

Software complexity evaluation based on functional size components

Luca Santillo

CFPS

luca.santillo@gmail.com

Abstract:

This work introduces a self-consistent model for software complexity, based on information which can be typically collected at early stages of software lifecycle, e.g. in the functional specification phase, when a functional size measurement is also usually performed. The proposed model considers software complexity as structured into three bottom-up stages of the software architecture (from internal complexity of each base functional component to the overall structural complexity of the software system). Conceptually, complexity can be considered as proportional to dimensionality = scale \times diversity, hence any complexity factor can be seen as an issue either of scale or of diversity, originating a corresponding specific factor of implementation difficulty, or complexity driver; such mapping suggests which factors could be referred to software structure and size, while other factors should be referred more adequately to the software development process. For each stage of the model, a description an one or more specific metrics – at different detail level – are proposed in order to provide an overall software complexity indicator to support in distinguish software systems from a complexity perspective. Moreover, it's shown how some or all of the complexity factors – at various stages – can be integrated with typical functional size measures (COSMIC, IFPUG) in order to normalize the latter in a more realistic software estimation process. A weighted profile scheme, or complexity mask, is also proposed to take into account the intrinsic differences when evaluating the complexity for systems of different type or domain.

Keywords

Software, complexity, system, base functional component, estimation

1 Introduction

This works introduces a model for software complexity evaluation, based on functional size measurement elements. The main reason for such model is to improve the estimation capability for development effort against the software functional analysis. Although the size of a software system is considered its main cost driver, and statistical analysis clearly shows a positive correlation between size and development effort, the only “size” information is usually not enough to provide an accurate estimation, such as a project manager could want to achieve

in order to plan, justify and/or negotiate time and resource needs on a specific software project. One of the possible reasons for initial resistance to the introduction of functional measurement methods in the industry is the fact that intrinsic 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.

In fact, software complexity is one of the most impacting factors for project effort, beside size; other relevant factors are, for example, software reuse and change requests in the on-going project process. Since reuse and change requests can be viewed as size adjustment factors in terms of *effectively worked software size* [7], an attempt is made to consider complexity as another adjustment, or weighting, factor for *effective size* of a software project. Similar attempts have been already made, more or less explicitly, in the history of software functional measurement, e.g. the value adjustment factor in the IFPUG method (several reasons exist to reject such attempt). This work proposes a structured approach to complexity evaluation, taking also advantage of past proposals, e.g. some complexity factors proposed for the COSMIC-FFP framework [9].

Many authors pointed out the main difficulty in dealing with complexity: we do not have a standard, specific, unanimous definition of complexity. Or, the definitions we can provide are too generic to generate a universally accepted notion of any complexity metrics. Therefore, in this work we deal with a generic definition of complexity, generically stated as “that characteristic that makes two equally-sized portions of software different in terms of comprehension and corresponding realization difficulty”. The proposed complexity evaluation model is pragmatically based on a GQM approach (Goal-Question-Metric), in order to produce an evaluation model for intrinsic complexity, which could be immediate to understand and to apply to real cases (Table 1, below); moreover, several approximation orders are proposed, at different levels of accuracy, in order to let the reader choose among metrics, depending on the wanted accuracy in estimation. The further relationship among intrinsic software complexity and the varying human resources capability (experience factor) to deal with it is not studied in this work, although an obvious link exists.

Goal	Question	Metric
Differentiate equally-sized software portions according to realization difficulty.	What makes the two portions different for a complexity point of view [on distinct levels]?	Several proposals [on distinct levels].

Table 1: A generic GQM approach to software complexity evaluation model.

Another complexity issue is the distinction between problem complexity and solution complexity [1]. Since we're looking for a complexity evaluation model which can be applied in early project stages, when the solution is actually still to be designed and built, typically solution complexity indicators, such as Software Science or Cyclomatic Number cannot be taken into account. The issue, whether the hereby proposed approach should be considered as a pure problem complexity model, either it's a mixed approach of problem complexity and high level, functional solution complexity, is left for further analysis.

2 Overview

Based on modern functional size measurement methods, we consider any software system as a hierarchy, which essential form is made of three levels:

- higher, *whole-system* level,
- intermediate level of (macro, decomposable) functional *modules*, and
- lower, high-detail level of (not significantly decomposable) *base functional components*.

The lower level is that of the Base Functional Components in any standard Functional Size Measurement method (Fig. 1). In the COSMIC method, modules typically correspond to *software peer items*, belonging to a given layer; the layer concept could be considered orthogonal to such structure (each layer as a "system") or could be re-introduced as a further level (to investigate). Note that, while in COSMIC method the BFC is intended as the Data Movement (each functional process contains two or more DMs'), for general discussion we will take the functional process, corresponding to the elementary process in the IFPUG method, as the effective BFC in further discussion.

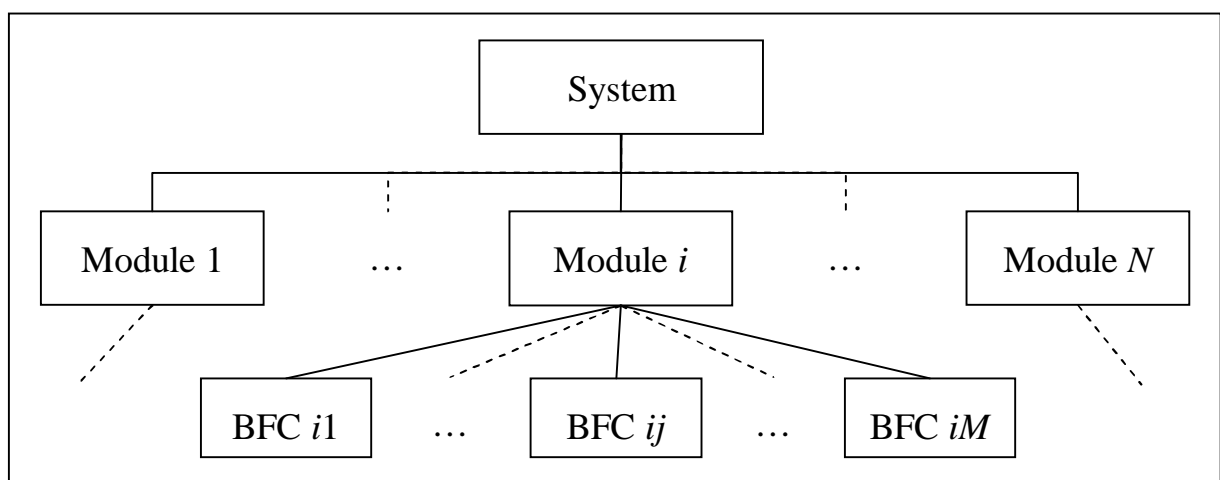


Figure 1: Generic software system conceptual structure.

2.1 About types of complexity and their relationships

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. Moreover, complexity can be viewed as a matter of scale (“the element being studied contains many components”) either of diversity (“the element being studied contains components of different types”). Both are possible complexity indicators (in terms of corresponding difficulty of realization), so both should be considered as complexity drivers. Note that, at the BFC level of any functional size measurement method, we find some kind of measure of the scale complexity:

- IFPUG: “functional complexity” (based on the amount of data element types and record element types for files, and of data element types and file types referenced from processes – with *bounded* ranges);
- COSMIC: unbounded size (based on the amount of data movements per process)

A conceptual relationship can be expressed between development difficulty, complexity and scale and diversity:

$$\text{difficulty} \propto \text{complexity} \propto \text{dimensionality} = \text{scale} \times \text{diversity},$$

where the “ \propto ” symbol means “is directly proportional to”. Software systems become more “complex” as their structures grow in scale and diversity:

- *scale* means “more of the same”, or “more of a similar kind of thing”; increases in scale can often be treated by a corresponding increase in the capacity of existing mechanisms, such as more development time, more people, etc.: scale is a quantitative effect, not a qualitative effect;
- *diversity* means “different stuff,” or “more of a different kind of thing”; increases in diversity cannot usually be treated with existing processing mechanisms – instead, diversity usually requires the extension or modification of existing mechanisms to accommodate the new diversity (sometimes entirely new conceptual models must be created to accommodate increased diversity): diversity is a qualitative effect, not a quantitative effect.

Dimensionality is a quantity that conceptually represents the complexity of a software development project. This quantity can be approximated in several ways, as shown in next sections, based on which factors we choose to measure scale and/or diversity.

2.2 Summarized complexity model structure

The overall structure of complexity evaluation model hereby proposed is reported in Table 2. Sections 3 to 5 report definitions for each stage evaluation, and different metrics approximation orders. Refer to section 6 for details about how to combine metrics collected at different stages.

Stage	Component type	Aspects & Approaches
BFC	Transactional functions	Size (extensions to standard)
		Processing logic/manipulation (primary)
		Amount of manipulation types
		Amount of manipulation instances
	Data functions	Size (extensions to standard)
		Global usage (overall access type)
		Specific usage (referring process types)
		Occurrence usage (referring processes)
Module	Transactional component	Averages of underlying level components
		Internal exchanges (cohesion)
		External exchanges (coupling)
	Data component	Averages of underlying level components
		Internal relationships
		External relationships
System	Modules	Averages of underlying level components
		Internal relationships (inter-modules)
	Layers	Internal relationships (inter-layers)
	Whole system	External relationships (i.e. communication with external systems)

Table 2: Complexity evaluation model.

The first stage represents the evaluation of each BFC complexity as a “per se” element. The intermediate stage can be evaluated as both a derived complexity (from the averages complexity of its components) and a separate complexity factor, based on the relationships between its components and as well as other

components. The system overall level, similarly, can be represented by both an average over its modules and new factors, based on relationships between modules and/or external systems – layers consideration is included.

3 Complexity evaluation & metrics for the lower level (BFC)

3.1 Transactional functions

Transactional Base Functional Components of the IFPUG measurement method are the elementary processes (External Input, External Output, and External Inquiry). Although BFCs’ in the COSMIC method are the Data Movements (included/performed by the functional processes), we shall consider the elementary/functional process as the base component of a generic measurement method, for sake of homogeneity and application of subsequent consideration about internal complexity.

Next, Table 3 reports a comparison of corresponding characteristics elements of functional/elementary processes for IFPUG and COSMIC approaches.

IFPUG	COSMIC
DETs’ (Data Element Types)	DMs’ (Data Movements – Entries & eXits)
FTRs’ (File Types Referenced)	DMs’ (Data Movements – Reads & Writes)
Type (Input, Output, Inquiry)	Type (Data Manipulations – currently not identified)

Table 3: Comparison of characteristic elements of functional processes for IFPUG and COSMIC measurement methods.

Moving from scale to diversity aspects in evaluating functional processes complexity, we find that the first possible level metrics are already considered by the current methods: the size itself, based on the quantity of internal characteristic elements, is given, with some relevant differences:

- the IFPUG approach provides size values that depends on the Type, thus mixing immediately scale and diversity (recent versions provide an exhaustive table of actions that may help in identify the processing logic type); moreover, IFPUG values are upper-bounded, so that extremely complex processes are not adequately differentiated from not-so-complex processes;

- the COSMIC approach provides a pure measure of scale (amount of Data Movements per process), but currently no diversity in data manipulation is standardized; the COSMIC approach has no upper-bound issue.

A digression about IFPUG limited values (transactional functions)

In order to extend the capability of the IFPUG approach, when a constraint is to use such method, extended experimental matrices and values are provided (Tables 4 and 5, functional processes). Note that not only the upper limits are augmented: some lower ranges are provided in order not to oversize trivial cases. The functional complexity for batch-like processes – no or little information crosses the system boundary, while many files are internally referenced – is augmented. Highlighted sub-matrices show the original values from the standard IFPUG approach.

		DETs'						
FTRs'	1-2	3-4	5-15	16-30	31+	CPLX	EI	
0-1	VL	L	L	A	H	L	3	
2	L	L	A	H	H	A	4	
3-4	L	A	H	H	VH	H	6	
5-7	A	H	H	VH	VVH	VH	10	
8+	H	VH	VH	VVH	VVH	VVH	18	

Table 4: Extended IFPUG functional complexity matrix (left) and unadjusted function point values (right) for external input functional processes.

		DETs'							
FTRs'	1-3	4-5	6-19	20-40	41+	CPLX	EO	EQ	
0-1	VL	L	L	A	H	L	4	3	
2-3	L	L	A	H	H	A	5	4	
4-6	L	A	H	H	VH	H	7	6	
7-10	A	H	H	VH	VVH	VH	11	10	
11+	H	VH	VH	VVH	VVH	VVH	20	18	

Table 5: Extended IFPUG functional complexity matrix (left) and unadjusted function point values (right) for output and inquiry functional processes.

From scale to diversity for functional processes

In order to take into account the diversity of functional processes, the IFPUG list of 13 actions that may be included in some or all of the processing logic of elementary processes is reported (Table 6, below).

	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 6: Summary of which forms of processing logic 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".

The *internal* processing logic component, or data manipulation in COSMIC terms (roughly equivalent to a first order approximation of algorithmic complexity), may contain two basic action types: check of existing data (values) and creation of new data (values). The external processing logic component, or data movement in COSMIC terms, is made of data flows across the boundaries and/or of I/O operations on referenced data groups; such component is already taken into account by the size of the process, based on the DET/FTR or DM amounts for that process (*scale*). Therefore, a first consideration about *diversity* of processes can be made, based on whether the given process performs “creation”, “check”, or no specific action type (column “classification” in previous Table 6). Each classification can be assigned a different complexity coefficient, e.g.:

- nominal = 1.0 (or even less than unitary) for no specific action type;
- 1.5 for “check” presence;
- 2.0 for “creation” presence;
- $1.5 \times 2.0 = 3.0$ for both “check” and “creation” presence.

Above values are suggested by heuristics; some hints in defining such coefficients may come from De Marco’s Bang complexity weighting factors, which can be mapped onto the newer IFPUG list of actions (Table 7, below).

Class	Weight	Class	Weight
Separation (divide incoming data items)	0.6	Synchronization (decide when to act or prompt others to act)	1.5
Amalgamation (combine incoming data items)	0.6	Output Generation (net output data flows – other than tabular outputs)	1.0
Data Direction (steer data according to a control variable)	0.3	Display (graphics, pictures, etc.)	1.8
Simple Update (update one or more items of stored data)	0.5	Tabular Analysis (formatting and simple tabular reporting)	1.0
Storage Management (analyze stored data, and act based on the state of that data)	1.0	Arithmetic (simple mathematics)	0.7
Edit (evaluate net input data at the man-machine boundary)	0.8	Initiation (establish starting values for stored data)	1.0
Verification (check for and report on internal inconsistency)	1.0	Computation (complex mathematics)	2.0
Text Manipulation (deal with text strings)	1.0	Device Management (controlling devices adjacent to the boundary)	2.5

Table 7: Complexity weighting factors by function class ([4]).

A further, mixed scale/diversity factor, or complexity level measure, can be derived from considering not only the predominant classification of action type in the process, but also the *amount* of distinct actions performed (one or all of the 13 IFPUG actions can be included in any process). A combination approach has to be chosen among summation or multiplication of the coefficients corresponding to each action type.

Finally, a count of each potentially distinct *instance* of an action, e.g. in case of different IF – THEN conditions, can be made. In this case, also a choice is to be made among summation and multiplication approaches, or a mixed approach is to be used (e.g. multiplication along each possible processing path *and* summation over distinct path inside each process).

Table 8, below, summarizes the complexity evaluation levels, or *approximation orders*, proposed for the transactional base functional components.

Order	IFPUG	COSMIC
0	Size (extended)	Size (standard)
1	Primary action type	Primary manipulation type
2	# action types	# manipulation types
3	# action instances	# manipulation instances

Table 8: Summary of complexity evaluation orders for transactional BFCs’.

3.2 Data functions

Data Base Functional Components of the IFPUG measurement method are the logical files (Internal and External – ILF and EIF). The actual COSMIC method version does not measure the data component of software, still it consider this component in terms of data groups and attributes.

A digression about IFPUG limited values (data functions)

As for the transactional case, the IFPUG method shows upper-bounded ranges for files; similarly, extended experimental matrices and values are provided (Table 9, logical files). Low ranges are provided in order not to oversize trivial cases, such as reference tables made of just code and description fields. Highlighted sub-matrices show the original values from IFPUG.

RETs'	DETs'				
	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	ILF	EIF
VL	4	3
L	7	5
A	10	7
H	15	10
VH	25	15
VVH	45	25

Table 9: Extended IFPUG functional complexity matrix (left) and unadjusted function point values (right) for internal and external logical files.

From scale to diversity for data groups

Along with the current null size values for data groups in the COSMIC method, the main differences among IFPUG and COSMIC approaches are:

- the explicit classification of internal versus external files (maintained versus read-only data groups)
- the RET (Record Element Type) concept, whose definition and interpretation is subject of a large debate in the IFPUG community.

The first classification is already taken into account in the size assignation by the IFPUG method (although the relative scales of ILF versus EIF types may result not adequately correlated to a notion of complexity and/or of development difficulty). In the COSMIC approach, data groups could also be assigned a size

measure, and this quantity could be differentiated by the global access made on each file (written at least once versus read-only). A distinction is to be made between persistent and transient data groups (i.e. objects of interest), but by now it seems that this distinction would add poor, if any, additional complexity evaluation.

About the debated RET concept, a new concept, complementary to the File Type Referenced for processes (FTR), can be introduced for files: the Transaction Type References (TTR), in place of the RET. The key concept is that a file that is to be referenced by many different processes requires – on the average – much more analysis and design than a file accessed by a small number of processes. When constructing a specific metrics for such case, the trivial case of management of the same file by the CRUD set of 4 typical processes (Create, Read, Update and Delete) should be kept in mind to avoid over-evaluation of complexity. Explicit suggestions about a numerical scale for such characteristic element are still to be provided, due to lack of practical reference; some other trivial cases are to be adequately evaluated, such as for example a table referenced by many processes for simple retrieving purposes, or by many instances of identical sub-processes, such as field listing into several functional processes (this difficulty could lead to the simplest solution: avoid any RET/TTR consideration for files, looking for a size/effort correlation based purely on data attributes amounts, i.e. on DETs’).

Table 10, below, summarizes the complexity evaluation levels, or *approximation orders*, proposed for the data base functional components. More orders could be proposed, for COSMIC, exploiting information at the sub-process level (separate amounts of Entries, eXits, Reads, or Writes, instead of amount of processes).

Order	IFPUG	COSMIC
0	Size (extended)	Size (non-standard)
1	Primary reference type	Primary access type
2	# referring elementary processes types	# referring functional processes types
3	# referring elementary processes potential instances	# referring functional processes potential instances

Table 10: Summary of complexity evaluation orders for data BFCs’.

4 Complexity evaluation & metrics for the intermediate level (module)

At the intermediate level, we can consider each module as having an internal complexity corresponding to its functional structure. The complexity evaluation of the module functional structure, analogously to the previous level approach, can be approximated with a gradual move from *scale* to *diversity* considerations. A module can be a subsystem (a peer item), or the whole system if no specific software partition is identifiable. The correct identification of modules is outside the scope of this work; boundary identification between modules is obviously a crucial aspect of such analysis. Note that modules are intended to be components of the system at the same level of functional decomposition; consequently, if different layers are identified, each layer should be seen as a “per se” system (refer to next section). Tables 11 and 12, below, summarize the complexity approximation orders for such level, with suggested metrics, for transactional and data components, respectively. From order “2” up, other significant complexity indicators from [9] can be considered.

Order	IFPUG	COSMIC
0	# processes within module	# processes within module
1	# processes × avg. process cplx	# processes × avg. process cplx
2	% FTRs’ vs total # files in the module ± % processes DETs’ vs total # processes in the module	% R/Ws’ vs total # data groups in the module ± % E/Xs’ vs total # processes in the module

Table 11: Complexity evaluation orders for transactional component of each software system module. “avg process cplx” is the average complexity per transactional BFC, obtained (or estimated) from the previous level.

Order	IFPUG	COSMIC
0	# files within module	# processes within peer item
1	# files × avg. file cplx	# processes × avg. process cplx
2	% TTRs’ vs total # processes in the module ± % files DETs’ vs total # files in the module	% R/Ws’ vs total # processes in the peer item ± % E/Xs’ vs total # data groups in the peer item

Table 12: Complexity evaluation orders for data component of each software system module. “avg file cplx” is the average complexity per data BFC, obtained (or estimated) from the previous level; “TTR” stands for “Transaction Type Reference”.

5 Complexity evaluation & metrics for the higher level (whole system)

At the whole-system level, analogously to the previous level, we consider the system as having a internal complexity corresponding to its functional structure (note how intrinsic complexity of an item, when seen as a whole , can always be redefined as its structural complexity, when seen as decomposed into constituents parts). Table 13 below summarizes the complexity approximation orders for such level, with suggested metrics. Note that that orders are applicable to a whole system, or to each layers representing its functionality, as decomposed by the measurer (based on measurement purpose and viewpoint).

Order	IFPUG	COSMIC
0	# modules within system	# peer items within system (layer)
1	# modules × avg. module cplx	# peer items × avg. peer item cplx
2	% total DETs' vs total # files in the system (layer) ± % DETs' vs total # processes in the module	(avg % R/Ws' within peer items ± avg % E/Xs' between peer items) vs total # DMs' in the system

Table 13: Complexity evaluation orders for the whole system (or each one of its layers). “avg module/peer item cplx” is the average complexity per macro component, obtained (or estimated) from the previous level; “DM” stands for “Data Movement” (note that the total # DMs’ equals the size of the system, or layer, expressed in COSMIC functional size units, or FFP).

6 Distinct level complexity metrics combination

While higher approximation orders for intermediate and higher level of the complexity model are still to be refined in their metrics definition, we can provide a generic way to combine the complexity indicators that arise from the three levels described in the previous section.

Orders “0” and “1” of complexity approximation must be excluded from this combination, since they express a simple *scale* evaluation of the previous level (how many processes and files are within a module, how many modules are within the system): combining complexities evaluated on such orders would be equivalent to re-count twice or three times the same complexity factors, as viewed at different detail levels.

When complexity approximation orders “2” or higher are used at one or more level in the complexity model, we can average the complexities at each level upon its items (since each level is constituted by N parallel items), then multiply the

three levels complexity indicators (since from approximation order “2” up, they should be independent one of each other):

$$complexity_{level\ i} = \frac{1}{N} \sum_{j=1}^{N\ items\ of\ level\ i} complexity_{item\ j} \quad where \quad i = 1, 2, 3$$

(1, 2, and 3 denote respectively the BFC, module and whole-system levels) and:

$$complexity_{overall} = \prod_{i=1}^3 k_i \cdot complexity_{level\ i}$$

The presence of multipliers k_i means that for different environments (domains) one can customise the overall complexity formula, in order to weight distinct levels contributions per environment; such formula is therefore denoted as the *complexity mask*, or *complexity weighted profile*, for such environments. No specific values are provided by now for such weights for different environments; therefore their starting values are unitary.

7 Conclusion

Although no specific definition of software complexity was stated, this work proposed a self-consistent complexity evaluation model, based on a generic notion of complexity, to be correlated to the difficulty of realization. No assumptions were made about the process complexity, which is subject to a totally different analysis.

No assessed coefficients were provided by now for the proposed complexity model at any level; some values are suggested, based only on heuristic or historical approaches. Still, opposite to some past proposals (e.g., in [8]), it is suggested to fix some values regardless of the environment or domain of the software being assessed: the adaptation should occur in the weighting scheme of the overall model.

Next, the proposal hereby introduced will be subject of:

- validation through adequate experimentation on real cases;
- extension and/or update, based on the validation results and/or more suggestions for outer research (e.g. some criteria deriving from specific software architectures are to be analysed; processes sequencing consideration; etc.);
- comparison against other complexity evaluation approaches, examining eventual correlation with physical or technical parameters (such as code characteristics, truly algorithmic complexity, etc.).

References

1. Cardoso, A.I., Crespo, R.G., Kokol, P., Two different views about software complexity, 11th European Software Control and Metrics conference – 3rd Software Certification Programme in Europe Conference (ESCOM-SCOPE), Munich, Germany, 2000.
2. Codefast, 10 Complexity Drivers for Software Builds, White Paper. Online: <http://www.codefast.com/z-software-builds/sc-software-builds-tenq.html>
3. COSMIC, Full Function Point Measurement Manual 2.2, Common Software Measurement International Consortium, 2003.
4. DeMarco, T., Controlling Software Projects, Yourdon Press, New York, 1982.
5. IFPUG, Function Point Counting Practices Manual 4.2, International Function Point Users Group, 2004.
6. Ivanov, P.B., Qualitative Complexity, Troitsk Institute for Innovation and Fusion Research, Moscow, Russia, 1997. Online: <http://www.geocities.com/unihl/texts/cx/cx.htm>
7. Meli, R., The Software Measurement Role in a Complex Contractual Context, 1st Software Measurement European Forum (SMEF), Rome, Italy, 2004.
8. Nonaka M., Kakurai A., Bukhary E., Azuma M., A complexity-weighted functional size metric for interactive software, 12th International Conference on Software Quality (ICSQ), 2002.
9. Tran-Cao D., Abran A., Lévesque G., Functional complexity measurement, 11th International Workshop on Software Measurement (IWSM), Montréal, Canada, 2001.