counting Practices Manual, Version 1.3.1", 1998, UKSMA, www.uksma.co.uk
- [7] Common Software Measurement International Consortium, "COSMIC Functional Size Measurement Method, version 3.0", 2007, COSMIC, www.gelog.etsmtl.ca/cosmic-ffp
- [8] Project Management Institute, www.pmi.org
- [9] International Software Benchmarking Standards Group, www.isbsg.org
- [10] Galorath, "SEER suite", www.galorath.com

ⁱ "MIERUKA": Japanese Copy term of "Visualize".