Size Estimation for Management Information Systems Based on Early Metrics:
An Automatic Metric Tool Based in Formal Specifications

Pedro Salvetto
Universidad ORT Uruguay
salvetto@athenea.ort.edu.uy
(05982) 902 15 05 ext 123

Juan C. Nogueira
Universidad ORT Uruguay
nogueira@athenea.ort.edu.uy
(05982) 902 15 05 ext 210

Cuareim 1451
Montevideo, Uruguay
Topic: Formal Specifications Based Metrics
Size Estimation for Management Information Systems Based on Early Metrics: An Automatic Metric Tool Based in Formal Specifications

Pedro Salvetto
Universidad ORT Uruguay
salvetto@athenea.or.edu.uy

Juan C. Nogueira
Universidad ORT Uruguay
nogueira@athenea.or.edu.uy

ABSTRACT
We present a tool to extend previous research [1], [2] with the goal of estimating risk and development time for management information systems (MIS). The previous work focused on real-time systems and related in simulations for model validation. Our research aims to generalize the previous results for MIS and lighter development environments. Our goal is a probabilistic estimation model based on early metrics, which characterize the evolutionary software development processes, such as team efficiency, requirement volatility, and complexity.

This paper discusses the metrics used to derive the input parameters of the model. These metrics are obtained automatically with our tool from the system specifications. At this stage of the research we used a small set of projects developed with a code generator based on formal specifications, specially suited for MIS developments.

KEYWORDS
Software economics, software metrics, MIS, estimation models, formal specification

1 INTRODUCTION
Despite the advances in hardware, software development has not experienced a radical improvement, and it is far behind the needs of the information age society. After many years of research, delays, cost overruns, and project cancellations after huge investments, are often the rules more than the exception. The software crisis seems that has not finished yet.

Many different estimation models have been introduced. Most of them are successful in the hands of their creators or highly trained experts. However, it seems that the learning curves are too difficult, adding intricacy in leveraging of these models over non-expert users. Consequently, the crisis continued for the majority of software development organizations [5]. Another drawback of these models is that they are applicable after the requirements are almost complete. Estimations based under the hypothesis of frozen requirements are too restrictive and more than optimistic. Hence, even with the existence of well-known models such as COCOMO [3], COCOMO II [4], SLIM [5] and others, optimistic planning is very common [6].

Despite the estimation problem, there is a big difficulty in assessing risk. The available risk assessment techniques for software projects are based on heuristics, check lists, recommended practices, data bases with historical data, risk taxonomies, and a few metrics [6], [7], [8], [9], [10], [11]. The main issue with the current state of the art is that these methods rely on human expertise. This constitutes a weakness because different people could have different perceptions of risk from the same scenario, arriving to very different conclusions [1]. Moreover, experienced project managers are a scarce resource, and the successes in previous projects do not guarantee present or future success.

Some of the factors that contribute to the immaturity of software engineering are: a) it is a young discipline; b) there is a lot of uncertainty about the product, the cost, and the required effort; c) the final product is intangible and its value depends not only in its correctness but also in the timing, quality, maintainability, and extensibility; d) software development requires intensive intellectual work and human interaction (humans are emotional actors instead of rational automata); e) planning tools such as PERT and CPM do not consider the iterations that characterize development processes, and neglect the intense interaction between roles and reworks; f) despite SEI’s CMM, repeatable software processes are rare g) software projects could be replicable but not repeatable h) technology changes so fast and radically that it is difficult to take advantage of past experiences.

Project managers must choose between taking early decisions under high uncertainty, or postponing the decision, trading time and cost by uncertainty. Part of the problem was solved with an early formal estimation method introduced by Nogueira [1]. This method - applicable only to real time systems- is based on formal specifications and code generated from them. However, the metrics used (efficiency, requirement volatility, and complexity) can be applied to other environments for risk estimation. Our research aims to extend this previous work in two directions. First, we extend the method for MIS. Second, we use real projects instead of simulation for validation and tuning purposes.
In order to maintain the hypothesis of the previous research, we used a set of projects developed with a code generation tool based on formal specifications. To prevent the introduction of confounding effects, we used metrics collected automatically from the systems specifications.

2 THEORETICAL BACKGROUND

2.1 NOGUEIRA’S MODEL

The model uses the following parameters: a) organization’s efficiency (EF); b) requirement volatility (RV); and c) complexity (CX). The metrics can be derived from the formal specification, and constitute the input parameters for the model, which returns the probability of finishing the development at a given time, or inversely the development time required for a given probability of success. The model uses a Weibull distribution function. Details of the model can be found in [1], [2]. The model was calibrated using simulation and real time systems developed with Computer Aided Prototyping System (CAPS)

1. CAPS is a code generator that uses formal specifications in PSDL [12]. The specification is constructed using a graphical interface with the elements of PSDL (operators, data streams, and abstract data types).

To measure complexity (CX), Nogueira’s model uses a metric called Large Granularity Complexity (LGC). LGC expresses the relational complexity of the system and is computed as the sum of operators (O), data streams (D), and abstract data types (T):

\[ LGC = O + D + T \]

The efficiency (EF) is obtained from the rate between direct working time and idle time.

The requirement volatility is calculated from the requirement birth rate and death rate. A requirement change is computed as a death and the birth of a new one.

\[ RV = BR\% + DR\% \]

2.2 GENEXUS

Genexus generates the relational database and application programs required for the MIS under construction from its formal specifications. The tool stores information about the product specification in a knowledge database (KB)

3. This data is very useful to capture metrics in an automated way.

The methodology behind the tool relies on two premises. First, in a middle or large company nobody has a global vision of the processes. Consequently, it is required to integrate different visions from various users. Second, requirements and database structures exhibit changes over the time.

The formal specification allows modeling the system, generating the database and code for the application. When a change of requirements occurs, a new specification is introduced and the tool reorganizes the database and generates the required code.

The specification uses different types of objects: transactions, procedures, reports, work panels, and menus. The construction of these objects is based on attributes, rules, events, subroutines, and external programs.

Transactions model entities from the real world. The tool designs and generates the database from these objects. Procedures model batch processes for database management. Reports enable the recovery of information. Working panels represent interactive queries to the database. Finally, menus organize all the previous objects.

From the system specification, Genexus generates the database design, the code for data base creation and the application code in the desired programming language. Also is possible to export the specification to a text file and have access to specification details in tabular form. We will use both source of information about the system specification to extract metrics.

Specification Publishing in Genexus

Genexus publishes the KB specification details in two flavor: textual in XML and tabular by means of GxPublic

5, an OLEDB provider. The first one aims to export the KB and the second to enable the programmers to access metadata, extend, and tailor the tool to their needs. GxPublic exposes the specification artifacts and its relationships. This facility was a decisive factor in the election of the tool. From this information, we will compute the (Knowledge Base Intellectual Size) KBIS.

Accessing the System Complexity Data Sources

3 In this paper, following the standard terminology of Genexus, KB or knowledge base refers to a set of system specification objects.

4 In the Genexus users annual meetings can be found the slides of an early proposal from Juan Grompone and a more recent work from José Luis Chalar about Computing Function points from KBs.

5 More information can be found in http://www.gxtechnical.com
There are three sources of complexity measures, all derived from the system specification without human intervention and therefore unbiased, automatically collectables and able to be integrated to a complete life cycle IDE: a) specification artifacts and their relationships (SAR), b) textual system specifications (TSS) and c) source code generated (SCG).

These sources of information are available at all stages of the life cycle. Of course, code generation has some limitations in early stages.

SAR and TSS depend only on the system specification. On the other hand, SCG depends on the language used. To overcome this issue we will consider source code generated in the same language to all surveyed systems. Given the textual nature of TSE and SCG, from them, we count tokens and lines.

Taking advantage of the easier access to the specification artifacts relationships given by Gxpublic, the Knowledge Base Intellectual Size (KBIS) is computed from the specification artifacts and its relationships.

3 DETERMINATION OF THE CANDIDATE METRICS

3.1 RESEARCH QUESTIONS

In spite of being outside of the scope of this article, we present here our main research questions due to their influence in the metric models design.

Our research aims to respond the following questions:

- Which are the early metrics automatically collectable during the software process that can predict risk of MIS developments?
- How can these metrics be applied to assess the project risk?
- How can we integrate the models into an evolutionary software process in order to keep track of the risks?
- Is it possible to obtain these metrics early and continuously to tolerate the changes in requirements? In other words, can we deal with requirement volatility?

The tool presented in this work is the first step in the research of metrics to construct models to be able to respond that questions.

The use of formal specifications is vital because: a) It allows us to use the same hypothesis than [1]; b) allows an unambiguous and automatic collection of metrics; c) allows the early measure of the complexity based on its requirements; and d) allows an objective post mortem measure of project size by comparing SLOC generated in the same language.

Before selecting the metrics, we stated the characteristics required for them, which include practical issues, precision, availability, automatism, people’s resistance, and significance for the user. The following is a summary of the more relevant characteristics found:

- Early availability, we need metrics available early in the development process to estimate.
- Late, we also require late metrics because we use post mortem measures to find correlations with early metrics.
- Repeatable, we require that different observers arrive to the same measures from the same experiment.
- Automatically collectable, human collection of metrics is very hard, time consuming, and subject of bias.
- Easy to calculate, we want metrics that can be calculated with simple algorithms.
- No disruptive, the metric collection must not interfere with the development process.
- Clear significance, users must understand clearly the meaning of the metric.
- Simplicity, we prefer metrics with the less number of parameters following Occam’s razor principle.

3.2 EFFICIENCY

The efficiency of an organization can be measured using interviews, questionnaires, etc. However, these approaches are subject to different interpretations depending on the person who conduct the measure. To avoid confounding factors, at this stage of our research we use organizations with large amount of projects in their history. To measure efficiency we use Putnam’s software equation [5]:

\[
\text{Productivity} = \frac{\text{Size}}{\left(\frac{\text{Effort}}{B}\right)^{\frac{3}{4}} \times \left(\frac{\text{Time}}{3}\right)^{\frac{4}{3}}}
\]

Where:

Size is measured in KLOC, Effort is measured in man year, Time is measured in years and B is a constant depending on the size, which can be calculated from tables in [5].

Because Genexus generates the code in different languages, the notion of line of code does not have sense. We overcome this problem by using applications code generated in the same language, and by counting the generated lines of code. Since we are working with concluded projects time and effort measures are derived manually from the logs of each project. Future work aims to integrate the tool and the models to a complete life cycle IDE. In that scenario the metrics will be collected automatically and the project risk evolution tracked constantly.

3.3 REQUIREMENT VOLATILITY

This metric is used for two purposes. First, it characterizes the real size of the project because the new requirements, and even the dropped ones, require extra work. Second, to observe the evolution of the project, which could be expanding, collapsing, or evolving normally. This approach has been proved very effective in the past [1].
To have a clear measure of the requirement volatility we record the number of new requirements, the dead and the change of the old ones comparing the baselines of two successive months.

\[
RBR = \frac{\text{Number of New Requirements}}{\text{Total Number of Requirements}}
\]

(2)\[
RDR = \frac{\text{Number of Dropped Requirements}}{\text{Total Number of Requirements}}
\]

(3)\[
RCR = \frac{\text{Number of Changed Requirements}}{\text{Total Number of Requirements}}
\]

(3)\[
RV = RBR + RDR
\]

(5)

Each change in a requirement is considered a death and a birth of a new one.

**Complexity**

We measure the complexity at the beginning of the project from the specification, and we compare this early metric with the final complexity post mortem.

We need objective, repeatable, and automatic measures to avoid the interpretation of human experts that can be biased [1], [5].

### Complexity Metrics Derived from Textual System Specification and Source Code

Since many years, we know that complexity is a major contributor to development time [5]. In the past 25 years many new methodologies appeared (prototyping, evolutionary methods, spirals, domain analysis, extreme programming, SCRUM, etc.). However, all these methods use a common metric: lines of code.

Lines of code are objective and easy to measure mainly in post mortem analysis, but very difficult to predict. However, LOC is a standard metric to measure effort because writing a line of code implies an intellectual effort independently of the abstraction level of the programming language [5].

A second issue in this metric is its lack of significance from programmer to programmer, and from language to language. When lines of code are written by humans, its significance could be controversial because programming styles and standards contribute to confound the measure.

The functionality that can be expressed by a certain number of LOCs depends on the programming language. Studies have been conducted to convert from function points to LOC [6]. These issues are minor when we measure automatically generated lines of code. In addition, to minimize the confusion we will count SLOCs generated in the same language.

For the previous reasons, we use Genexus-generated SLOC in Visual Basic to measure the post mortem complexity. The metric can be extracted from the knowledge base.

We compute four complexity measures derived from TSS and SCG: a) Genexus-generated SLOC in VB (SCGVBLOC), b) number of tokens in the Genexus-generated SLOC in VB (SCGVBT), c) number of tokens in the textual system specification (TSST) and d) number of lines in the textual system specification (TSSL).

### Complexity Metrics Derived from the Specification

We wish a metric that relates with the required effort to construct the product. We need to measure the intellectual complexity of the product. The metric should depend directly on the functional requirements, instead of implementation details that are solved by the code generator. For that reason we use some of the artifacts provided in the formal specification which result independent of the generator tool, which are important in developing MIS and represent MIS development essential issues.

Figure 3 shows a causal analysis identifying Genexus candidate specification artifacts related with intellectual effort. In future research, we plan to conduct a similar analysis using data mining. In the following paragraph we discuss these elements and present the formulas used to compute de complexity of the elements of the KBIS.

The elements written with all capital letters are multipliers whose value at present is 1. The values of these multipliers could be adjusted by model calibration if necessary.

### Transactions (TR)

Transactions are the main contributor to complexity that appears during the requirement analysis phase. They represent different user’s visions from which GX generates the database. Their complexity depends on their structure, rules, events, formulas, and subroutines.

\[
TrCx = TrStrCx + \sum_{r} \text{RuleCx}(k) + \sum_{k} \text{EventCx}(k) + \sum_{f} \text{FormulaCx}(k) + \sum_{s} \text{SubroutineCx}(k)
\]

(6)

Where \( r, e, f \) and \( s \) are the transaction’s set of rules, events, formulas and subroutines.

Transaction’s structure complexity depends on the transaction number of levels and its associated table’s complexity.

\[
TrStrCx = \text{TrnNumber of Levels} \times \text{LEVEL \_WEIGHT} + \sum_{k \in a} \text{TableCx}(k)
\]

(7)
This model is based upon transactions, folders, reports, procedures, work panels, and web panels.

**Post-mortem**

This model is based upon transactions, folders, reports, procedures, work panels, web panels, rules, conditions, events and cross references between objects.

### 4 The Tool Architecture

Figure 1 shows the tool context diagram. More details about tool data flows and main processes can be found in figure 5.

Project data (effort and time) is taken from the project logs. Specification details are obtained by means of Genexus Knowledge Manager export function and GxPublic.

![Fig 1 Tool Context Diagram](image)

The textual specifications are processed with text files processor developed in C++. The tabular specification obtained from GxPublic is processed by an application developed with Genexus.

### 5 Field Test and Data Analysis

We used a sample of 13 MIS projects with different sizes developed by a public service enterprise. We found a large variance of the efficiency of this organization probably due to bureaucracies.

The following table shows the collected metrics

<table>
<thead>
<tr>
<th>Project #</th>
<th>Analysis</th>
<th>Early Design</th>
<th>Post-mortem</th>
<th>Generated LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>23160</td>
</tr>
<tr>
<td>2</td>
<td>478</td>
<td>673</td>
<td>1954</td>
<td>36291</td>
</tr>
<tr>
<td>3</td>
<td>2621</td>
<td>3046</td>
<td>4375</td>
<td>184721</td>
</tr>
<tr>
<td>4</td>
<td>142</td>
<td>222</td>
<td>915</td>
<td>43756</td>
</tr>
<tr>
<td>5</td>
<td>388</td>
<td>528</td>
<td>1273</td>
<td>67995</td>
</tr>
<tr>
<td>6</td>
<td>582</td>
<td>822</td>
<td>2175</td>
<td>113632</td>
</tr>
<tr>
<td>7</td>
<td>660</td>
<td>900</td>
<td>1571</td>
<td>104323</td>
</tr>
<tr>
<td>9</td>
<td>819</td>
<td>859</td>
<td>1364</td>
<td>127346</td>
</tr>
<tr>
<td>10</td>
<td>331</td>
<td>411</td>
<td>861</td>
<td>64602</td>
</tr>
<tr>
<td>11</td>
<td>575</td>
<td>700</td>
<td>1475</td>
<td>73901</td>
</tr>
<tr>
<td>12</td>
<td>1442</td>
<td>1732</td>
<td>2454</td>
<td>130484</td>
</tr>
<tr>
<td>13</td>
<td>1191</td>
<td>1426</td>
<td>2841</td>
<td>107759</td>
</tr>
</tbody>
</table>

![Fig 2 KBIS Metrics](image)

As expected, we found a strong correlation between KBIS at the analysis stage and KBIS at the design and post-mortem stages ($R^2 = 0.993$ and 0.8634 respectively). The correlation between analysis KBIS and post-mortem KBIS is smaller because of two reasons: First, at post-
mortality we consider objects that are not necessarily present in the entire sample (e.g., web panels). Second, the sample is too small. We plan to increase the number of projects in order to calibrate the model.

We found a strong correlation between KBIS at analysis stage and the final generated lines of code. This relationship is useful for estimation purposes. Linear regression shows a correlation with $R^2 = 0.8203$. The correlation improves using a polynomial tendency ($R^2 = 0.8791$).

This correlation can be used with practical purposes. The estimation of the complexity (KBIS) and the size (LOC) can be derived very early in the development from the metrics. This first approximation shows a promissory set of metrics useful for estimation purpose.

6 CONCLUSIONS

We introduced a set of early metrics that can be automatically collected from a formal specification. The complexity metric KBIS is a good indicator to estimate the final size of the system. This metric is especially well suited for MIS. It is independent from any detail related to technology, and focuses mainly in the intellectual effort involved in the design of the MIS.

We found a strong correlation between KBIS at the beginning of the project (analysis phase) and the KBIS at later phases. More important, we found a strong correlation between KBIS at the beginning of the project and the final size of the generated code.

We plan to increase our project database to refine and calibrate the metric. After that we will use it in as the complexity parameter for the estimation model that is under construction.

ACKNOWLEDGEMENTS

This work would not have been possible without the collaboration and support of our good friends from the Ministry of Transport and Public Works of Uruguay Enrique Latorres and José Luis Subelzú. Needless to say, the smart comments and reviews of Dr. Javier Segovia of UPM were of great help to improve this paper.

REFERENCES


Fig 3 Fishbone diagram with the causal Analysis showing the contributors for Intellectual Complexity in a Genexus Spec.
Fig 4  Elements considered in each complexity model

Fig 5 Tool data flows and main processes
Fig 6. Correlation between KBIS at analysis and design and post-mortem KBIS

Fig 7. Correlation between KBIS at analysis vs. generated LOC