A Framework for Software Project Estimation Based on COSMIC, DSM and Rework Characterization

Sharareh Afsharian
Computer Science Department
University of L'Aquila
Via Vetoio, Loc. Coppito
I-67100 L'Aquila, Italy
and
Ericsson R&D Italy
Via Anagnina 203
I-00118 Rome, Italy
sharareh.afsharian@ericsson.com

Marco Giacomobono
Ericsson R&D Italy
Via Anagnina 203
I-00118 Rome, Italy
marco.giacomobono@ericsson.com

Paola Inverardi
Computer Science Department
University of L'Aquila
Via Vetoio, Loc. Coppito
I-67100 L'Aquila, Italy
inverard@di.univaq.it

ABSTRACT

Effective software project estimation is one of the most challenging activities in software development. In today's highly competitive world, accurate software estimation can make the difference between successful projects and dismal failures. Proper project planning and control is not possible without a sound and reliable estimate.
In this paper we propose a framework, developed by Ericsson R&D Italy, for project time and cost estimation for software development projects in the telecommunications domain. The customization of Design Structure Matrix (DSM), the application of COSMIC and the study of defect complexity curves are the components of this new estimation framework. The joint application of these three components allows all stakeholders interested in the estimation result to have a common view based on objective data and to understand how a change to functional and quality requirements can impact the result.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*complexity measures, performance measures*; D.2.9 [**Management**]: Cost estimation

General Terms

Management, Measurement, Economics, Experimentation

Keywords

COSMIC, DSM, estimation, defect analysis

BIPI'08, May 13, 2008, Leipzig, Germany.

1. INTRODUCTION

How well do most people estimate? This is a crucial question. The software industry's estimation track record provides some interesting clues to the nature of the problems associated with estimating software projects. In recent years The Standish Group [4] has published biennial surveys called *The Chaos Report*, which describe the outcome of software projects. Interestingly the survey published in 2004 reports that 54% of projects were delivered late, only 18% failed outright, and 28% were delivered on time and within budget.
The reasons that projects miss their targets are manifold, but it is certain that one of the most important reasons is due to a failure in project estimation. Keeping this in mind and considering the major estimation methodologies (e.g. delphi process [24, 26, 11], statistical sizing [19], analogy [25], algorithmic models [22, 23], function point [9], object points [10], or use case point [20, 16]) we can say that project estimation has still a long way to go and there is not an univocal well-known way of working or methodology recognized from all scientific and business communities. This is due to the fact that each of these methodologies was developed for a particular domain and each has its own strengths and weaknesses. So how should the appropriate methodology be chosen for a particular project? The current thinking, with which many practitioners strongly concur, is that you should use at least three different methods [21]. Each provides some insight and understanding, which can then be combined to derive an answer.
Taking this into consideration, Ericsson R&D Italy set about to define and deploy a generic framework for software project schedule and cost estimation, referred to as the *Ericsson Project Estimation Analyzer*. The following requirements were established at the outset:

1. to have a formal and rigorous method capable of measuring functional requirements. This is based on the strong belief that the measurements of any entity must be obtained by means of objective valuation and not from subjective criteria, as stated by Tom De Marco: "you cannot control what you cannot measure" [15]. Such a method should be able to demonstrate high

level of precision and repeatability, be easy to understand and apply in the selected domain be inexpensive to not require changes to already existing development processes.
2. to have a method able to estimate possible rework during the lifecycle of the product.
3. to have a tool that takes as input the results of the two above methods and that can simulate different estimation scenarios when the project characteristics (e.g. requirements and quality factors) are changed.

To fulfill these requirements, the following solutions were chosen:

1. the use of *COSMIC* to estimate the functional requirements of a product. COSMIC was developed by the Common Software Measurement International Consortium (COSMIC) and in 2003 was adopted as the ISO IEC 19761 standard. The approach is to analyze the system not only from the user's point of view but also as a white box, estimating the impact of requirements on all system components [18, 8, 2].
2. The classification of projects in terms of complexity classes and performance trends (i.e. defect curves) for each class, by analyzing the historical data of previous projects executed at Ericsson.
3. The use of a *design structure matrix* (DSM) [13, 27] to analyze and simulate different scenarios taking as input both functional estimations and defect sizes. DSM is based on a square matrix that captures dependencies between system components. These components can include, but are not limited to, product parts, teams, processes, or activities. When effort estimation, learning curve, rework probability and percentage of impact are provided as input, DSM is able to simulate the whole system and to find the most likely behaviour.

These three components are the basis for the proposed framework. This paper shows the process needed to develop such a framework, and the motivation that guided the selection of the chosen techniques. The experiments on significant projects within Ericsson R&D Italy validated the innovation of this framework in terms of efficacy and precision.
This paper is organized into the following sections: Section 2 describes the motivation behind choosing COSMIC as a suitable FSM method for use in the telecommunications domain. Section 3 provides a brief description of how Ericsson R&D Italy defines the term *rework* and why the authors considered it so important to measure. Section 4 describes the reasons drove the choice of DSM as a performance management tool. Section 5 introduces the defined framework and the process of deployment. Section 6 reports on the experiments performed and COSMIC performances. Section 7 states the authors' conclusions.

2. WHY COSMIC?

Functional size measurement (FSM) methods [1] are intended to measure the size of software by quantifying the *functional user requirements*. IFPUG FPA and COSMIC are the main methods of FSM: they are already international standards and are widely used in practice. COSMIC represents the second generation of FSM [17]. They are applicable not only for the development of business application software but also for the development of real-time, embedded and telecoms software [6]. The introduction of the concept of a *measurement viewpoint*, that allows the analysis of software at different levels, makes the measurement of the software more suitable and realistic. It captures the size of each layer of multi-layered software architecture as well as the size of each separate component within each layer. Such size measure should prove useful for understanding development performance and for estimating purposes. The first part of the analysis undertaken was to identify the best FSM method for the telecommunications domain. Both FPA and COSMIC had previously been applied on several projects in order to evaluate effort estimation and the relationship between the number of function points and man-hours. This paper shows how the applied measurement procedure of COSMIC provides very good estimates close to the actual effort spent in each project, and highlights the reasons why COSMIC outperforms FPA in the telecommunication domain. Moreover the paper describes how the COSMIC methodology adequately fulfills the requirements stated above and also captures non functional requirements (e.g. scalability, memory usage, processor load).

3. THE PROBLEM OF REWORK

Even with a good estimation model for functional size measurement, there are a lot of obstacles during the product development phase that cause deviations. In our context the experience has shown that *rework* is the main cause of deviations. Therefore Ericsson historical data was analysed in order to characterize this rework. The resulting model is based on the relation:

$$rework = defectiveness$$

This assumption is justified by the fact that our analysis highlighted that defects discovered during the development phase are the main cause of rework. The analysis also highlighted two important factors: firstly, it was not sufficient to simply incorporate a buffer during project estimation, the buffer in more cases was or too large or not enough compromising the estimation made. Secondly the cost of fixing the defects found in different verification phase, has a significant difference in terms of effort required. More precisely, four complexity classes were defining for our projects. Each class is characterized by technologies used (i.e. hardware, software, firmware), size of development, number of test case planned to verify the product, number of internal and external project dependencies, development languages used for coding, etc. According to this classification it was found that elements in the same class perform in the same way. From this it is possible to define *performance curves* that characterize each class in terms of time, cost and quality. In the quality dimension an analysis was performed on the number of defects discovered during the development phase. Trends were found connected both to the number of defects per test phase (e.g. desk check, basic test and function test) and to the number of faults *slipping through*. Slipping through evaluates the effectiveness of the verification phases by analyzing the defects slipped from early verification phases to later ones. As the number of hours required to fix a fault increase the later the fault is found, it is possible to claim that slipping through is the main cause of rework. Moreover it was discovered that this increases with complexity. This implies therefore that it is important to have a *rework model* and to be aware as to how the defectiveness (slip through) is a very interesting characterization of rework.

4. DSM - A TOOL FOR PERFORMANCE MANAGEMENT

Products, processes, and organizations are complex systems which pose a challenge to plan the manage. However one method is starting to be adopted for representing and analyzing complex system architectures. This method, the design structure matrix (DSM) [13, 27] is helping researchers and practitioners plan and manage product architectures, organizational structures and process flows. It is considered a system modeling tool. It has two main strengths. Firstly, it can represent a large number of system elements and their relationships in a compact way that highlights important patterns in the data (such as feedback loops and modules). Secondly, it can be amenable to matrix-based analysis techniques, which can be used to improve the structure of the system [7].

DSM is based on a square matrix that documents dependencies between system components. These components can be product parts, teams, processes, activities, or other things. From simple analysis, it is possible to prescribe a modular system architecture or organization structure. Adding a time-basis enables a faster, lower-risk process to be prescribed. Because DSM highlights process feedbacks, it helps identifying iteration and rework loops which are key drivers of cost and schedule risk. DSM can also show how delays in external inputs, such as requirements and equipment can directly lead to increased cost, schedule, and risk.

DSM is concise and visually appealing and it is in use in a number of industries, companies, and agencies (it is also known as the dependency structure matrix and the dependency source matrix [3]).

Consider a system (or project) that is composed of two elements /sub-systems (or activities/phases): Element A and Element B. A graph may be developed to represent this system. The graph is constructed by allowing a node on the graph to represent a system element and an edge joining two nodes to represent the relationship between two system elements. The directionality of influence from one element to another is captured by an arrow instead of a simple link. The resultant graph is called a directed graph or simply a digraph. There are three basic building blocks for describing the relationship amongst system elements: parallel (or concurrent), sequential (or dependent) and coupled (or interdependent).

The matrix representation of a digraph is a binary and square matrix with m rows and columns, and n non-zero elements, where m is the number of nodes and n is the number of edges in the digraph. The matrix layout is as follows: the system elements names are placed down the side of the matrix as row headings and across the top as column headings in the same order. If there exists an edge from node i to node j, then the value of element i, j is marked 1 (or black). Otherwise, the value of the element is zero (or left empty).

This paper shows that the characterization made on DSM provides a valid tool to simulate the wanted estimation and improves visibility and understanding of project/system complexity for every stakeholder of the project estimation.

5. THE FRAMEWORK

This work has been driven by the following considerations and questions:

1. Is it possible to have a method that, taking requirements as input can measure in a formal and rigorous way the respective effort in our domain?
2. If yes, what kind of requisite should this method have in order to be attractive and easy to insert in our context? The minimum requirement should be performant in terms of precision and repeatability, ease of application in our domain, easy to understand, inexpensive to introduce and not require changes to existing development processes.
3. Would it be sufficient to have a method that only covers the functional part of software and not all aspects (i.e. defectiveness, dependencies, non functional requirements) [14]?
4. Is it possible to have a tool that can in an easy way simulate the planning scenarios changing the sensible factors of a project, like requirements, times, and quality?

By answering these questions a definition of a framework based on three components was derived:

- *COSMIC* for function requirements estimation (as an answer to the first and second questions / requirements).
- *Performance curves* for rework analysis and rework estimation (as an answer to the third question).
- *Design structure matrix* (DSM) as a model for simulating the complete system (as an answer to the fourth question as a performance management tool).

The steps required to estimate a new development by means of this framework can be summarized as follows:

1. Evaluate the functional size using COSMIC.
2. Translate the COSMIC size in effort (MHrs) using the multiply factor as described in Section 5.2.
3. Define the work breakdown structure for the new development listing all activities and dependencies among them.
4. Evaluate the man-hours of each activity (min, max and most likely) using Table 4.
5. Determine the complexity class of the new development and thereby the performance curves.
6. Use the performance curves to characterize the rework index.
7. Simulate the system using DSM. In this way DSM, through hundred iteration of the whole system, is able to find the most probable behaviour that characterize the project estimation.

The next sections describe the roadmap that drive us to the definition of the above steps[1].

5.1 Description of the application domain

Before showing the customization of these methods to our domain let us provide a very short sketch of our system and products.

Ericsson products are mostly characterized by a layered architecture structured in software, firmware and hardware components. The communication among different layers and components is based on signaling exchanges. A signal is an event triggered between two different processes that carries data to be computed.

When a customer request new features in the form of re-

[1]Note: For reason of confidentiality figures have been omitted from pictures and tables that describe historical performance of projects. Only the trend is shown.

Table 1: Complexity characterization table.

Compl. Class	SW	SW/FW	SW/FW /HW	FW	HW	FW/HW	Multi-site (Design)	Partners	Sub-projects	Exec. Length (months)
C1	≤2 blocks	≤2 blocks		≤2 blocks	1 board DfE		one-roof	none	none	≤4
C1		≤2 SW + 2 FW				1 board + 1 block				
C2	≤5 blocks	≤5 blocks	≤2 blocks + 1 board	≤5 blocks	2 boards	1 board 2 blocks	2 sites	none	none	≤6
C3	6≤blocks<10	≤3 SW + 3 FW	≤2 SW + 2 FW + 2 board			1 board + >2 blocks	3 sites	1	1	≤8
C4	≥blocks	≥7 blocks	≥5 blocks + n board	≥7 blocks	≥3 boards	>1 board + >2 blocks	> sites	≥2	≥2	>8

quirements it is possible to identify the system that is impacted and to distribute the new features on the impacted subsystems, their relative components (software, firmware or hardware), down to identification of individual software unit, firmware unit, or hardware unit.
In general, the identification of the right system and its subsystem is performed during the pre-study phase, where requirements are collected and accepted and an impact analysis made.
When impacts in the system are determined and dependencies with other system discovered, then during the feasibility stage the inter-work between different components in different subsystems are all analyzed.
In this context both FPA and COSMIC were applied to several projects in order to observe and analyze their behaviours in terms of precision and performance. The reason for applying both was to validate the choice of COSMIC, in order to see if the requirements to determine a functional size measurement was fully satisfied.

5.2 Evaluate functional size using COSMIC

The measurement procedure defined in order to apply COSMIC in our domain needs to be clarified.

- The scope of the counting is the total subsystems scope.
- The boundary and the function point of view definition come natural and immediately: the boundary and the user point of view can be dynamically moved from one component to another.
- Every signal is considered as a functional process where the number of data movements is equivalent to the number of parameter it contains.
- The four data movements are grouped into a couple of data movements: ENTRY-WRITE and EXIT-READ. This is because there is a correlation between each entry signal and the writing of its parameters in the component system.
- In COMIC the number of the parameters in a data movement represents its weight.

The counting with COSMIC was found to be very straightforward, without the need to adapt the method to our system and the counting was found to be systematic without any misleading.
Naturally a large number of components and signals causes the counting process to become time consuming. Howaver, this is possible to automate using as input the implementation proposal document (written during feasibility stage of a project) where interwork among different components of the impacted subsystems are described and well documented, also in terms of data exchanges (i.e. the parameters in the signals).
So from customer requirements, using our standard process to map functional requirements onto systems, subsystem, component units, it was possible to make a rigorous and formal functional estimation. In this way the counting is:

- independent from the person who counts because connected to measurable entities (signals and related parameters);
- not ambiguous, we counts all signals between componets (clear level of application).

The application of this method to several past projects resulted in a very good performance. Using the man-hours for past projects and applying COSMIC it was found a multiply factor that allow to translate COSMIC size in effort (man-hours), see Table 10. It has been noted that the *unit cost ratio*[2] remains constant with an average error of 5% as showed in Section 7.
Therefore it is possible to conclude that COSMIC is the answer to the first and second questions in Section 5.

5.3 Rework characterization

The process defined to characterized the rework concept is as follows:

1. A list of project characteristics that can better capture the notions of size/complexity was defined and the raw data (related to these characteristics) was collected concerning the projects involved in the analysis process.
2. Four complexity classes were defined: *initial* (C1), *intermediate* (C2), *adequate* (C3), *advanced* (C4) in order to classify all our development projects. Each class has specific characteristics in terms of technologies (software, firmware, hardware) used, size of development, number of test cases planned to verify the product, number of internal and external project dependencies, development languages used for coding, etc, as shown in Figure 1.
3. We found, according to this classification similar performance for all projects in each class.
4. Rework was defined on the basis of complexity and performance classes.

According to the previous classification similar performances of all elements in each class was found. This allowed the def-

[2]The unit cost ratio is also referred as *project delivery rate* in the ISBSG repository [5]

inition of performances curves that characterize each class in all three dimensions: time cost and quality. Studies were focused mainly in the quality dimension analyzing the defects discovered during development. Trends were found connected to the number of defects for the test phase (e.g. basic test and function test) and also to the slip through. The *slip through* evaluates the effectiveness of the verification phases by analyzing the defects slipped from early verification phases to later ones.
Figure 1 shows the slip through characterization. More precisely the histogram shows the number of defects found in each verification phase: review and inspection (R&I), code desk check (DC), basic test (BT), module test (MT), function test (FT), and system test (ST) . Figure 1 also presents the ideal defect trend in a project (the blue line), a real defect trend (the green line) and the percentage of faults slipped in each phase (the black line). Note that in a ideal trend the percentage of faults should be zero.
It can be observed that a large number of faults sleep in advanced test phases. The distance between the two curves (blue and green) represents how the project deviates from the expected behaviour. The bigger the distance the more is the rework in the project activities.
If it is considered that the hours spent to fix a fault increases the later the fault is discovered, then it is possible to state that this is the main cause of rework, moreover it was found to increase with increasing the complexity of the project (from C1 to C4). This relation is evident looking at Table 3 where it is reported the effort needed to fix a defect in different implementation phases for each complexity class. Thanks to different time reporting of designer in fixing defect it was possible to collect these data.
Therefore defectiveness and slip through are the best characterizations of the rework.
In conclusion the answer to the third question in Section 5 (*It would be enough to have a method that covers only the functional part of software and not the whole aspects?*) is that it would not be enough and that rework has a crucial role in project estimation and it is important to have a tool that allows this to be included in the project estimation.
It is now possible to define three other important factors that are very useful when building an estimation framework:

- The rework probability for each complexity class per developing phase. See Table 2.
- The percentage of impact (in term of effort - manhours) for each complexity class per developing phase. See Table 2 and Table 3
- The percentage of activity phase design, coding, basic test, module test, function test, system test). See Table 4.

In all of the above mentioned tables, the data reported in class C1 forms the basis of the evaluation for the other complexity class and it is expressed as a range that is subject to a percentage of increase based on the complexity class. It is important to remark that these tables capture the performance trend of Ericsson R&D Italy projects. As a consequence, in order to be used in another context, they have to be rebuilt to capture the complexity and trend characterizing that context.
The next section introduces the DSM tool that uses the characterization of rework and the functional size measurements made by means of COSMIC to simulate a complete project estimation.

Table 6: DSM estimation table example.

Name	Best Case	Most Likely	Worst Case
D_HW_1run	46	49	51
BT_1run	70	73	76
D_HW_2run	57	60	63
BT_2run	47	49	51
D_FPGA	142	150	158
D_FW	109	115	121
BJT	78	82	86
D_Block	160	167	174
MT	41	43	45
FT_P	189	197	205
FT_E	59	62	65

5.4 Simulation with DSM

The DSM method adapted in the Ericsson project context is the *Task-Based DSM Simulation* (called *DSM Sim*) [12]. As input, DSM-Sim requires the following data:

1. A DSM matrix with activities and dependencies. A DSM binary matrix was characterized by setting rows and columns with the tasks used for the development of a feature in a project (design, coding, basic test, module test, function test, and system test) reporting the dependencies among different tasks. See an example in Table 5 where the possible activities for implementing a feature x are reported with their dependencies.
2. For each activity, three cost estimates are required: an optimistic or *best case value* (BCV), a *most likely value* (MLV), and a pessimistic or *worst case value* (WCV). The effort of a feature is calculated by COSMIC. Using Table 4 we estimate the effort of the the single activities as the percentage of the total future cost. The range expressed in Table 4 gives the three values of the estimation table (BCV, MLV, WCV). The most likely value is the average of the best and worst case in the ranges expressed in Table 4 For example the estimation table can be expressed in man-hours as in Table 6. It is then possible to translate the effort in duration for time simulation.
3. Each activity also has an associated improvement curve, which represents the learning factor. This is based on the assumption that repeating the same activity a second or successive number of times costs less time/effort. The improvement curve is given as a percentage, the percentage of the original effort required to regenerate the activity (e.g. it takes X% of the original effort to repeat activity a second and successive times). For instance as shown in Table 7 D_HW_1run will take 25% effort less compared with the first time the activity was executed.
4. In addition, for each activity, the model requires an assessment on the probability that a rework occurs and the percentage of impact caused by that rework. Using Table 2 and Table 3 we can define three DSM matrices for the probability of rework: the best case, the most likely case and the worst case, see Table 8[3]). Moreover we can define three DSM matrices also for the percentage of impact (best case, most likely case and worst case, see Table 8).

[3]Due to size limitations, only the worst case example is reported.

Figure 1: Slip through characterization.

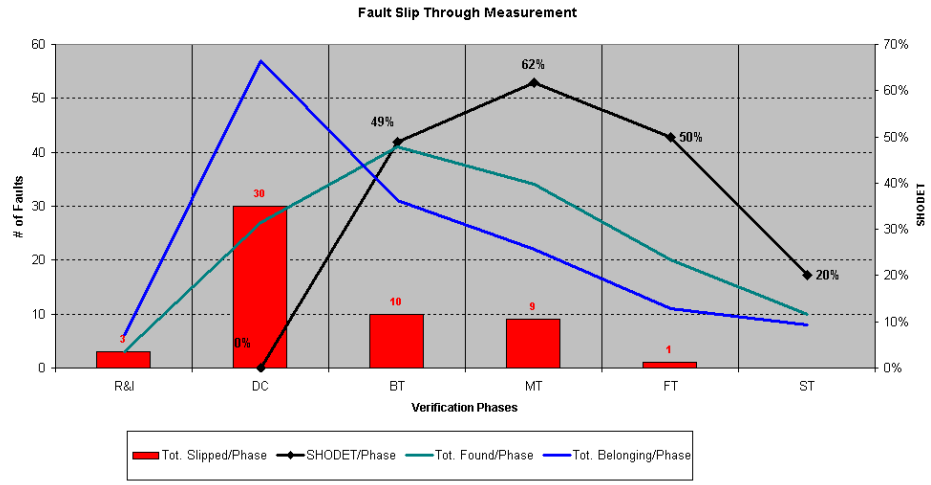


Table 2: Probability of the rework per complexity class.

Probability of rework	C1	C2	C3	C4
Design	X1, X2	X1+10%, X2+13%	X1+20%, X2+23%	X1+25%, X2+28%
Coding	Y1, Y2	Y1+10%, Y2+13%	Y1+20%, Y2+23%	Y1+25%, Y2+28%
Basic Test	Z1, Z2	Z1+2%, Z2+3%	Z1+6%, Z2+9%	Z1+9%, Z2+11%
Module Test	W1, W2	W1+2%, W2+3%	W1+6%, W2+9%	W1+9%, W2+11%
Function Test	V1, V2	V1+2%, V2+3%	V1+6%, V2+9%	V1+9%, V2+11%
System Test	T1, T2	T1+2%, T2+3%	T1+6%, T2+9%	T1+9%, T2+11%

Table 3: Percentage of impacts per complexity class.

Percentage of impact	C1	C2	C3	C4
Design	G1, G2	G1+10%, G2+13%	G1+20%, G2+23%	G1+25%, G2+28%
Coding	H1, H2	H1+10%, H2+13%	H1+20%, H2+23%	H1+25%, H2+28%
Basic Test	L1, L2	L1+2%, L2+3%	L1+6%, L2+9%	L1+9%, L2+11%
Module Test	K1, K2	K1+2%, K2+3%	K1+6%, K2+9%	K1+9%, K2+11%
Function Test	J1, J2	J1+2%, J2+3%	J1+6%, J2+9%	J1+9%, J2+11%
System Test	I1, I2	I1+2%, I2+3%	I1+6%, I2+9%	I1+9%, I2+11%

Table 4: Percentage of implementation phase per complexity class.

Percentage of the implement. phase	C1	C2	C3	C4
Design	P1, P2	P1+10%, P2+13%	P1+20%, P2+23%	P1+25%, P2+28%
Coding	Q1, Q2	Q1+10%, Q2+13%	Q1+20%, Q2+23%	Q1+25%, Q2+28%
Basic Test	R1, R2	R1+2%, R2+3%	R1+6%, R2+9%	R1+9%, R2+11%
Module Test	M1, M2	M1+2%, M2+3%	M1+6%, M2+9%	M1+9%, M2+11%
Function Test	N1, N2	N1+2%, N2+3%	N1+6%, N2+9%	N1+9%, N2+11%
System Test	O1, O2	O1+2%, O2+3%	O1+6%, O2+9%	O1+9%, O2+11%

Table 5: DSM matrix for a feature X example.

Activity Name		1	2	3	4	5	6	7	8	9	10	11
D_HW_1run	1											
BT_1run	2	1										
D_HW_2run	3		1		1			1				
BT_2run	4			1								
D_FPGA	5							1				1
D_FW	6							1				1
BJT	7		1			1	1					
D_Block	8									1		
MT	9								1			
FT_P	10											
FT_E	11							1		1	1	

Table 8: DSM probability of the rework (worst case).

Activity Name		1	2	3	4	5	6	7	8	9	10	11
D_HW_1run	1											
BT_1run	2	0,22										
D_HW_2run	3		0,22		0,70			0,22				
BT_2run	4			1,00								
D_FPGA	5							0,22				0,22
D_FW	6							0,22				0,22
BJT	7		1,00			1,00	1,00					
D_Block	8									0,70		
MT	9								1,00			
FT_P	10											
FT_E	11							0,22		0,22	0,22	

Table 9: DSM percentage of impact (worst case).

Activity Name		1	2	3	4	5	6	7	8	9	10	11
D_HW_1run	1											
BT_1run	2	0,11										
D_HW_2run	3		0,11		0,25			0,11				
BT_2run	4			0,25								
D_FPGA	5							0,11				0,11
D_FW	6							0,11				0,11
BJT	7		0,11			0,11	0,11					
D_Block	8									0,25		
MT	9								0,25			
FT_P	10											
FT_E	11							0,11		0,11	0,11	

Table 7: DSM learning curve example.

Name	LC
D_HW_1run	0,25
BT_1run	0,16
D_HW_2run	0,25
BT_2run	0,16
D_FPGA	0,2
D_FW	0,2
BJT	0,16
D_Block	0,18
MT	0,2
FT_P	0,15
FT_E	0,2

The model also uses three other vectors, each with a length equal to the number of activities. First, a sequencing vector specifies the order of the activities in DSM. Second, a work vector, W, keeps track of the amount of work remaining to be done on each activity. Usually, each entry in this vector is set to 100% to begin each simulation run. Third, a *work now* vector of boolean entries indicates whether an activity has been started.

The simulation uses a simple, time advancing approach. Each run consists of a series of equal time steps, Δt, the size of which is smaller than the duration of the shortest activity (e.g. if activities have durations ranging from five to 50 weeks, a reasonable Δt could be 0.5 weeks, as activity durations are rounded off to an integer number of time steps. Smaller time steps provide greater model resolution at the expense of greater simulation execution time).

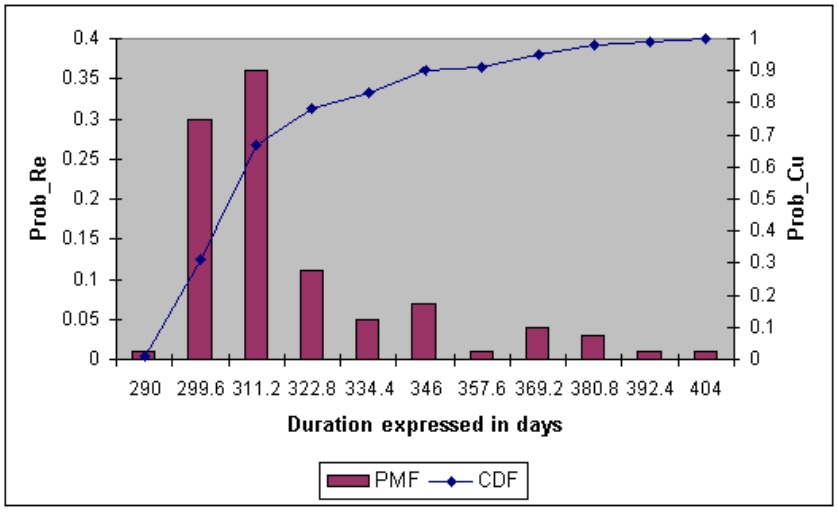
During each time step, the model checks for the upstream most activity requiring work and any activities that can be executed concurrently. Activities do not begin work until their required inputs are available from completed, upstream activities. Work is done on available activities during the time step and their work remaining is reduced by the fraction of their duration represented by the time step. The cumulative process cost is increased based on the cost of this work. Whenever an activity finishes, the model checks for potential iterations (rework for upstream activities) and second-order rework resulting from iterations using the probabilities in DSM. If rework occurs, its amount – a percentage of the activity given in DSM and modified by improvement curve effects – is added to the W vector. When all activities are complete, all W vector entries equal zero. The model converts the number of time steps required into appropriate units and outputs this as the process duration or schedule, S, for the run.

DSM as characterized above, executes a simulation of a complete system with hundreds iterations. In each iteration the DSM algorithm selects random values in the ranges indicated in the DSM matrices. The result is a Gaussian curve representing the time duration distribution called the *probability mass function* and a *cumulative distribution function* representing the probability of the distribution.

By combining these two results it is possible to determine find the most probable duration for the feature development. See an example of simulation of a worst case in Figure 2: here the probability mass function indicates that the duration most probable in the worst case is 311 days. The cumulative distribution functions show the most probable values are between 70% and 80%, about 311 and 322 days in the worst case.

DSM can be used as a live tool during the entire project, showing how changes can cause deviation in the original plans. Most software projects still tend to run late because of arbitrary estimate overruling by customers and senior executives, creeping requirements, and inadequate early quality control. A famous phrase from an Agile project management

Figure 2: Probability mass and cumulative distribution functions for worst case.



is "Maybe it is not our estimating skills that need upgrading, but our negotiating skills". In software, we can negotiate the completion date and the development resources, we can negotiate functionality, or we can cancel the project. And such negotiation is often an ongoing process. So it is very important to have a tool that in an easy and effective can demonstrate to all stakeholders in the project, how the project is impacted by adding or removing requirements and activities. DSM is therefore considered a very powerful tool.

6. EXPERIMENTS RESULTS

In order to test the framework ten projects were selected from an historical database. For each project we simulated to be in the planning phase where we usually identify the technical impacts and describe them in a document called *implementation proposal* (IP). We used as input for the estimation method defined in Section 5, the IP of the project used actually at its planning phase time. It is important to take into account the fact that the IP captures both functional and not functional requirements describing their impacted on the architecture of the system[4]. The validation process applied to each project is described in the following steps:

1. Evaluate functional size applying the COSMIC procedure method on the architecture described in the IP, using only the project data that were available in the initial project phase.
2. Translate the COSMIC size in effort (man-hours) using the multiply factor as described in Section 5.2;
3. Define the work breakdown structure for the new development listing all activities and dependencies among them.
4. Evaluate the man-hours of each activity (min, max and most likely) using Table 4.
5. Find the complexity class of the new development and according to it the performance curves.
6. Use performance curves to characterize the rework index.
7. Simulate the system using DSM. In this way DSM, through hundreds iteration of the system, is able to find the most probable behaviour that characterize the project estimation.
8. Compare the DSM results with the final project data (time and budget) available in the historical database.

Applying the framework to these projects we had very encouraging results. DSM performs constantly around 98% of precision.

The contribution of COSMIC to such results is very significant, as shown in Table 10 which contains the number of COSMIC and FPA per man-hours. It is evident that the trend of COSMICs is stable with respect to the size of the projects. Furthermore the maximum distance error for COSMIC is about 2%. It is also interesting to note that FPAs have a non-linear trend. It is also very interesting that using the IP as input for counting COSMICs both functional and not functional requirements are captured. In that sense the estimation method can be considered also more reliable in our domain.

7. CONCLUSIONS

As described in the introduction, software estimation is a strategic activity in order to be competitive in the market respecting planned time and costs. It follows therefore that the business impact of having an accurate framework for software project estimation is essential for software company.

In this paper a new framework for estimation project size based on COSMIC, DSM and rework characterization has been proposed. In addition to the good performance obtained and the very good feedback that can be given to the COSMIC community, the main contribution of this work consists in the approach used to build the framework. In fact it can be easily generalized to be used in other contexts, simply by customizing the questions that drove the definition of the framework described. Another important outcome of this work is the identification of rework as a cru-

[4]The non-functional requirements in our domain (e.g. scalability, processor load, memory usage.) are constant.

Table 10: Customized COSMIC and FP method application results.

Size of project in Mhrs	FP	COSMIC	COSMIC in Mhrs	FP in Mhrs	COSMIC/FP
5783	239	626	9,238	24,197	2,619
7850	291	866	9,065	26,976	2,976
6600	358	719	9,179	18,436	2,008
10480	433	1168	8,973	24,203	2,697
15484	865	1727	8,966	17,900	1,996
13700	668	1499	9,139	20,509	2,244
14740	590	1578	9,341	24,983	2,674
3200	200	347	9,222	16	1,735
4000	180	434	9,216	22,222	2,411
7000	311	768	9,114	22,508	2,469

cial factor in a project estimation process. This is because rework due to defectiveness causes a chain reaction in the projects, making significant deviation in terms of time, budget and quality. Furthermore since rework often requires a task force or puts high pressure on the organizations, motivation and productivity can be impacted. In that sense we conclude that estimation without taking into account this factor cannot be considered reliable. The model proposed to estimate rework performs very well in our domain and it significantly contributes to the good results obtained during our experiments.

8. REFERENCES

[1] http://www.geocities.com/lbu_measure/fpa/fpa.htm.

[2] Advantages of the COSMIC method. http://www.cosmicon.com/advantagecs.asp.

[3] The Design Structure Matrix web site. http://www.dsmweb.org/.

[4] Standish Group web site. http://www.standishgroup.com/.

[5] Web site of the International Software Benchmarking Standards Group (ISBSG). http://www.isbsg.org/.

[6] Focus on Charles Symons, founder of COSMIC and creator of the Mark II function point. http://www.compaid.com/, September 2006. A CAI State of the Practice Interview.

[7] Definition of Design Structure Matrix published on Wikipedia. http://en.wikipedia.org/, 2008.

[8] A. Abran, C. Symons, and S. Oligny. An overview of the COSMIC FFP field trial results. In *proceedings of the ESCOM Conference '01*, London, UK, 2001.

[9] A. J. Albrecht. Measuring application development productivity. In I. Press, editor, *proceedings of IBM Application Development Symposium*, October 1979.

[10] R. D. Banker, R. J. Kauffman, C. Wright, and D. Zweig. Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment. *IEEE Trans. Softw. Eng.*, 20(3):169–187, 1994.

[11] B. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981.

[12] T. Browning and S. Eppinger. Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Transactions on Engineering Management*, 49(4):428–442, 2002.

[13] T. R. Browning. Use of dependency structure matrices for product development cycle time reduction. In *Proceedings of the Fifth ISPE International Conference on Concurrent Engineering: Research and Applications*, pages pp. 89–96, Tokyo, Japan, July 15-17 1998.

[14] L. Buglione. *Project Size Unit (PSU) - Measurement Manual, v1.20*, August 2007.

[15] L. Buglione. *Misurare il Software. Quantitá, qualitá, standard e miglioramento di processo nell'Information and Communication Technology (3rd edition)*. Franco Angeli, January 2008.

[16] R. K. Clemmons. Project estimation with use case points. *CROSSTALK - The Journal of Defense Software Engineering*, February 2006.

[17] Common Software Measurement International Consortium. *Measurement Manual COSMIC Full Function Points 2.2 - The COSMIC Implementaion Guide for ISO/IEC 19761*, 2003.

[18] Common Software Measurement International Consortium. *Measurement Manual - The COSMIC Functional Size Measurement Method Version 3.0*, 2007.

[19] D. D. Galorath and M. W. Evans. *Software Sizing, Estimation, and Risk Management.* Auerbach Publications, Boston, MA, USA, 2006.

[20] G. Karner. Metrics for objectory. Master Thesis, 1993. Linkuping University.

[21] L. M. Laird. The limitations of estimation. *IT Professional*, 8(6):40–45, 2006.

[22] T. McGibbon. Modern empirical cost and schedule estimation tools - a DACS state-of-the-art report, 1997.

[23] C. R. Pandian. *Software Metrics: A Guide to Planning, Analysis, and Application.* AUERBACH, September 2003.

[24] Rowe and Wright. *Expert Opinions in Forecasting - Role of the Delphi Technique*, 2001.

[25] M. Shepperd, C. Schofield, and B. Kitchenham. Effort estimation using analogy. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 170–178, Washington, DC, USA, 1996. IEEE Computer Society.

[26] M. Turoff and H. Linstone. *The Delphi Method: Techniques and Applications.* Addison-Wesley, Reading, MA, 1975.

[27] A. Yassine. An introduction to modeling and analyzing complex product development processes using the design structure matrix (DSM) method. Quaderni di Management (Italian Management Review), 2004. n.9, www.quaderni-di-management.it.