A Worked Function Point model for effective software project size evaluation

Luca Santillo, Italo Della Noce

Abstract

This work explains the Worked Function Point model for effective software project size evaluation. This model was originally proposed in a contractual framework of fixed cost per function point, in order to achieve a more significant "worked size" to be correlated with development effort and cost. The model is suitable also for internal development estimation, wherever an average productivity is applied to a large set of non-homogeneous software projects. The model structure is based upon measurement of software reuse, replication, complexity, and change request impact during the project lifecycle; specific metrics are proposed for such aspects, and a overall combination of those is suggested to move from standard functional size measures to effective worked size measures. Reuse and replication adjustments include consideration of functional as well as technical "similarity" among functions, over the same or distinct platforms, respectively. Complexity adjustment takes into account both base components internal complexity and external (overall) system complexity; a weighting profile scheme, or worked size mask, is eventually proposed to take into account the intrinsic differences when evaluating complexity for different system types or domains. The change request evaluation is based on a enhancement-like approach, as applied to an on-going project. Specific experimentation and case-studies are carried out in an public administration contractual framework, upon projects extracted from the software portfolio management of the Provincia Autonoma of Trento.

1. Introduction

Trento's Autonomous Province ("PAT" hereafter) is a government agency which, together with Bolzano's Autonomous Province, forms the Trentino – Alto Adige Region. The Italian legal system grants a special autonomy to both the Region and to the two Provinces (a unique case in Italy) from 1948. A special charter, which was reviewed in 1972 and updated in 2001, mainly assigns the two Provinces the right to issue their own laws in a broad number of areas and to execute the relative administrative functions. In other words the Autonomous Province is a partial replica on a local scale of the nation's complex administrative structure. PAT is composed of 67 different service areas, distributed over 15 departments [1].

Obviously, PAT needs to be provided with several software applications to manage a broad variety of data, at a central level, at single offices' level throughout the territory and at the institutional counterparts level (such as schools, public bodies and local authorities). The software supply is governed by specific contractual agreements and to-date this is executed almost exclusively by a single software house, controlled partially by public participating interest (PAT itself): Informatica Trentina ("IT-supplier") [2].

The number of managed software application is over 100 systems, 58% of which are of the "4GL" type, corresponding to a (declared) total of 64,000 IFPUG unadjusted Function Points, and 42% of which are of the "3GL" type, representing a (declared) total of 48,000 "equivalent" Function Points (obtained from the Source Lines of Code using a contractually defined conversion factor) (early 2004). PAT's specific Information & Organisation Service ("SOI") area structure, among other aspects, is responsible for the following activities:

- to promote and coordinate implementation of the provincial IT system and to define the IT needs, also on the basis of specific requests made by others provincial structures;
- to manage relations with the companies that provide the IT services, as well as checking and monitoring the aspects relating to the supply of the services.

Finally, at the end of 2002, SOI hired Data Processing Organisation ("DPO") as a consulting partner in relation to the Function Point assessment of the software applications and projects. On a regular basis, based on end users requirements and requests, a number of new development and enhancement software projects are started and installed - upon approval by SOI – by the IT-supplier. Excluding some exceptions (e.g. legacy systems and data warehousing) such projects are measured in IFPUG Function Point and acquired by PAT through a fixed cost per function point mechanism, which has been set up by early versions of the contractual agreement among customer and supplier. Following a first step focused on the software application assets [3], more recent analyses showed that the fixed cost per function point approach leads very often to cases where the supply results over-paid or under-paid with respect to the real extent and scope of the software project being realized.

2. Functional size versus effective size

The effort needed to release a given software project depends on the number and intrinsic nature of the technical items that should be designed, built and tested, more than on the logical functionalities that those items aim to implement. This is why functional measures *alone* are not particularly well correlated to effort, e.g. as it may be seen from the ISBSG software benchmarking database. A clear evidence of the previous statement is reached when we consider the practice of reusing software items as module libraries, components catalogues and the like; the effort needed to realize the overall system is not proportional at all to the logical "services" required (Function Points), but it will be proportional to the number and nature of the new items to be build and to the number and difficulty of the integrations of already available, reused items that should be carried on in the general system architecture.

Basically, the effort needed to complete the early phases of the software life cycle is expected to be more closely proportional to the functional software size (the weight of requirements), whereas the effort for the remaining phases should be more proportional to "technical" sizes [4].

Using a single fixed cost per (functional) size unit to price the supply in an open multi-project contract determines a subtle and dangerous problem. Due to the desired uncertain nature of the open contract in terms of requirements (what, how, when and how much) useful to maximize the flexibility of the outsourcing choice, there is no guarantee that the final mix of actual required projects be the same as the estimated one on which the productivity and cost averages have been originally calculated and agreed upon. Consequently, the initial economical consistency validation – made on the basis of a "wrong" project portfolio – might be invalid for the actual project portfolio. In other words, it is paid a wrong price for the right things or the right price for the wrong things!

2.1. From Released Size to Worked Size

In order to avoid the problems of the fixed unitary cost for the whole software contract in case of a multi-project open supply, it is possible to use the following scheme, including the concepts of Released FP and of Worked FP (Fig. 1, below).

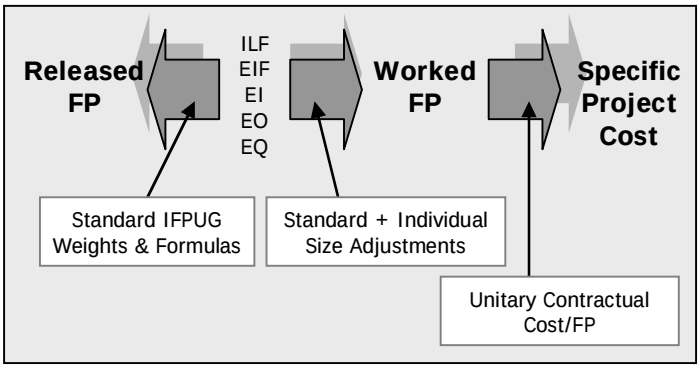

Figure 1. *Released vs. Worked FP in a "fixed cost per FP" contractual framework.*

The key points are the following: the Released FP (measured according to the standard counting rules) are not always corresponding to the actual extent of the "worked" portions of software. There are two main reasons for that: the reuse practice and the need to replicate the same logical functionalities in different technological environments [5]; it seems more appropriate to evaluate the impact of these factors on the functional size (involved by the reuse or by the replication) instead than on the unitary or global cost per FP; for this reasons the "Worked Function Points" are no more a pure functional measure but a "productivity contaminated" measure useful to produce the cost evaluation in specific cases.

The next relevant productivity factor to be taken into account is complexity [6]. Historically, one of the possible reasons for initial resistance to the introduction of functional measurement methods in the industry was the fact that another important productivity factor – the intrinsic, non-functional complexity of software systems – seems to be not enough taken into account; this led to the introduction of some general system characteristics, such as complex processing, in the VAF factor of the IFPUG method, or the CPLX adjustment factor of COCOMO-like estimation models, which are not measured differently within the set of software items being developed. Besides that, it is also debated the fact that the functional complexity measured by the IFPUG measurement method is bounded in its values (Low, Average, High), resulting in underestimated sizes for "extra large" software objects. In the WFP approach, a practical attempt is made to consider complexity as another adjustment, or weighting, factor, for effective size of the software functions. Briefly, the complexity adjustment expresses the relative variation in productivity per WFP with respect to the specific analysis and realization "difficulty".

Finally, considering the way software requirements are often found to be incomplete or evolving through all the project lifecycle, a third class of adjustment is being considered in the WFP model for the PAT contractual environment: the change request. The standard measurement methods cannot still adequately express the so-called "*scope creep*": often, the difference between initial and final counts is not a good measure of change request, since old requirements can be simply replaced by new ones, and the rework activities are not visible through this simple metrics. A change request approach is developed to measure the scope creep adequately.

The WFP result is a "functional productivity contaminated" measure that is simply multiplied by the originally fixed unitary cost to give the final cost of any single project in the multi-project contract. Using this approach is possible to keep the simplicity of a fixed unitary cost approach and the fairness of an articulate and situational costing approach. A good example of the advantages of this kind of approach is provided by the assessment of an enhancement project that takes reuse into account. A change that involves adding a field in a template and the implementation of a new function would be considered equivalent in economic terms in the case of an agreement based on a fixed price per FP. Whereas, the considerations regarding reuse assign the correct weight to the two projects based on the worked FP's. Another advantage provided by the WFP approach is that in the on going negotiation, attention of the parties tends to focus on aspects relative to productivity, rather than on size in FP terms, which once again represents an objective aspect in the case of disagreement as regards the economic assessment of a given project.

The two Function Point assessments will then be available for a given software project:

- the result obtained by applying the IFPUG standards and corresponding to the actual software services usable by the users (Released FP's);
- the result that provides an assessment of the "worked" functionalities and therefore the *effective* size of the project that has been implemented (Worked FP's).

3. The WFP Model Instance in the PAT Environment

Due to overlapping of factors, and the fact that the IFPUG General System Characteristics are not appliable to specific software functions or subsystems, but only to the overall software system being developed, the WFP approach takes as an input only the Unadjusted size, or UFP. In the following sections, the specific adjustment factors are provided.

3.1. Software Reuse Impact

Functional reuse may be defined as the re-utilization of user recognizable and existent logical data structures and functionalities to build up new logical features. Depending on the particular architectural environment, we might have an extreme situation in which the functional reuse is very high but the technical capability of reusing existing "physical" software items is very low: we must re-build the desired logical item almost by scratch. This is the case, for example, when, in a multi level client-server architecture, we want to deliver a functionality logically similar to an existing one, but in a technical environment completely different from the original one. Functional reuse means possible (likely) exploitation of pre-existing requirement documentations, analysis and design models and schemes, and the like.

The **functional reuse** size adjustment factor ***fR*** is evaluated for every function based on the following cases, for both transactional and data functions (Table 1, next page). Specific PAT guidelines are provided to identify the correct case, based on DET/FTR similarity of the functions being examined (not shown here). The proposed adjustment values derive, with some variations, from the NESMA enhancement approach [7]. Here, the approach is applied also in case of internal reuse for a new development project.

Table 1. *Functional reuse adjustment coefficients by case.*

Oper.	**Case**	***fR***	**Note**
ADD	New *ex novo* function	1	
	Imported/copied from another system or subsystem.	0,25	
CHG	A small amount of data elements, referenced files, or processing logic is changed.	0,25	Also for imported functions, where **some** data elements or references are added or removed.
	A significant amount of data elements, referenced files, or processing logic is changed.	0,60	Also for imported functions, where **many** data elements or references are added or removed.
	The amount of changes to the functions is so high to require a total re-analysis and re-design of the function.	1	Also for imported functions, where **most** data elements or references are added or removed.
DEL	Simple deletion of function.	0,10	
	Function deleted and replaced with a similar one.	0,25	The adjustment applies to the newly added function (as per add-by-import functions). The deleted function is assigned a null coefficient.

Technical reuse may be defined as the re-utilization of existing physical data structures and software items (modules, objects, programs etc.) in order to build up new technical items to be used in the construction of new logical features. Depending on the particular functional requirements, we might have an extreme situation in which the functional reuse is very low but the technical capability of reusing existing "physical" software items is very high: we can build the desired new logical feature using almost effortlessly some existing technical "items". This is the case, for example, when we want to deliver a set of functionalities to manage (CRUD) a number of logical files which are similar in structure (i.e. unique id., description, numerical values) but different in contents (i.e. money conversion table, time conversion table, length conversion table etc.).

The **technical reuse** size adjustment factor *tR* is evaluated for every transactional function based on the following cases (Table 2, below). Specific PAT guidelines are provided to identify the correct case (not shown here). This approach is applied also in case of internal reuse for a new development project (exploitation of code from one function to another, e.g. *insert*, *update* and *delete* elementary processes over the same logical file).

Table 2. *Technical reuse adjustment coefficients by case.*

Level	**Case**	***tR***	**Note**
High	No new source code is developed with respect to the selected existing function.	0,05	Includes component selection, non-significant source code adaptation and minimal test.
Average	Less than 25% new code developed or modified for the function.	0,40	Includes component selection, small source code adaptation and base test activities.
Low	25% to 50% new code developed or modified for the function.	0,70	Includes component selection, normal source code adaptation and typical test activities.
Null	More than 50% of code is developed or modified for the function.	1	Includes component selection, significant source code adaptation and massive test.

3.1.1. Phase coefficients and final reuse impact

In order to combine the functional and technical reuse impacts for each function, weighting phase coefficients are fixed for a generic waterfall model:

- $a = 0{,}15$ for analysis tasks;
- $d\&b = 0{,}55$ for design and building tasks;
- $t = 0{,}25$ for test tasks.

For the i^{th} function impacted by the development or enhancement project, the overall reuse impact factor IR is obtained combining the previous factors with the following formula:

$$IR_i = (a + \tfrac{1}{2}\, t) \times fR_i + (d\&b + \tfrac{1}{2}\, t) \times tR_i$$

3.2. Software Replication

Software replication means porting the same functionality (one or more transactional and data functions) from one platform/technical environment to another (or more). Exactly the same considerations as for the software reuse can be made for software replication, if we denote the second, third, nth replica as reused with respect to the first, original version of functionality involved. So exactly the same aspects, factors and cases as for reuse are applied in case of replication. An index is added to the reuse factor *IR* to distinguish every platform/environment copy of functions: IR^2, IR^3, and so on. By definition, the first platform/environment cannot be assigned any replication correction, but it already may have some reuse corrections. Please note that often the most reuse in replication is functional and not technical (no new functionality has to be analysed and designed, but the source code have to be adapted – possibly heavily – to perform on different platforms/environments).

3.3. Software Complexity Impact

We distinguish a concept of complexity, which is applicable to the software system, from the complexity of the software development process; the latter could be related to the different phases, methodologies, team composition and coordination factors, which are applicable when executing the software project – those factors are process factors and are not included when evaluating the software system complexity: in the actual approach such process complexity is averaged in the contractual unitary cost per FP and, however, is basically under control of only the supplier/developer.

For transactional functions, the IFPUG method provides size values that depends on the type (EI, EO, EQ: external input, output, or inquiry). Moreover, IFPUG values are upper-bounded, so that extremely complex processes are not adequately differentiated from not-so-complex processes. The last question is solved by means of extended complexity matrices, or equivalently of *functional complexity* (*fC*) adjustment factors depending on data element types and file type referenced by the processes [Tables 3, 4, next page].

Table 3. *Extended IFPUG functional complexity matrix (left) and adjustments coefficients and unadjusted function point values (right) for EI functional processes.*

	DETs'				
FTRs'	**1-2**	**3-4**	**5-15**	**16-30**	**31+**
0-1	VL	L	L	A	H
2	L	L	A	H	H
3-4	L	A	H	H	VH
5-7	A	H	H	VH	VVH
8+	H	VH	VH	VVH	VVH

CPLX	*fC*	UFP (EI)
VL	0,5	1.5
L	1	3
A	1	4
H	1	6
VH	1,67	10
VVH	3,0	18

Table 4. *Extended IFPUG functional complexity matrix (left) and adjustments coefficients and unadjusted function point values (right) for EO and EQ functional processes.*

	DETs'				
FTRs'	**1-3**	**4-5**	**6-19**	**20-40**	**41+**
0-1	VL	L	L	A	H
2-3	L	L	A	H	H
4-6	L	A	H	H	VH
7-10	A	H	H	VH	VVH
11+	H	VH	VH	VVH	VVH

CPLX	*fC*	UFP (EO)	*fC*	UFP (EQ)
VL	0,5	2	0,5	1.5
L	1	4	1	3
A	1	5	1	4
H	1	7	1	6
VH	1,57	11	1,67	10
VVH	2,86	20	3,0	18

Similarly, the IFPUG method shows upper-bounded ranges for files; similarly, extended experimental matrices and values are provided (Table 5, below). Low ranges are provided in order not to oversize trivial cases, such as reference tables made of just code and description fields. Highlighted portions are the extensions to the original IFPUG values.

Table 5. *Extended IFPUG functional complexity matrix (left) and adjustments coefficients and unadjusted function point values (right) for Internal and External logical files.*

	DETs'				
RETs'	**1-5**	**6-19**	**20-50**	**51-90**	**91+**
1	VL	L	L	A	H
2-3	L	L	A	H	H
4-5	L	A	H	H	VH
6-10	A	H	H	VH	VVH
11+	H	VH	VH	VVH	VVH

CPLX	Adj.	UFP (ILF)	*fC*	UFP (EIF)
VL	1,75	4	1,67	3
L	1	7	1	5
A	1	10	1	7
H	1	15	1	10
VH	1,67	25	1,5	15
VVH	3,0	45	2,5	25

Regarding technical (algorithmic) complexity, which actually does apply for transactional functions only, we recall that the IFPUG method (release 4.1 and higher) makes use of 13 actions to assist in classifying the function type; these actions may be included in some or all of the processing logic of elementary processes. The *technical complexity* (*tC*) adjustment factor is evaluated, based on whether the given process performs "creation", "check", or no specific action type, and adjustment coefficients are proposed based on the range of utilization of such types of actions in every transactional function (Tables 6-7, next page). These adjustment coefficients are firstly obtained by heuristics and values from literature (e.g. De Marco's Bang complexity weighting factors [8]); they are under validation and can be agreed upon in the contractual framework. No technical complexity adjustment is considered for data function types (logical files).

Table 6. *Summary of processing logic actions that may be performed by EIs', EOs', and EQs'. The 13 actions do not by themselves identify unique elementary processes.*
Legend: c = "can be performed", m = "mandatory", m = "mandatory at least one", n = "cannot be performed", ILF = "Internal Logical File", EIF = "External Interface File".*

	Action	EI	EO	EQ	Type
1	Validations are performed	c	c	c	Check
2	Mathematical formulas and calculations are performed	c	m*	n	Creation
3	Equivalent values are converted	c	c	c	Check
4	Data is filtered and selected by using specified criteria to compare multiple sets of data	c	c	c	Check
5	Conditions are analyzed to determine which are applicable	c	c	c	Check
6	One or more ILFs' are updated	m*	m*	n	Creation
7	One or more ILFs' or EIFs' are referenced	c	c	m	-
8	Data or control information is retrieved	c	c	m	-
9	Derived data is created by transforming existing data to create additional data	c	m*	n	Creation
10	Behaviour of the system is altered	m*	m*	n	Creation
11	Prepare and present information outside the boundary	c	m	m	-
12	Capability exists to accept data or control information that enters the application boundary	m	c	c	-
13	Data is resorted or rearranged	c	c	c	-

Table 7. *Technical complexity adjustment coefficients for transactional functions.*

tC		Check Types in Function		
		0	**1**	**2+**
Creation Types in Function	**0**	0,75	0,75	1,00
	1	1,00	1,00	1,25
	2+	1,25	1,50	1,75

The overall complexity impact factor for each function (*IC*) for a given function is simply given by the product of the two described adjustments (functional extension for both transactional and data functions, and technical complexity for transactional functions only):

$$IC = fC \times tC$$

3.4. Change Request Impact

Once a requirements and measurement baseline has been established for the project, any Change Request to this baseline should be evaluated in terms of impact on the existing and future systems, measured, managed as if it was a functional enhancement maintenance request issued when the product is not completely done (see later). The sizing of the CR's allows to reconsider economical rewards towards the supplier and/or requirement process improvement actions towards the users who initially formulated the requirements [9].

The proposed approach is to measure the CR size (FP_{CR}) for those CR's that are issued during the development stage as if they were enhancement requests upon the system being

developed as it was already completed. Such measure is to be corrected by some adjustment factors to keep into account the reuse level (for reworking the same functions) and/or the waste of effort due to the changed or abandoned requirements. The FP_{CR} amount may be derived for both development or enhancement software project types.

In order to count the FP_{CR} for a given ongoing project, an initial count is required, that corresponds to the initial view of logical files and elementary processes that should be delivered at the project completion. This initial count is assumed as the baseline for the evaluation of the subsequent possible CR's. Each time a single relevant CR or a set of CR are issued, they will be analysed in terms of functional impacts they have on the baseline using the following formula:

$$FP_{CR} = FP_{ADD} + \Sigma_i (FP_{CHGA,i} \times CL_i \times LCP_i) + FP_{CONV} + \Sigma_j (FP_{DEL,j} \times LCP_j)$$

where:

- FP_{ADD} (*Added Function Point*) is the total contribution of any data or transactional function that is to be added to the baseline for the CR to be implemented.
- $FP_{CHGA,i}$ (*Changed After Function Point*) is the contribution from the i^{th} data or transactional function that is already present in the baseline and is to be modified for the CR to be implemented.
- CL_i (*Change Level*) is a number between 0 and 1 (or 0%-100%) representing the relative amount of logical change required on the i^{th} data or transactional function that is already present in the baseline, for the CR to be implemented – a value of CL_i close to 0 is associated to a situation where the proposed changes are impacting in a marginal way the existent functionality; a value of CL_i close to 1 is associated to a situation where almost nothing could be reused to implement the change.
- LCP_i (*Life Cycle Progress*) is a number between 0 and 1 (or 0%-100%) representing the relative weight of the life cycle phase of the project in which the CR is issued.
- FP_{CONV} (*Conversion Function Point*) is the total contribution of any data or transactional function that is to be developed, but used only once (*una tantum*) for the CR to be implemented.
- $FP_{DEL,j}$ (*Deleted Function Point*) is the contribution from the j^{th} data or transactional function that is to be deleted for the CR to be implemented (a *Life Cycle Progress* factor applies to deletions, as well).

When any CR should have been accepted and incorporated into the overall development or enhancement project, the baseline project size should be updated using the standard IFPUG approach. Any new CR should be measured using the same technique described previously starting from this updated baseline (i.e. the project count goes through several releases, before being completed, depending on how many CR's are proposed and approved). At the very end of the project, the sum of all the occurred CR's produces the overall size of occurred change request for that project (*Total* FP_{CR}). This amount represents an addition to the WFP of the project, to be recognised as the scope creep with respect to the initial measure for the project.

Moreover, if we normalize the size of the CR to the total number of Released Function Point for the project, we can quantify the *requirements turnover*, or "project turbulence", and the extent of some of the risks correlated with that.

4. Overall WFP Formula and conclusions

Combining the factors that are illustrated in the previous sections, we obtain a formula for the calculation of the Worked Function Point amount for a given software project:

$$WFP = \Sigma_{i,j} (FP_i \times IR_i^j \times IC_i) + Tot.FP_{CR}$$

where the index *i* runs over all the functions involved and identified for the project, and the index *j* denotes all the platform/environments eventually involved for replication of functionality (2, 3, and so on).

As stated in section 2, the number of Worked Function Points is expected to be better correlated to effort & cost for a software project than the Release Function Points, that is the standard "functional only" size which is so far involved in so many open contract frameworks with fixed unitary cost. Of course, an extended study must be performed to validate the heuristic coefficients that have been proposed in the current work as a correction from standard to effective size, and to eventually improve the correlation. PAT and DPO are developing an extensive validation process, based on real cases from the large scale software delivery agreement, eventually involving the IT-supplier.

Possibly, future works on this subject will provide improved values and formulas within the overall stated framework for an effective use of functional metrics for effort and cost estimation and correct payment process.

It is worth note that many aspects of the proposed model are not set towards a reduction, or limitation, of the size amounts and measures; rather, the WFP approach helps distribute in a more agreeable and realistic way the weight of a fixed cost per size unit over a large and varying project portfolio. So, "small" projects with heavy intrinsic complexity, or excessive requirement volatility, thus requiring more effort than other even "larger" projects, will be recognized and treated with more attention from both the economical and the management perspectives, with higher satisfaction of both the customer and the IT-supplier involved areas. Conversely, "large" projects made by trivial, "all-the-same" functions, will be recognized as not-so-heavy projects, and the counterparts will be able to redistribute part of their previously allocated budgets to other and more relevant projects or activities. Moreover, the change request measurement could permit do discover, analyse and improve critical aspects in the negotiations of the software requirements and of the software projects, in general.

Nowadays, the desire to oversimplify the content of any agreement is strong, but the problems overlooked at the negotiation and contractual phase will always explode in the implementation phase, leading to illegal practices or damaging conflicts. This work proposed a useful approach to improve the management of an existing long-term software contract based on fixed cost per size unit, with the primary purpose of "lower[ing] the level of litigation and continuous negotiation experienced in the final years of the past century!" [3]

References

[1] PAT Web Site: www.provincia.tn.it, Trento's Autonomous Province (2005).

[2] *Provincial law n. 10, 6 May, 1980*, Trento's Autonomous Province (1980).

[3] Della Noce I., Nardelli L., Santillo L., *Measuring Software Assets in a Public Organization – A Case History from Provincia Autonoma di Trento*, Software Measurement European Forum 2004, Rome, 28-30 January 2004.
Available at: http://www.dpo.it/english/resources/papers.htm.

[4] Meli, R., *The Software Measurement Role in a Complex Contractual Context*, Software Measurement European Forum 2004, Rome, 28-30 January 2004.
Available at: http://www.dpo.it/english/resources/papers.htm.

[5] Poulin J.S., *Measuring Software Reuse*, Addison-Wesley, 1997.
ISBN 0201634139.

[6] Santillo L., *Software complexity evaluation based on functional size components*, International Workshop on Software Measurement 2004, Berlin, 3-5 November 2004.
Available at: http://www.dpo.it/english/resources/papers.htm.

[7] NESMA, "Function Point Analysis for Software Enhancement", Netherlands Software Metrics Users Association, 1998.
English translation: 2001. Available at: http://www.nesma.nl/english/.

[8] DeMarco, T., *Controlling Software Projects*, Yourdon Press, New York, 1982.
ISBN 0131717111.

[9] Meli R., *Measuring Change Requests to support effective project management practices*, 12th European Software Control and Metrics Conference 2001, London, 4-6 April 2001.
Available at: http://www.dpo.it/english/resources/papers.htm.