Computing and Informatics, Vol. 32, 2013, 1147–1169

A STATISTICAL APPROACH FOR THE MAXIMIZATION OF THE FINANCIAL BENEFITS YIELDED BY A LARGE SET OF MMFS AND AES

Antonio J. Alencar*, Carlos A. S. Franco
Eber A. Schmitz

Tércio Pacitti Intitute for Computer Research and Application Development
Federal University of Rio de Janeiro (UFRJ)
Av. Athos da Silveira Ramos 274
21941-916 Rio de Janeiro, RJ, Brazil
e-mail: juarezalencar@br.inter.net, carlosfranco@dcc.ufrj.br,
eber@nce.ufrj.br

Alexandre L. Correa

Department of Applied Informatics
Federal University of the State of Rio de Janeiro (UNIRIO)
Av. Pasteur 458
22290-240 Rio de Janeiro, RJ, Brazil
e-mail: alexcorr@yahoo.com

Abstract. This article introduces a statistical approach for the maximization of the financial benefits yielded by software projects that have been broken down into a considerable number of minimum marketable features modules (MMFs) and architectural elements (AEs). As the statistical approach requires a polynomial computational effort to run and provides approximation solutions with an arbitrarily chosen degree of confidence, it allows managers and developers to be more confident about the rightness of the decisions they make with little additional computational effort.

* corresponding author

Keywords: Value-based software engineering, incremental funding method, minimum marketable features, scheduling algorithms, software project appraisal

Mathematics Subject Classification 2010: 68N30

1 INTRODUCTION

Ever since the first commercial electronic computers were introduced into the corporate market in the early 1950s, the total cost of computer-related hardware has been decreasing at an astonishing pace [1].

For instance, in the second half of the 20$^{\text{th}}$ century computers were so expensive that only government agencies and large corporations had enough investment capital to acquire or lease them. However, nowadays consumers from various different social strata can actually buy machines that are much more powerful than those early commercial computers. In reality, they may select their favorite model in the displays of department and specialised stores, place them in their shopping carts and pay for them at the checkout with their personal credit cards [2].

Moreover, when computers are bought these days, it is often the case that, for a fraction of the selling price, dealers can provide consumers with a product warranty plan that covers all the services and replacement parts that are necessary to keep the hardware running for a couple of years [3].

Nevertheless, the total cost of software has followed a completely different path. Although building software systems was relatively inexpensive at the beginning of the commercial-computer era, the cost of carrying out this task has increased considerably over time, surpassing by far the cost of hardware [4].

According to Machiraju et al. [5], in complex IT projects the total cost of software (which includes design, coding, deployment and maintenance) is already three times the cost of hardware, and is still going up [6]. Therefore, it is not without reason that the total cost of software has become an area of great concern to both management and developers alike [7].

As a result, in the course of time, many concepts, models and techniques have been suggested by both academics and practitioners alike as a means of keeping the total cost of software under control [8]. While some of the suggestions put forward drive practitioners to reduce the cost of building software systems [9], others prompt managers to make use of tactics and strategies aimed at increasing the perceived business value of software [10].

Consistent with these ideas, the Incremental Funding Method (IFM) has emerged in recent years as an influential mechanism in the effort to bring financial discipline to the practice of software engineering and, as a result, increase the perceived value of software in the minds of investors and executives who foot the bill for software building [11].

Easy to understand and apply, the IFM's concepts and techniques have gained followers among a variety of ranks, including IT researchers, practitioners and managers, who have undertaken the task of further developing the method [12, 13, 14] and making software supporting tools widely available to others [15, 16].

The IFM builds upon the idea that it is frequently the case that software development projects can be divided into smaller self-contained software units, embodying features that are valuable to business. Furthermore, the order in which these units are built can improve quite substantially the business value of the final product of software projects.

Central to the ideas put forward by the IFM is a small set of polynomial-time approximation scheduling algorithms that allow for the maximization of the financial value yielded by software development projects. Unfortunately these algorithms provide no dependable estimate of the approximation error they make [11].

This paper introduces a statistically-based approximation scheduling algorithm that requires a polynomial computational effort to run, being able to deal with complex software projects efficiently. Moreover, the statistically-based algorithm is capable of providing reliable estimates concerning how far its final results are from the optimum scheduling solution.

The rest of this paper is organized as follows. Section 2 presents a review of the principal concepts and methods used in the subsequent sections. Section 3 introduces a statistical approach to the IFM's scheduling problem. In Section 4 the statistical approach is applied step-by-step to an example. Section 5 discusses the merits of the statistical approach and presents the conclusions of this paper.

2 CONCEPTUAL FRAMEWORK

2.1 The Incremental Funding Method

The Incremental Funding Method (IFM), credited to Denne and Cleland-Huang [17], is a financially conscious approach to software development that uses the ideas of Chang et al. on Functional Class Decomposition [18] to partition the software to be developed into smaller self-contained units that create value for business and can be deployed in shorter periods of time.

According to Denne and Cleland-Huang [17] such units, called *Minimum Marketable Feature Modules* or *MMFs* for short, create value for business in at least one of the following areas:

1. competitive differentiation,
2. revenue generation,
3. cost savings,
4. brand projection and
5. enhanced customer loyalty.

Although an MMF is a self-contained unit, it is often the case that it can only be developed after other project parts have been completed. These project parts may be either other MMFs or the architectural infrastructure, i.e. the set of basic features that offers no direct value to customers, but that are required by the MMFs.

It is also important to keep in mind that the architectural infrastructure itself can usually be decomposed into self-contained deliverable units. These units, called *architectural elements*, or AEs for short, enable the architecture to be delivered according to demand, further reducing the initial investment needed to run a project.

Moreover, the total value brought to a business by a software consisting of several interdependent MMFs and AEs, each one with its own cash-flow stream and precedence restrictions, is highly dependent on the order in which these units are developed. Consider the diagram presented in Figure 1, which describes the dependency relations that hold true among the software units (SU) of a software building project that have been divided into MMFs and AEs.

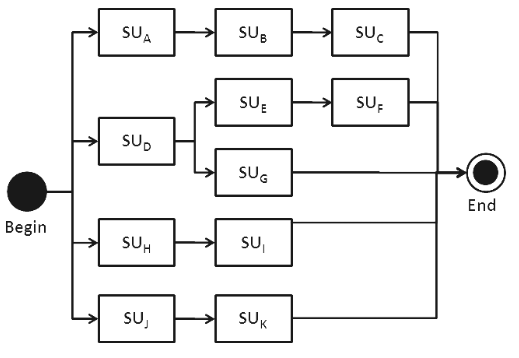


Figure 1. A precedence diagram

In the diagram $\mathrm{SU}_{X\in\{A,B,\ldots,K\}}$ are either MMFs or AEs, and *Begin* and *End* are dummy software units that take no time to be developed, require no capital investment and yield no returns. Also, an arrow going from one software unit to another, e.g. $\mathrm{SU}_A \rightarrow \mathrm{SU}_B$, indicates that the development of the former (SU_A) must precede the development of the latter (SU_B). In these circumstances, SU_A is called a predecessor of SU_B.

It should be noted that predecessor is a transitive relation. Therefore, as $\mathrm{SU}_A \rightarrow \mathrm{SU}_B$ and $\mathrm{SU}_B \rightarrow \mathrm{SU}_C$, then necessarily $\mathrm{SU}_A \rightarrow \mathrm{SU}_C$. Most often transitive relations are not made explicit in precedence diagrams, so as to keep them simple.

Table 1 shows the undiscounted cash-flow elements of each software unit in the diagram introduced in Figure 1. See Schniederjans [19] for an introduction to the basic concepts of software-project financial appraisal.

For example, according to the information presented in Table 1, SU_A requires an initial investment of US\$ 200 000, or US\$ 200K for short. Once its development is completed at the end of the first period, it provides a series of positive returns

Cash-flow Elements (US$ 1 K)								
Unit	**Period**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
SU_A	−200	98	89	81	72	63	54	45
SU_B	−250	0	0	0	0	0	0	0
SU_C	−250	63	90	117	144	171	198	200
SU_D	−200	0	0	0	0	0	0	0
SU_E	−350	35	35	70	90	108	126	144
SU_F	−120	90	90	90	135	135	135	135
SU_G	−350	60	80	100	60	70	70	70
SU_H	−100	50	50	50	25	5	5	5
SU_I	−100	35	25	25	25	5	5	5
SU_J	−50	0	0	0	0	0	0	0
SU_K	−75	25	25	25	25	25	25	20
Unit	**Period**							
	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
SU_A	36	27	18	9	5	4	3	2
SU_B	0	0	0	0	0	0	0	0
SU_C	225	225	225	225	225	225	225	225
SU_D	0	0	0	0	0	0	0	0
SU_E	162	180	180	180	180	180	180	180
SU_F	135	135	135	135	135	135	135	135
SU_G	80	120	150	250	450	100	100	100
SU_H	100	20	20	20	20	20	20	20
SU_I	5	5	5	5	5	5	5	5
SU_J	0	0	0	0	0	0	0	0
SU_K	20	20	20	20	10	10	10	10

Table 1. Software unit cash-flow elements

until the sixteenth period, when the software as a whole becomes obsolete and has to be replaced by a new and more advanced tool. A similar path is followed by units SU_C, SU_E, SU_F, SU_G, SU_H, SU_I and SU_K. Therefore, all these units are indeed MMFs.

A different path is followed by units U_B, U_D and U_J. Once they are completed, they provide no financial returns on their own in respect of the investment required for their development. Hence, these units are architectural elements.

Because it is improper to perform mathematical operations on monetary values without taking into account a discount rate, in order to compare the financial value of different MMFs and the investment required by AEs, one has to resort to their discounted cash-flow [19].

Table 2 shows the sum of the discounted cash-flow of each software unit in Figure 1, considering a discount rate of 0.5 % per period. Such a sum is the net present value (NPV) of all cash-flow elements of a unit.

In order to make understanding easier, the figures presented in Table 2 have been rounded to the nearest integer value. The remaining figures presented in this paper follow the same convention.

Net Present Value (US$ 1 K)								
Unit	**Period**							
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
U_A	391	387	382	376	370	360	342	315
U_B	−249	−248	−246	−245	−244	−243	−241	−240
U_C	2 398	2 179	1 962	1 745	1 530	1 316	1 102	890
U_D	−199	−198	−197	−196	−195	−194	−193	−192
U_E	1 578	1 405	1 232	1 061	890	720	552	383
U_F	1 684	1 552	1 420	1 289	1 158	1 029	900	771
U_G	1 418	1 319	1 220	1 122	703	470	330	218
U_H	314	294	274	254	235	215	196	176
U_I	61	56	51	46	41	37	32	27
U_J	−50	−50	−49	−49	−49	−49	−48	−48
U_K	204	194	184	174	164	145	126	107
Unit	**Period**							
	9	**10**	**11**	**12**	**13**	**14**	**15**	**16**
U_A	280	238	187	128	61	−13	−95	−185
U_B	−239	−238	−237	−235	−234	−233	−232	−231
U_C	679	492	308	149	16	−92	−174	−231
U_D	−191	−190	−189	−188	−187	−187	−186	−185
U_E	233	99	−17	−116	−198	−262	−292	−323
U_F	643	516	389	263	138	55	−28	−111
U_G	143	78	14	−51	−106	−197	−269	−323
U_H	84	79	74	69	45	−1	−47	−92
U_I	22	18	13	8	−15	−38	−60	−92
U_J	−48	−48	−47	−47	−47	−47	−46	−46
U_K	88	69	46	22	−1	−24	−47	−69

Table 2. Software unit net present value

For instance, according to the information presented in Table 2, if U_C is developed in the first period, it yields an NPV of US$ 2 398 K i.e.

$$\frac{-250}{(1+0.5\,\%)^1}+\frac{63}{(1+0.5\,\%)^2}+\frac{90}{(1+0.5\,\%)^3}+\cdots+\frac{-225}{(1+0.5\,\%)^{16}}.$$

On the other hand, if U_C is developed in the second period, it yields an NPV of US$ 2 179 K, in the third US$ 1 962 K and so on.

Obviously, not every MMF can be developed in the first period. The precedence diagram presented in Figure 1 indicates that only SU_A, SU_D, SU_H and SU_J can be developed in that period. Because in this example, at any given time, only one unit can be in its development phase, SU_C, for example, cannot be developed until the third period at best.

Furthermore, each particular sequence of software units yields its own NPV. For instance, the sequence

$$\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_F \to \mathrm{SU}_G \to \mathrm{SU}_A \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_H \to \mathrm{SU}_J$$
$$\to \mathrm{SU}_K \to \mathrm{SU}_I$$

yields US\$ 5 187 K, which is the highest NPV among all possible development sequences.

It is important to note that the NPV of a software-unit development sequence is the sum of the NPV of each of its components, considering the period in which they are expected to be built. Therefore,

$$\mathrm{NPV}(\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_F \to \mathrm{SU}_G \to \mathrm{SU}_A \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_H \to \mathrm{SU}_J \to \mathrm{SU}_K$$
$$\to \mathrm{SU}_I) = \mathrm{NPV}_1(\mathrm{SU}_D) + \mathrm{NPV}_2(\mathrm{SU}_E) + \mathrm{NPV}_3(\mathrm{SU}_F) + \ldots \mathrm{NPV}_{11}(\mathrm{SU}_I) =$$
$$(-199 + 1\,405 + 1\,420 + \ldots + 13) \times \mathrm{US\$}1\,K = \mathrm{US\$}\,5\,187\,\mathrm{K},$$

where $\mathrm{NPV}_t(\mathrm{SU}_X)$ is the NPV of unit SU_X, considering that its development starts in period t.

Table 3 indicates all possible development sequences for the software units introduced in Figure 1 together with their respective NPV, considering that:

1. the first non-dummy unit must be developed in period one,
2. at any given period only one unit can be in its development stage,
3. once the development of a software unit starts it cannot be stopped or paused,
4. there is no delay between the completion of a software unit and the beginning of the development of the next, and
5. all software units have to be developed eventually.

These conventions are adopted in the rest of this paper.

Sched. Option	**Period**									**NPV (US\$ 1K)**
	1	**2**	**3**	**4**	...	**8**	**9**	**10**	**11**	
1	SU_D	SU_E	SU_F	SU_G	...	SU_H	SU_J	SU_K	SU_I	5 187
2	SU_D	SU_E	SU_F	SU_G	...	SU_H	SU_I	SU_J	SU_K	5 173
3	SU_D	SU_E	SU_F	SU_G	...	SU_H	SU_J	SU_I	SU_K	5 169
4	SU_D	SU_E	SU_G	SU_F	...	SU_H	SU_J	SU_K	SU_I	5 154
5	SU_D	SU_E	SU_G	SU_F	...	SU_H	SU_I	SU_J	SU_K	5 140
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
207 900	SU_D	SU_J	SU_A	SU_B	...	SU_G	SU_E	SU_F	SU_C	1 561

Table 3. Scheduling options

2.2 The IFM's Scheduling Algorithms

Because, in general, the number of possible implementation sequences grows exponentially with the number of software units to be built, it is often the case that the sequence that maximizes the financial return of a software project cannot be found in polynomial time [17].

Therefore, for software development projects that have been divided into MMFs and AEs, the IFM provides managers and developers with three distinct approximation algorithms to find the implementation sequence that maximizes a project's NPV, i.e. the greedy, the simple look-ahead and the weighted look-ahead approaches.

2.3 The Greedy Approach

The greedy approach is based upon a shortsighted heuristics that selects the next software unit to be built among those whose predecessors have already been fully developed. According to the greedy approach, the unit to be developed next is always the one with the highest NPV, considering the development period that the software project is currently in.

Consider the example introduced in Section 2.1. According to the information presented in Figure 1, the only units that can be developed in the first period are SU_A, SU_D, SU_H and SU_J.

As stated in Table 2, if developed at this point in time, these units yield NPVs of US\$ 391 K, US\$ −199 K, US\$ 314 K and US\$ −50 K, respectively. As SU_A is the unit that yields the highest NPV, this is the unit that is selected for development by the greedy approach.

The unit candidates for development in the second period are SU_B, SU_D, SU_H and SU_J. At this point in the development cycle, these units yield NPVs of US\$ −248 K, US\$ −198 K, US\$ 294 K and US\$ −50 K, respectively. Therefore, SU_H, which yields the highest NPV, is the unit selected for development by the greedy approach.

The process proceeds until the eleventh period, when the last unit is selected for development. At this point, the greedy approach selects the following development sequence for the software project introduced in Section 2.1:

$$\mathrm{SU}_A \to \mathrm{SU}_H \to \mathrm{SU}_I \to \mathrm{SU}_J \to \mathrm{SU}_K \to \mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_F \to \mathrm{SU}_G$$

$$\to \mathrm{SU}_B \to \mathrm{SU}_C,$$

which yields an NPV of US\$ 2 193 K.

2.4 The Simple Look-Ahead Approach

The simple look-ahead approach is a farseeing heuristics that analyses the paths connecting the units that have already been built to the undeveloped software units. Denne and Cleland-Huang [17] call these paths *strands*.

According to the simple look-ahead approach the next unit to be developed is always the one heading the strand that yields the highest NPV, considering the development period the software project is currently in.

Consider the example introduced in Section 2.1, because the dummy unit *Begin* requires no time to be built, at the very beginning of period 1 its development will have already been completed.

Therefore, the initial set of strands analysed by the simple look-ahead approach contains the software units in the paths and sub-paths connecting the already developed unit *Begin* to the remaining undeveloped units. Table 4 presents the strands considered by the single look-ahead approach in the first period of the development cycle.

Strand	**NPV (US$ 1 K)**	
	Adding Terms	**Total**
$\mathrm{SU}_A \curvearrowright SU_B \curvearrowright SU_C$	$391 - 248 + 1\,962$	2 105
$\mathrm{SU}_A \curvearrowright SU_B$	$391 - 248$	143
SU_A	391	391
$\mathrm{SU}_D \curvearrowright SU_E \curvearrowright SU_F$	$-199 + 1\,405 + 1\,420$	2 606
$\mathrm{SU}_D \curvearrowright SU_E$	$-199 + 1\,405$	1 206
SU_D	-199	-199
$\mathrm{SU}_D \curvearrowright SU_G$	$-199 + 1\,132$	933
$\mathrm{SU}_H \curvearrowright SU_I$	$314 + 56$	370
SU_H	314	314
$\mathrm{SU}_J \curvearrowright SU_K$	$-50 + 194$	144
SU_J	-50	-50

Table 4. Strands considered by the single look-ahead approach in the first period of the development cycle

As $SU_D \curvearrowright SU_E \curvearrowright SU_F$ is the strand that yields the highest NPV, i.e. US$ 2 606 K, SU_D is the software unit initially selected for development by the single look-ahead approach.

In the beginning of the second period unit U_D will have already been built, so the set of strands requires some adjustment. Table 5 presents the strands that are analysed by the simple look-ahead approach at the beginning of that period.

Because $\mathrm{SU}_E \curvearrowright SU_F$ is the strand that yields the highest NPV, i.e. US $ 2 825 K, SU_E is selected for development. The process continues until the last undeveloped unit is selected for development in the eleventh period. For the example introduced in Section 2.1 the simple look-ahead approach indicates that development of AEs and MMfs should be carried out as follows:

$$\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_A \to \mathrm{SU}_F \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_G \to \mathrm{SU}_H \to \mathrm{SU}_I$$
$$\to \mathrm{SU}_J \to \mathrm{SU}_K,$$

which yields an NPV of US$ 4 475 K.

Strand	**NPV (US$ 1 K)**	
	Adding Terms	**Total**
$\mathrm{SU}_A \curvearrowright SU_B \curvearrowright SU_C$	$387 - 246 + 1\,745$	$1\,866$
$\mathrm{SU}_A \curvearrowright SU_B$	$387 - 246$	141
SU_A	387	387
$\mathrm{SU}_E \curvearrowright SU_F$	$1\,405 + 1\,420$	$2\,825$
SU_E	$1\,405$	$1\,405$
SU_G	$1\,319$	$1\,319$
$\mathrm{SU}_H \curvearrowright SU_I$	$294 + 51$	345
SU_H	294	294
$\mathrm{SU}_J \curvearrowright SU_K$	$-50 + 184$	134
SU_J	-50	-50

Table 5. Strands considered by the single look-ahead approach in the second period of the development cycle

2.5 The Weighted Look-Ahead Approach

According to Denne and Cleland-Huang [17], by negatively weighting the number of periods required for the development of each strand, one actually increases the chances of the simple look-ahead approach selecting the strand that maximizes the financial return of a software project.

This weighting favors the development of strands that are delivered over a shorter period of time, despite the fact that they may provide an NPV that is similar to others. Therefore, the weighted look-ahead approach facilitates even further the earlier appropriation of the financial benefits yielded by a software project.

According to Denne and Cleland-Huang [17] the most effective weighting factor depends on a number of project characteristics such as the shape of the precedence diagram and the window of opportunity, i.e. the length of time from the beginning of a software project to the time when the project's final product becomes obsolete and has to be replaced by a more attractive solution.

Denne and Cleland-Huang [17] suggest the use of the following formula to negatively weight the NPV of a given strand S:

$$\text{Weighted-NPV(S)} = \text{NPV(S)} \times (1 - (\text{WF} \times (\text{p} - 1))), \tag{1}$$

where WF is a pre-selected weighting factor and p is the number of periods necessary to build all software units in S.

Practical experiments carried out by Denne and Cleland-Huang [17] indicate that for a sixteen-period window of opportunity the ideal weighting factor belongs to the close interval [10 %..15 %], while for an eight-period window it belongs to the close interval [20 %..25 %].

For example, consider a strand $\mathrm{SU}_X \curvearrowright \mathrm{SU}_Y \curvearrowright \mathrm{SU}_Z$ that takes three periods to be developed and yields an NPV of \$ 50 K if its development starts in period 1. Also, allow for a weighting factor of 10 %. In these circumstances

$$\text{Weighted-NPV}(\mathrm{SU}_X \curvearrowright \mathrm{SU}_Y \curvearrowright \mathrm{SU}_Z) =$$

$$\$\,50\,K \times (1 - (10\,\% \times (3 - 1))) = \$40K.$$

Now, consider a strand $\mathrm{SU}_V \curvearrowright \mathrm{SU}_W$, which yields the same NPV, if its development starts in period 1 but takes only two periods to be developed. Hence, this strand yields a weighted NPV of

$$\$\,50\,K \times (1 - (10\,\% \times (2 - 1))) = \$\,45\,K,$$

indicating that it is a more attractive choice for development.

Keeping in perspective the example introduced in Section 2.1, Table 6 presents the strands that are analysed by the weighted look-ahead approach in the first period of the development cycle. Following Denne and Cleland-Huang's advice [17], a weighting factor of 10 % has been used to figure the weighted-NPVs that are shown in Table 6.

Strand	**NPV (US$ 1 K)**	
	Unweighted	**Weighted**
$\mathrm{SU}_A \curvearrowright \mathrm{SU}_B \curvearrowright \mathrm{SU}_C$	2 105	$2\,105 * (1 - (10\,\% \times (3 - 1))) = 1\,684$
$\mathrm{SU}_A \curvearrowright \mathrm{SU}_B$	143	$143 * (1 - (10\,\% \times (2 - 1))) = 129$
SU_A	391	$391 * (1 - (10\,\% \times (1 - 1))) = 391$
$\mathrm{SU}_D \curvearrowright \mathrm{SU}_E \curvearrowright \mathrm{SU}_F$	2 626	$2\,626 * (1 - (10\,\% \times (3 - 1))) = 2\,101$
$\mathrm{SU}_D \curvearrowright \mathrm{SU}_E$	1 206	$1\,206 * (1 - (10\,\% \times (2 - 1))) = 1\,085$
SU_D	-199	$-199 * (1 - (10\,\% \times (1 - 1))) = -199$
$\mathrm{SU}_D \curvearrowright \mathrm{SU}_G$	1 120	$1\,120 * (1 - (10\,\% \times (2 - 1))) = 1\,008$
$\mathrm{SU}_H \curvearrowright \mathrm{SU}_I$	370	$370 * (1 - (10\,\% \times (2 - 1))) = 333$
SU_H	314	$314 * (1 - (10\,\% \times (1 - 1))) = 314$
$\mathrm{SU}_J \curvearrowright \mathrm{SU}_K$	144	$134 * (1 - (10\,\% \times (2 - 1))) = 130$
SU_J	-50	$-50 * (1 - (10\,\% \times (1 - 1))) = -50$

Table 6. Strands considered by the weighted look-ahead approach in the first period of the development cycle

Note that $SU_D \curvearrowright SU_E \curvearrowright SU_F$ is the strand that yields the highest Weighted-NPV, i.e. US\$ 2 101 K. Because SU_D is the software unit that heads that strand, it is this unit that is selected for development in the first period.

Table 7 presents the strands that are considered by the weighted look-ahead approach in the second period of the development cycle.

Note that it is now $\mathrm{SU}_E \curvearrowright \mathrm{SU}_F$ that yields the highest Weighted-NPV, i.e. US\$ 2 543 K. Hence, it is the unit SU_E that is selected for development in the second

Strand	**NPV (US$ 1 K)**	
	Unweighted	**Weighted**
$SU_A \curvearrowright SU_B \curvearrowright SU_C$	1 886	$1\,886 * (1 - (10\,\% \times (3-1))) = 1\,509$
$SU_A \curvearrowright SU_B$	141	$141 * (1 - (10\,\% \times (2-1))) = 127$
SU_A	387	$387 * (1 - (10\,\% \times (1-1))) = 387$
$SU_E \curvearrowright SU_F$	2 825	$2\,825 * (1 - (10\,\% \times (2-1))) = 2\,543$
SU_E	1 405	$1\,405 * (1 - (10\,\% \times (1-1))) = 1\,405$
SU_G	1 319	$1\,319 * (1 - (10\,\% \times (1-1))) = 1\,319$
$SU_H \curvearrowright SU_I$	294	$294 * (1 - (10\,\% \times (2-1))) = 294$
SU_H	314	$314 * (1 - (10\,\% \times (1-1))) = 314$
$SU_J \curvearrowright SU_K$	134	$134 * (1 - (10\,\% \times (2-1))) = 121$
SU_J	-50	$-50 * (1 - (10\,\% \times (1-1))) = -50$

Table 7. Strands considered by the weighted look-ahead approach in the second period of the development cycle

period. The process proceeds until the last unit is selected for development in the eleventh period. At this point, the weighted look-ahead approach indicates that

$$SU_D \to SU_E \to SU_F \to SU_A \to SU_B \to SU_C \to SU_G \to SU_H \to SU_I$$

$$\to SU_J \to SU_K$$

is the strand that yields the highest NPV, i.e. US$ 4 600 K.

2.6 Comparing the Results Presented by the IFM's Scheduling Algorithms

Table 8 presents the results provided by each of the IFM's scheduling algorithms considering the example introduced in Section 2.1. Moreover, for comparison purposes, Table 8 also presents the result provided by the brute-force algorithm, which exhaustively analyses all possible implementation sequences in order to select the one that yields the highest NPV.

It is important to note that without the support of the brute-force approach, which requires an exponential computing effort to yield its results, one is at a loss to determine how far from the implementation sequence that yields the actual highest NPV the outputs of the IFM's algorithms are.

2.7 The Approximation Algorithm's Mathematical Foundation

In the 1930s a general result obtained by Kolmogorov[1] [23, 24] on the theory of statistics allows for the establishment of a confidence interval around the empirical

[1] Also known as Kolmogoroff, Andrei Nikolaevich, the well-known Russian mathematician [22].

Approach	**Implementation Sequence**	**NPV (US$ 1 K)**	**Percent**
Brute force	$\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_F \to \mathrm{SU}_G \to \mathrm{SU}_A \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_H \to \mathrm{SU}_J \to \mathrm{SU}_K \to \mathrm{SU}_I$	5 187	100.0 %
Greedy	$\mathrm{SU}_A \to \mathrm{SU}_H \to \mathrm{SU}_I \to \mathrm{SU}_J \to \mathrm{SU}_K \to \mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_F \to \mathrm{SU}_G \to \mathrm{SU}_B \to \mathrm{SU}_C$	2 193	42.3 %
Single look-ahead	$\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_A \to \mathrm{SU}_F \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_G \to \mathrm{SU}_H \to \mathrm{SU}_I \to \mathrm{SU}_J \to \mathrm{SU}_K$	4 475	86.3 %
Weighted look-ahead	$\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{SU}_F \to \mathrm{SU}_A \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_G \to \mathrm{SU}_H \to \mathrm{SU}_I \to \mathrm{SU}_J \to \mathrm{SU}_K$	4 600	88.7 %

Table 8. Results presented by the IFM's algorithms

density function of any continuous random variable, with an arbitrary degree of confidence. In this section, Kolmogorov's result along with related results obtained by others [31, 29] are used to lay down the mathematical foundations of an approximation algorithm that identifies a software project's implementation sequence of MMFs and AEs that yields the highest NPV.

First, Kolmogorov's result is presented in a formal manner. Next, related work that extends Kolmogorov's result to encompass discrete random variables is discussed. Finally, all these results are combined to lay down the mathematical foundations of the approximation algorithm.

2.7.1 The Kolmogorov Confidence Contours

In formal terms, for a continuous random variable x let

$$F(x) = P(X \leq x)$$

be its cumulative density function, or cdf. Also, let X_1, X_2, …, X_n be a random sample of x and

$$S_n(x) = P_n(X \leq x) = \frac{1}{n}\sum_{i=1}^{n}\begin{cases} 1 & \text{if } X_i \leq x \\ 0 & \text{otherwise} \end{cases}$$

be the corresponding empirical density function, or edf.

In addition, let $D_n = sup|F(x) - S_n(x)|$, where *sup* stands for the supreme (least upper bound) of a set of ordinal-scale values. According to Glivenko [20] and Cantelli [21]

$$\lim_{n\to\infty} D_n = \lim_{n\to\infty} sup|F(x) - S_n(x)| \to 0,$$

i.e. as the size of the sample increases, the distance between the cdf and the edf tends toward zero.

A strong result obtained by Kolmogorov [23, 24] in the 1930s not only shows that the statistic D_n does not depend on $F(x)$, but also states that the probability

α of D_n not exceeding an arbitrary value in the form of $\frac{\lambda}{\sqrt{n}}$ is given by

$$P(D_n \leq \frac{\lambda}{\sqrt{n}}) = \alpha.$$

In the course of time λ has been tabulated for different sample sizes and levels of confidence. Table 9 presents the value of λ for different values of n (the sample size) and α (the level of confidence). Tables containing a more detailed list of values of λ can be found in [25, 26, 27, 28] and many other statistics texts.

n	α		
	0.90	**0.95**	**0.99**
10	0.323	0.369	0.457
20	0.232	0.265	0.329
30	0.190	0.218	0.270
40	0.165	0.189	0.235
$n > 40$	$\frac{1.07}{\sqrt{n}}$	$\frac{1.22}{\sqrt{n}}$	$\frac{1.52}{\sqrt{n}}$

Table 9. The value of λ for different values of n and α

As a result, if one takes k random observations of a continuous random variable x, where k is greater than 40, the probability that the distance between x's cdf and its edf is smaller than $\frac{1.22}{\sqrt{k}}$ is 0.95, i.e.

$$P(D_k \leq \frac{1.22}{\sqrt{k}}) = 0.95.$$

2.7.2 Dealing with Discrete Random Variables

In the 1970s the results obtained by Kolmogorov [23, 24] were extended to handle discrete distributions by Conover [29], and Coberly and Lewis [30] independently.

Furthermore, according to Walsh [31], when applied to discrete variables the values derived from Kolmogorov's work lead to a safely conservative estimate of $\frac{\lambda}{\sqrt{k}}$. This claim was later acknowledged by Pettitt and Stephens [32], and Conover himself [27].

2.7.3 The Basis of the Approximation Algorithm

If the random sample that is used to build an edf is comprised of NPVs of the implementation sequences of MMFs and AEs belonging to the same software project, then those confidence intervals bear a special meaning. When applied to the highest NPV in the sample, the confidence interval indicates how close to the actual absolute-maximum NPV that particular value is, in relative terms.

For instance, consider a 3 000 random sample from the implementation sequences displayed in Table 3 together with their respective NPVs. Also, let h be the highest

NPV in that sample. In these circumstances, as the estimations are conservative

$$P\left(D_{3\,000} \leq \frac{1.22}{\sqrt{3\,000}}\right) \geq 0.95 \implies P\left(|F(h) - S_{3\,000}(h)| \leq \frac{1.22}{\sqrt{3\,000}}\right) \geq 0.95.$$

Because h is the highest value in the sample, $\mathrm{S}_{3\,000}(\mathrm{h})$ is necessarily 1. As a result

$$P(|F(h) - 1| \leq 0.0223) \geq 0.95$$

$$\Downarrow$$

$$P(-0.0223 \leq F(h) - 1 \leq 0.0223) \geq 0.95$$

$$\Downarrow$$

$$P(0.977 \leq F(h) \leq 1.0223) \geq 0.95.$$

As by definition $F(h)$ cannot exceed 1,

$$P(0.977 \leq F(h) \leq 1) \geq 0.95.$$

Therefore, the probability that all the other NPVs in the set of all possible NPVs are smaller than or equal to h is 0.977, with a level of confidence that equals or exceeds 95 %. Hence, h may be considered a good approximation to the highest possible NPV, and the implementation sequence that has h as its NPV may be taken as the implementation sequence to be followed during the development of the corresponding software project. Table 10 compares the estimated and actual differences between $F(h)$ and $S_n(h)$ for different sample sizes, in absolute terms, considering the NPVs displayed in Table 3.

Note that if one is not satisfied with the results provided by a certain sample size, one may randomly increase the number of observations in the sample and improve the results until one is fully satisfied with them.

Sample Size	**h (US$ 1 K)**	**F(h)**	**|F(h) – S(h)|**	
			Real	**Estimated**
100	4 628	0.9960	0.0040	0.1220
200	4 628	0.9961	0.0039	0.0683
500	4 724	0.9979	0.0021	0.0546
1 000	4 898	0.9994	0.0006	0.0386
3 000	4 983	0.9998	0.0002	0.0223
5 000	4 995	0 9998	0 0002	0 0173

Table 10. Sample related statistics for $\alpha = 0.95$

3 THE STATISTICAL APPROACH

Consider the precedence graph $G = (\mathrm{SU}, E)$ of a software system that has been divided into MMFs and AEs. In these circumstances,

- SU = $\{su_1, su_2, \ldots, su_n\}$, the set of vertex, is a set of MMFs and AEs, and
- E, the set of edges, is a set of ordered pairs, such that if $(su_a, su_b) \in E$, then su_a is a predecessor of su_b, indicating that the development su_a must precede the development of su_b.

Figure 1 presents an example of such a graph. A quite comprehensive introduction to graph theory is given by Diestel [33].

Algorithm 1 introduces a finite sequence of steps that randomly select an implementation sequence that contains all the units in SU and complies with the precedence constraints specified in E. Table 11 presents the meaning of the variables, sets and functions used in the specification of Algorithm 1.

Symbol	Meaning
du_{su}	the duration of the development of software unit $su \in SU$
st_{su}	the period in which the development of su starts
$Pred(su)$	the set of the predecessors of su as specified in E
$Ready$	the set of all software units that are ready for development, as all their respective predecessors have already been built
$InDev$	the set of software units that are currently being developed
$Blocked$	the set of software units blocked for development, because at least one of its predecessor have not yet been developed
$Built$	the set of software units that have already been built
$period$	the time counter
$\#teams$	the number of teams available to work on the development of the software units in SU
SU'	an auxiliary set such that $SU' \subseteq SU$
$Select$	a function $\mathcal{P}(SU) \times \mathbb{N} \to \mathcal{P}(SU)$, where $\mathcal{P}(SU)$ is the power set of SU and $\mathbb{N}$ is the set of natural numbers, such that $Select(SU', k)$ randomly selects k elements from SU'

Table 11. The meaning of the variables, sets and functions used in the specification of the statistical approach scheduling algorithm

Note that Algorithm 1 permits the concurrent development of an arbitrary number of software units. However, at all times, the number of sequences that are being concurrently developed is limited by the number of development teams that are available to work on the construction of the software project under consideration.

In order to obtain the implementation sequence that best maximizes the NPV of a software project, one should first run Algorithm 1 n times. In these circumstances the choice of n depends on two factors, i.e.

- how close to the actual sequence that maximizes the NPV one wants the result to be, and
- how confident one wants to be that the result presented by the statistical approach is close enough.

Table 9 can help in solving these questions. According to the information displayed in Table 9, the higher the value of n, the closer to the actual solution one tends to be. Also, the value of α can be used to state how confident one can be that the NPV yielded by the statistical approach is close enough to the real NPV that maximizes the financial benefits of a software project, in relative terms. Table 12 summarizes the steps comprising the statistical approach.

Step	**Action**
1	Identify the precedence diagram $G = (\mathrm{SU}, E)$ of a software project that has been divided into MMFs and AEs
2	Identify the number of development teams that are available to work on the software project, i.e. $\#teams$
3	Select the number n of development sequences to be randomly generated according to the constraints imposed on E. One may use the information shown in Table 9 to better select n
4	Execute Algorithm 1 n times
5	For every execution of Algorithm 1 save the implementation sequence generated by the algorithm together with its NPV
6	Among the implementation sequences generated by Algorithm 1, identify the one that yields the highest NPV, naming this sequence H
7	Use Kolmogorov's results presented in Section 2.7 to create a confidence interval D_n around $F(h = \mathrm{NPV}(H))$
8	Take $D_n = \lvert F(h) - S(h)\rvert$ as an indicator of the relative distance between h and the NPV of the implementation sequence that actually maximizes the financial benefits of the software project
9	Take H as the sequence that best maximizes the financial benefit of the software project that has G as its precedence diagram

Table 12. The steps comprising the statistical approach

4 AN EXAMPLE

As examples tend to make understanding easier, the statistical approach proposed in Section 3 is applied step-by-step to a reasonably complex software project.

Step 1: Identify the precedence diagram of a software project

In this example one should take into consideration the diagram presented in Figure 1.

Step 2: Identify the number of development teams

Suppose that due to unfavorable market conditions only one development team is currently available to work on the software project under consideration, i.e. $\#teams = 1$.

Step 3: Select the number of development sequences to be randomly generated

In order to obtain a tight confidence interval around the estimates yielded by the statistical approach, consider generating a random sample with 5 000 observations.

Steps 4 and 5: Execute Algorithm 1 five thousand times, saving the results

Table 13 presents the 5 000 random sample generated by Algorithm 1.

Sched. Option	**Period**									**NPV (US$ 1K)**
	1	**2**	**3**	**4**	…	**8**	**9**	**10**	**11**	
1	SU_J	SU_D	SU_H	SU_A	…	SU_C	SU_I	SU_K	SU_F	2 954
2	SU_D	SU_E	SU_G	SU_A	…	SU_H	SU_J	SU_I	SU_K	4 881
3	SU_D	SU_E	SU_F	SU_G	…	SU_C	SU_J	SU_K	SU_I	5 016
⋮	⋮	⋮	⋮	⋮	…	⋮	⋮	⋮	⋮	⋮
5 000	SU_J	SU_D	SU_H	SU_A	…	SU_C	SU_I	SU_K	SU_F	2 954

Table 13. Random sample of possible implementation sequences

Step 6: Identify the implementation sequence that yields the highest NPV

Among the implementation sequences generated in the previous step

$$\mathrm{SU}_D \to \mathrm{SU}_E \to \mathrm{U}_G \to \mathrm{SU}_F \to \mathrm{SU}_A \to \mathrm{SU}_B \to \mathrm{SU}_C \to \mathrm{SU}_J \to \mathrm{SU}_K$$

$$\to \mathrm{SU}_H \to \mathrm{SU}_I$$

is the sequence the yields the highest NPV, i.e. US$ 5 076 K. Name this sequence H.

Step 7: Create a confidence interval around the NPV obtained in the previous step

Let $h = \mathrm{NPV}(H)$. According to Walsh (see Section 2.7)

$$P\left(|F(h) - 1| \leq \frac{1.22}{\sqrt{5\,000}} = 0.0173\right) \geq 0.95.$$

Note that $P(|F(h) - 1| \leq 0.0173) \geq 0.95 \implies P(0.9827 \leq F(h) \leq 1) \geq 0.95$. As $F(h) = 0.9998$ (see Table 3), the result predicted by Walsh not only is accurate, but it also makes H a very good choice for the implementation sequence that should be followed during the development of the corresponding software project.

Step 1: Initialization of the set containing the software units in development and that have already been built. Also, the initialization of the time counter.

$InDev \leftarrow \{\}$
$Built \leftarrow \{\}$
$period \leftarrow 1$

Step 2: Initialization of the set containing the software units that are ready to be developed.

$Ready \leftarrow \{su \in SU | Pred(su) = \{\}\}$
if $Ready = \{\}$ then go to *Step 8*

Step 3: Initialization of the set containing the software units that are not ready to be developed.

$Blocked = SU - Ready$

Step 4: Select software units to be developed.

if $|InDev| < \#teams \wedge \text{Ready} \neq \{\}$

$$\text{then} \begin{cases} \text{SU}' \leftarrow Select(Ready, \#teams - |InDev|) \\ \forall su' \in \text{SU}' \implies st_{su'} \leftarrow period \\ InDev \leftarrow InDev \cup \text{SU}' \end{cases}$$

Step 5: Finish building the software units whose completion time are the earliest. The time counter is updated accordingly

$$\text{SU}' = \{su_i \in InDev |\ \forall su_j \in InDev \implies st_{su_i} + du_{su_i} \leq st_{su_j} + du_{su_j}\}$$

$Built \leftarrow Built \cup \text{SU}'$
$InDev \leftarrow InDev - \text{SU}'$
Take any$su' \in \text{SU}'$
$period \leftarrow st_{su'} + du_{su'}$

Step 6: Make available for development the software units whose predecessors have already been built

$Ready \leftarrow \{su_i \in Blocked | \forall su_j \in \text{Pred}(su_i) \implies su_j \in Built\}$

Step 7: Check whether there are still software units to be developed

if $InDev \neq \{\} \vee Ready \neq \{\}$ then go to *Step 4*

Step 8: All software uits have been scheduled

Stop

Algorithm 1. The statistical approach scheduling algorithm

5 CONCLUSIONS

Many of the software systems that are being built today tend to be complex from the perspective of not only the problems they are set up to solve, but also in dealing with the large number of self-contained units they are often divided into. Therefore, situations in which there is an exponential number of possible scheduling options for the development of these units are slowly becoming the rule in software projects rather than the exception [34].

The approximation method presented in this paper is specially useful in the presence of large sets of interconnected MMFs and AEs, when the use of the brute-force approach becomes inviable due to the computational effort it requires to yield a precise result. Hence, it is better suited to deal with a larger variety of real world situations.

There is however an obvious drawback one should be aware of: if the project under consideration is small (less than ten software units for example) the brute-force approach to finding the scheduling option that maximizes the financial value of a project is likely to provide an exact answer in a bearable length of time. Therefore, in these circumstances the brute-force approach may be preferable to the statistical approach presented in this paper as it tends to yield a better result with a computational effort that may be considered negligible.

Nevertheless, as the statistical approach provides approximation solutions with an arbitrary degree of confidence, it is preferable to the heuristical methods put forward by the authors of the IFM, which provide no dependable estimate for the approximation error they make.

Moreover, by using the statistical approach, whose approximation error can be made as small as one wishes, developers and project managers can feel more confident about the rightness of the decisions they made with the intent to speed up the appropriation of the financial benefits yielded by the software projects they are responsible for, keep the capital required by those projects small and mitigate the risk exposure due to changes in the marketplace.

REFERENCES

[1] Campbell-Kelly, M.: Will the Future of Software be Open Source? Communications of the ACM, Vol. 51, 2008, No, 10, pp. 21–23.

[2] Swedin, E. G.—Ferro, D. L.: Computers: The Life Story of a Technology. The Johns Hopkins University Press 2007.

[3] Kurataa, H.—Namb, S. H.: After-Sales Service Competition in a Supply Chain: Optimization of Customer Satisfaction Level or Profit or Both? International Journal of Production Economics, Vol. 127, 2010, No. 1, pp. 136–146.

[4] Jones, C.: Estimating Software Costs: Bringing Realism to Estimating. 2nd ed., McGraw-Hill Osborne Media 2007.

[5] Machiraju, V.—Bartolini, C.—Bartolini, C.—Casati, F.: Chapter 1: Technologies for Business-Driven Infomation Technology Management. In: Caveron, L., Maamar, Z. (Eds.): Extending Web-Services Technologies: The Use of Multi-Agent Approaches. Springer 2005, pp. 1–28.

[6] van Vliet, H.: Software Engineering: Principles and Practice. 3rd ed., Wiley 2008.

[7] Toffolon, C.—Dakhli, S.: An Iterative Meta-Lifecycle for Software Development, Evolution and Maintenance. In: Proceedings of Third International Conference on Software Engineering Advances (ICSEA) 2008, Sliema, Malta, pp. 284–289.

[8] Sauvé, J.—Moura, A.—Sampaio, M.—Jornada, J.—Radziuk, E.: An Introductory Overview and Survey of Business-Driven IT Management. In: Hellerstein,

J. L., Stiller, B. (Eds.): Proceedings of the First IEEE/IFIP International Workshop on Business-Driven IT Management(BDIM), Vancouver, Canada 2006, pp. 1–10.

[9] Trendowicz, A.—Münch, J.: Chapter 6: Factors Influencing Software Development Productivity State-of-the-Art and Industrial Experiences. In: Zelkowitz, M. V. (Ed.): Advances in Computers, Vol. 77, Academic Press 2009, pp. 185–241.

[10] Boehm, B.: Chapter 1: Value-Based Software Engineering: Overview and Agenda. In: Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., Grnbacher, P. (Eds.): Value-Based Software Engineering, Springer 2006. pp. 3–11.

[11] Denne, M.—Cleland-Huang, J.: The Incremental Funding Method: Data-Driven Software Development. Software, IEEE 2004, Vol. 21, 2004, No. 3, pp. 39–47.

[12] Little, T.: Value Creation and Capture: A Model of the Software Development Process. IEEE Software 2004, pp. 48–53.

[13] Steindl, C.: From Agile Software Development to Agile Businesses. In: Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE Computer Society 2005, pp. 258–265.

[14] Szoke, A.: Decision Support for Iteration Scheduling in Agile Environments. In: Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M. J., Szyperski, C., Bomarius, F., et al. (Eds.): Product-Focused Software Process Improvement. Vol. 32 of Lecture Notes in Business Information Processing, Springer Berlin Heidelberg 2009, pp. 156–170.

[15] Denne, M.—Cleland-Huang, J.: IFM Software Tool 2004. Information available on the Internet at `www.softwarebynumbers.org`. Site last visited on May 31, 2011.

[16] Gylterud, S.—Jodal, S. M.—Knutsen, J.—Ottesen, E. B.—Taraldset, R. B.: MMF Planner. Customer Driven Project, Norwegian University of Science and Technology 2007. Information available on the Internet at `https://github.com/jodal/mmfplanner`. Site last visited on May 31, 2011.

[17] Denne, M.—Cleland-Huang, J.: Financially Informed Requirements Prioritization. In: Proceedings of the 27th International Conference on Software Engineering (ICSE), St. Louis, Missouri, USA 2005, pp. 710–711.

[18] Chang, C. K.—Hua, S.—Cleland-Huang, J.—Kuntzmann-Conbelles, A.: Function-Class Decomposition. IEEE Computer 2001, Vol. 34m 2001, No. 12, pp. 87–93.

[19] Schniederjans, M. J.—Hamaker, J. L.—Schniederjans, A. M.: Information Technology Investment: Decision-making Methodology. 2nd ed., World Scientific Publishing Company 2010.

[20] Glivenko, V. I.: Sulla Determinazione Empirica Delle Leggi di Probabilità. Giornale dell'Istituto Italiano degli Attuari, 1933, No. 4, pp. 92–99.

[21] Cantelli, F. P.: Sulla Determinazione Empirica Delle Leggi di Probabilità. Giornale dell'Istituto Italiano degli Attuari, 1933, No. 4, pp. 221–424.

[22] Charpentier, E.—Nikolski, N. K.—Lesne, A.: Kolmogorov's Heritage in Mathematics. Springer 2007.

[23] Kolmogorov, A. N.: Sulla Determinazione Empirica Delle Leggi di Probabilità. Giornale dell'Istituto Italiano degli Attuari, 1933, No. 4, pp. 83–91.

[24] Kolmogorov, A. N.: Confidence Limits for an Unknown Distribution Function. Annals of Mathematical Statistics, Vol. 12, 1941, No. 4, pp. 461–463.

[25] Birnbaum, Z. W.—Tingey, F. H.: One-Sided Confidence Contours for Probability Distribution Functions. The Annals of Mathematical Statistics, Vol. 22, 1951, No. 4, pp. 592–596.

[26] Miller, L. H.: Table of Percentage Points of Kolmogorov Statistics. Journal of the American Statistical Association. Vol. 51, 1956, No. 273, pp. 111–121.

[27] Conover, W. J.: Practical Nonparametric Statistics. 3rd ed., Wiley 1998.

[28] Gibbons, J. D.—Chakraborti, S.: Nonparametric Statistical Inference. 5th ed., Chapman and Hall/CRC 2010.

[29] Conover, W. J.: A Kolmogorov Goodness-of-Fit Test for Discontinuous Distribution. Journal of the American Statistical Association, Vol. 67, 1972, No. 339, pp. 591–596.

[30] Coberly, W. A.—Lewis, T. O.: A Note on a One-Sided Kolmogorov-Smirnov Test of Fit for Discrete Distribution Functions. Annals of the Institute of Statistical Mathematics, Vol. 24, 1972, No. 1, pp. 183–187.

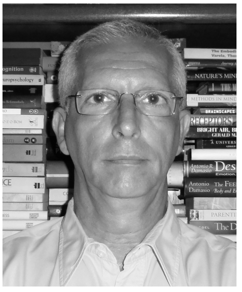
[31] Walsh, J. E.: Bounded Probability Properties of Kolmogorov-Smirnov and Similar Statistics for Discrete Data. Annals of the Institute of Statistical Mathematics, Vol. 15, 1963, No. 1, pp. 153–158.

[32] Pettitt, A. N.—Stephens, M. A.: The Kolmogorov-Smirnov Goodness-of-Fit Statistic with Discrete and Grouped Data. Technometrics. Vol. 19, 1977, No. 2, pp. 205–210.

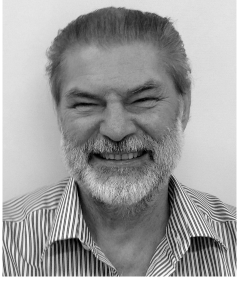
[33] Diestel, R.: Graph Theory. 4th ed., Springer 2010.

[34] Taveira, G. V.—Alencar, A. J.—Schmitz, E. A.: A Method for Portfolio Management and Prioritization: An Incremental Funding Method Approach. In: Filipe, J., Cordeiro, J. (Eds.): Proceedings of the International Conference on Enterprise Information Systems (ICEIS), Vol. 3, Funchal, Madeira Island, Portugal 2010, pp. 23–33.

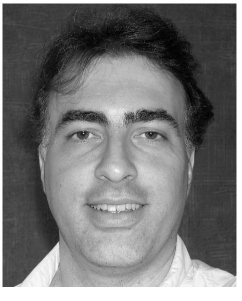
Antonio J. Alencar is a researcher with the Federal University of Rio de Janeiro (UFRJ), Brazil. He received his B. Sc. in mathematics and M. Sc. in system engineering and computer science from UFRJ. He received a D. Phil. in computer science from Oxford University, England. His research interests include economics of software engineering, IT strategy and risk analysis.



Carlos A. S. Franco is a Professor with the Federal University of Rio de Janeiro, Brazil. He received his Dr. Sc. in system engineering and computer science from the Federal University of Rio de Janeiro. His research interests include software development methodologies and economics of software engineering.



Eber A. Schmitz is a professor of computer science with the Federal University of Rio de Janeiro. He received his B. Sc. in electrical engineering from the Federal University of Rio Grande do Sul, his M. Sc. in electrical engineering from the Federal University of Rio de Janeiro and his Ph. D. in computer science and control from the Imperial College of Science, Technology and Medicine, England. His research interests include software modeling tools, business process modeling and stochastic modeling.



Alexandre L. Correa is a Professor of computer science with the Federal University of the State of Rio de Janeiro. He recved his B. Sc. in mathematics, and M. Sc. and Dr. Sc. in system engineering and computer science from the Federal University of Rio de Janeiro. His research interests include reverse engineering, system validation and software development tools.