

Risks, requirements and estimation of a software project

Roberto Meli

Abstract

The purpose of this work is to make a practical contribution to those interested in achieving a successful software project by knowingly and methodically reducing the major risks associated with it. For a project, risk may be imagined as the possibility of not achieving one or more of its specified and agreed-upon objectives, with the damage that derives from this. Typical areas of expected results are those of functional requirements, product quality, working effort, and the duration and cost of the development or enhancement project. Systematic review of the actual reasons for project failure has shown that poor definition of objectives and inadequate allocation of resources are two of the most significant factors capable of negatively impacting the project's result. For this reason, managing user requirements and estimating the resources needed for the project are two processes of strategic importance for any organization that wishes to achieve quality results without wasting human and economic resources. Unfortunately, both processes are marked by great uncertainty, as they are often based on incomplete or summary information. Disciplined requirements management can ensure that the system being developed will be the "right" system for the user's needs. At the same time, accurate estimation of the size, effort, duration, staff, and costs associated with the project's development or enhancement may make it possible to reduce the waste caused by failure to meet deadlines and by the need to review budgets, involve a greater number of developers, etc. Every improvement obtained in these areas will more than pay for the associated implementation costs. Therefore, when we can detect a change in requirements as soon as possible and immediately derive a new cost estimation, we can significantly diminish the project risks. This work illustrates the integrated framework of reference for the process to manage risk and requirements, and to estimate software supported by public domain methods and commercial tools.

1. The risk of a software project

Risk has been defined in many ways, and with different nuances [1]. This work picks up on the SAFE method [2], which defines as risk: *the expected value of the damage caused to the project by a combination of uncertain conditions.*

We may recall that the expected value of a probabilistic phenomenon – in simplified statistical terms – may be understood as the product of the probability of an event, expressed with a value from 0 to 1, by the numerical quantification linked to the event itself, extended to all the possible events for that phenomenon. For example, if we were to toss a coin and lose 1 Euro for each “tails” and nothing for each “heads,” the expected value of the loss associated with the toss would be 0.5 Euro ($0.5 \times 1 + 0.5 \times 0$). Actually, we would never lose exactly 0.5 Euro on any toss (we could only lose 1 Euro or nothing), but if we played this game long enough and divided the total loss by the number of tosses, the average loss would be very close to or exactly 0.5 Euro. In fact, according to the law of large numbers, the number of “heads” tends, over time, to equal that of “tails.” Therefore, 1-Euro losses would be around one half the number of tosses. The average loss would then be one half of the single possible loss : 0.5 Euro.

Getting back to risk, we may say that risk involves inconvenience, suffering, or loss perceptible by given subjects, as the outcome of the occurrence of situations that are possible but not certain. Therefore, for there to be risk:

- one or more subjects must be involved,
- there must be an event or combination of events with a non-null probability of occurring,
- there must be a cause and effect chain capable of determining a final situation,
- this situation must be considered negative with respect to the expectations and scale of values of the subjects involved, thereby causing damage to them.

This approach, called Condition, Transition, Consequence (CTC), has been described by the Software Engineering Institute in [1].

The subjectivity of point of view is essential to identifying risk, since a given situation may take on negative connotations for some, and positive for others. Therefore, to appropriately assess the potential negative consequences of an event, that is the amount of the associated damage, it is essential to understand the scale of values and expectations of a project's stakeholders. Although assessment of the other component of risk - the probability of the chain of negative events - should be more objective than that of the amount of damage, there are many situations in which this probability is influenced by the subject that is doing the assessing. Entrusting the same project to two different Project Managers produces two different risk assessments not only because they see things differently, but also because they have different personal abilities to influence things. A more expert and powerful Project Manager may attribute lower risk to the possibility of not receiving enough resources for his or her project than would a Project Manager just starting out, with little sway among top management. In these cases, risk is not merely perceived as different, but is different as a matter of fact.

To put it less technically and more intuitively, we may say that there is risk whenever it is suspected that the project's objectives in terms of product quantity (number and type of functions and data), result quality (non-functional requirements), cost, or duration are not achieved.

1.1. An initial, basic risk factor

Analysis of the reasons for project failure as reported by Project Managers in many studies fingers indeterminate, ambiguous, undefined, or generic - in short, poorly formulated - project objectives as the primary culprits. In this situation, a project behaves as if it were proceeding blindly, staggering in search of the right direction, at the mercy of those pushing it to and from, depending on the whims of the stakeholders who are constantly changing their mind as to what they need. Of course, if we have no clear idea beforehand of where we wish to go, we would have a hard time convincing travellers afterwards that where we have got to is where we wish to have gone. Everyone would be free to think and maintain that the agreements were different, and paradoxically, everyone would be right. An ambiguous formulation may hide hundreds of different specific formulations. Therefore, for objectives to be able to guide and drive the project work, they must be shared and well formulated. To objectives to be formulated in a suitable manner, they must be clear, that is expressed in a language that is understandable to the various stakeholders, specific, and above all measurable. In fact, an objective's measurability associated with the definition of the expected values of its measurements, makes it possible to recognize its having been achieved and, definitively, the project's success.

Consulting and training experience tends to show that project groups are generally capable of dealing with high-tech challenges and using complex working methods and techniques, but are not particularly able to or skilled in focusing on the real problems and objectives to be pursued. It is as if

they had a rifle that was well-oiled, loaded, and ready to shoot, but whose sights were regulated only approximately, with shooters at risk of shooting themselves instead of their desired target.

There are many reasons for this phenomenon, which have to do with the lack of attention paid during a project's initial phases to defining what problems are to be faced, which stakeholders are to be involved, and what expectations these stakeholders have as to results. Many working groups tend to think that until they get to performing practical activities, such as software coding, they are basically wasting time, thereby justifying the saying that the less time we spend analyzing the situation, the more time we spend revising, testing, and correcting the necessary software programs.

Often, the first question a project group asks is "what do we have to do?" when it would be better first to ask: "what do we wish to be left with after the project is finished?" It is, in fact, a common habit to confuse activities with results, means with ends. Performing market research is not a good project objective; it is a possible action. On the other hand, obtaining knowledge on a certain market segment is a significant objective. Perhaps it would be better to purchase an ready-made market study than to produce a new one. Therefore, concentrating on activities often leads us to neglect less immediate, and sometimes more effective solutions.

In addition to the bad habit of not defining a project's objectives and problems because we think there is not enough time to do so, it is often objectively difficult to produce a vision common to all stakeholders of what the project is and what effects it should produce. And yet, sharing objectives is essential to not getting mired in impossible missions. Many techniques can be of help here, such as the GOPP (Goal Oriented Project Planning) method [3] that is increasingly gaining ground in contexts where concerted action among distinct subjects with divergent interests is a critical factor for the projects' success.

Another cause for poor quality of objectives is the perception by the group in charge of producing a software system that it has been asked to develop only a technological tool that will serve others to pursue goals higher than which cannot be dealt with. In other words, the tendency is to focus on the tool that is required, losing sight of the purpose for which it is needed. In many cases, this tendency is reinforced by the project's customer, which often relegates the software supplier (in-house or outside) to the role of mere executor, not allowing it to perceive the goals, and restricting its mandates to means alone. Unfortunately, once the mandate is assigned, it happens that no one is any longer concerned with guaranteeing coherence between means and ends. This gives rise to software products that are "aesthetically" or technologically magnificent, but that no one uses, because they are ill suited for the organization's actual needs. We therefore have the noted phenomenon of the so-called "absent solution": instead of analyzing problems in their true nature, we think of a solution already available, deny it, and believe that the lack of this solution is the problem gripping the stakeholders. Therefore, we often have solutions in search of a problem instead of the other way around.

1.2. A second basic risk factor

It is common experience that on average, projects end up late and over budget. Therefore, inadequate allocation of resources for carrying out project planned activities is also considered a basic cause for project failure. This may derive from two different but equally important reasons:

- the project receives the right resources but is improperly managed and wastes a good part of them, making it necessary to assign additional resources;
- the project receives fewer resources than would be needed, due to poor initial estimation.

It is often difficult to understand whether improper resource allocation depends on poor management or on poor estimation. When in doubt, it is necessary to act on both fronts, improving the abilities to both manage and estimate the project's essential resources.

The two issues dealt with to this point – low objectives/requirements quality and poor resource estimation ability – are highly interrelated, because the latter depends greatly on the former. If a project fails to suitably define the functional and non functional requirements and objectives of the required software product, the resources necessary for development will also be imprecise, unsuitable, and often insufficient.

In addition, very frequently, when estimates are needed – which is to say at the beginning of the project – the very information on which we base our “wagers” is generic, imprecise, and lacking in detail by necessity. And here, any improvement in the process to define requirements, and especially to consequently capitalize on the necessary resources, holds particular importance in reducing the two leading causes of project risk.

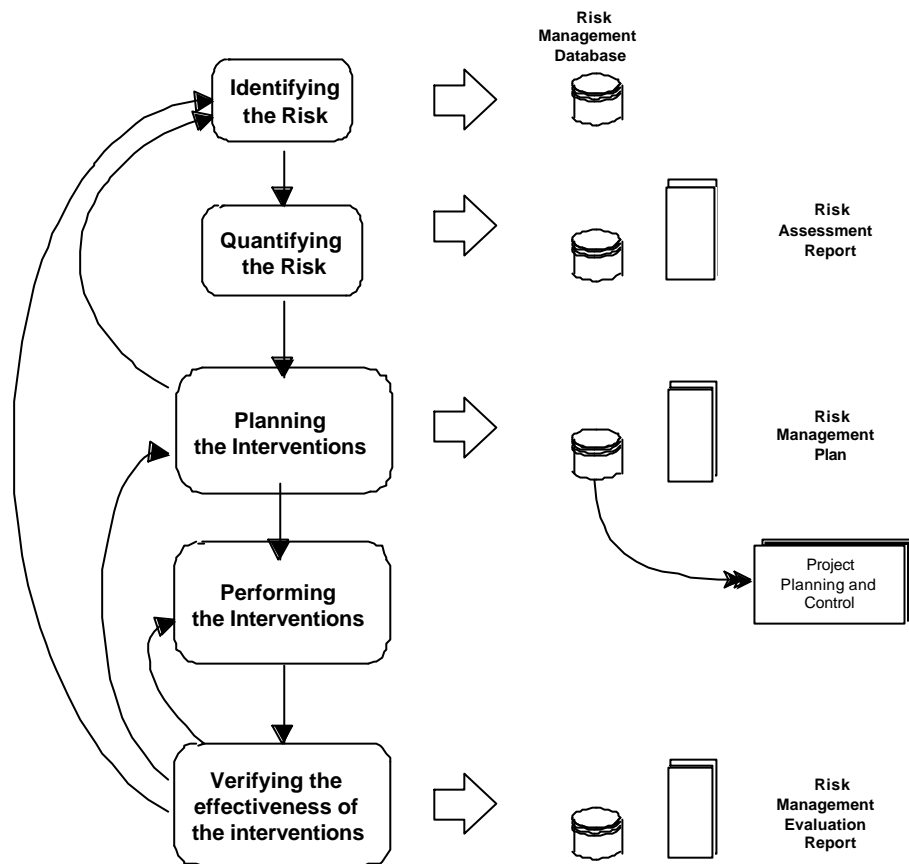
1.3. A risk reduction strategy

After the above, it appears clear that in order to significantly reduce a project's risks, a good action plan may be that of:

- making the risk management process explicit and methodical
- improving requirements management
- improving project estimation ability

1.4. The SAFE risk management method

The aforementioned work [2] introduces the SAFE (Safe Activities For Enhancement) method, which describes the risk management process outlined in the following figure.



According to this approach, risk management is an iterative process performed throughout the project, although it takes on particular importance at the work's initial stages. The phase in which the specific causes for risk are diagnosed is followed by a phase to assess its size, quantified using appropriate working techniques. Diagnosis is followed by prognosis, consisting of a risk management plan containing actions to prevent, monitor, and combat the individual risk factors that have been identified. The SAFE method makes it possible to pre-assess the quality of the risk management plan. This is quantified in a risk removal effectiveness index.

Adopting this and other possible risk management processes helps make the factors on which the planning interventions' success depends clear, communicable, and subject to influence. It therefore constitutes the foundation for any initiative to mature the organization's software development process.

We will focus now our attention on the requirements management and software estimation processes, and then present an integrated framework of reference for effectively interrelating these two aspects.

2. Requirements of a software project

Many studies have shown that the percentage of projects that come in late or over budget, or that fail to meet stakeholders' expectations, is quite high. According to a report by the Standish Group in 1997, Fortune's leading 1,000 firms spent, in 1996, 53% of applications development resources on projects that were technical failures or that required many more resources than initially provided. Other research shows that errors in requirements are not only more expensive, but more frequent as well. Removing an error committed in the user specifications definition phase can cost up to 20 times more than removing one made in the software development phase. The reworking needed to offset imprecise requirements definition often takes up more than 40% of a project's budget. These data point to a problem that although highly serious, can be transformed into an exceptional opportunity: by improving requirements management activities, we can drastically reduce failure, as well as the costs associated with our organizations' software development projects.

2.1. What "requirements" are

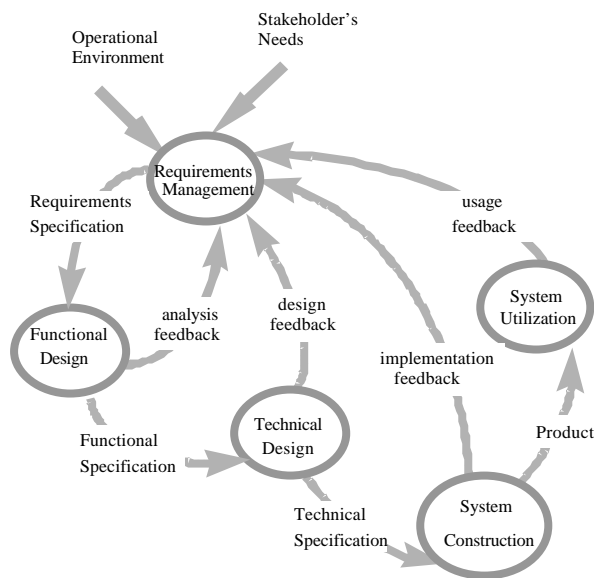
But what is meant by "requirement?" And how can requirements best be managed? Basically, a requirement is a demand expressed and accepted by the project's stakeholders as to the function of the system being developed, or the behaviour of the project itself. The former case is a technical type of requirement, and the latter is an operational type of requirement. The technical requirement may be functional – regarding behaviours expressed by the software or the data on which it operates - or non-functional – regarding the quality or performance to which the functional requirements must be held. Operational requirements, on the other hand, may regard the working process, tools, technologies, or resources to be used. Most projects' problems are caused by failure to define the technical requirements. The term "requirement" is often used to indicate highly different levels of detail of the project specifications. Requirements range from purely textual ones at a highly general level, to those expressed graphically in semi-formal representation models; from structured ones formulated using formal procedural or non-procedural languages, to prototypes developed using visual languages. In the technical literature, however, most of the time the term "user requirement" is understood as a formulation in natural language of the functional and non-functional characteristics that the system must prove that it possesses in order to satisfy the stakeholders' expressed and implied needs.

Requirements for a software project may be found in any kind of document, such as organizational manuals, marketing documents, product specifications, business rules, functional specifications, quality plans, etc. In general, they will be numerous and cumbersome, with many mutual interrelations. Above all, they will be unstable over time. Despite suffering from a variety of flaws, requirements are the project's most important asset, because they constitute the ring joining the stakeholders' needs and perceptions to the technological solutions designed by the software experts.

Managing requirements means operating systematically to identify, organize, document, communicate, and maintain over time the changeable requirements of a software application. Using standard methods and tools to manage requirements makes it possible to increase the process's effectiveness and efficiency, improving the ability to establish the right goals for project organization, and reducing the impact of changes required during the work, thanks to the ease in understanding the consequences that these changes may have on the general design.

2.2. Managing requirements: the ASK method

Like risk management, requirements management is also a process that accompanies the entire duration of the project, although it is of especially decisive importance in the initial phases of the project's lifetime. Requirements come into sharper focus over the life cycle, and at the same time are better specified by developing in directions that are only outlined at the outset. They may be joined by new requirements, substantially modified or even abandoned, if necessary. The figure shows the relationships between the main phases of a project and the requirements management process.

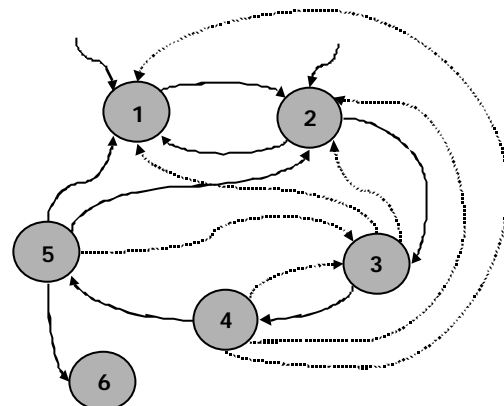


Given that requirements can influence the resources needed for the project's development, requirements management is a double-edged process: technical and operational. In general, varying requirements leads to changing not only the technical nature of the system being developed, but also the allocation of the

resources needed to conclude the project.

The ASK (Applying Stakeholder's Knowledge) method is a requirements management model that includes the following activities:

1. Identifying the stakeholders
2. Gathering problems and needs
3. Preparing and documenting requirements
4. Checking the requirements
5. Negotiating the requirements
6. Accepting the requirements



The above figure shows the work flow between these activities. Broken lines show possible although infrequent transitions, and unbroken lines represent the more probable transitions and the process triggering points. Each of these activities is described in greater detail in terms of input, output, method, and working techniques. The skills to be developed in order to properly manage the project's requirements belong both to the technical and to the behavioural domain. They range from problem solving and brainstorming to techniques for developing statistical studies, from communicative to psychological skills, and from the ability to conduct role play to that of managing questionnaires and interviews.

In properly defining requirements, it is essential from the initial phases to recognize and involve the stakeholders who will benefit from or be disadvantaged by the project's outcome. In this way, the risk will be diminished of having to call into question a project's purposes and expected results every time new people come into the picture, thereby also avoiding having to recycle work due to misunderstandings or simply the psychological detachment perceived by those excluded from the project group. Typical stakeholders to be considered include customers, direct and indirect users, development participants, operating representatives, systems, management, and external regulators

(various kinds of authorities that do not take part in the project, but are interested in compliance with preset external constraints).

One highly effective technique to support the gathering and verification of requirements, called Viewpoint, is described in [4]. To prepare and document requirements, a determining factor may be that of using a software tool capable of automizing the more repetitive parts of activities and of making it possible to group, extract, and intersect the most varied types on these requirements. In fact, interrelation matrices will serve to indicate cases of inconsistency and redundancy among the requirements, allowing them to be solved before they impact the software's development, with the corresponding recycling of work.

Methodical requirements management can therefore make it possible to considerably cut into a project's risks, reducing them significantly.

3. Estimating a software project

As discussed above, when a software project is late and over-budget in providing the expected results, the problem may lie in the inadequacy of either the management process or of the resource assignment process. Determining whether either of or both these possibilities are true is always an extremely difficult operation. Certainly, there are many collateral indicators that can provide helpful hints as to the quality of the management process and the estimation capacity. For example, the use of advanced project management techniques and methods, the use of standards, or the level of group motivation may provide indications as to management quality, while the use of modelling consolidated in the technical literature, or of tools of a proven effectiveness calibrated on reliable benchmarking databases may be able to suggest the level of quality of the estimation process. These indicators may swing the needle of the scale from one side to the other, but the question as to what the cause and effect may be of not complying with the resource constraints tends to look too much like a loop: a project with few assigned resources tends to overheat just like an out-of-phase engine, and this tends to lead to abandoning mature labour practices and standards in favour of approximating activism and anxious attitudes which, in turn, tend to waste human resources. The project gets increasingly close to the point of no return, where everyone tries to abandon ship while there is still time to do so with honour. The same thing may happen if, vice versa, resources are sufficient but the project is poorly managed. The same overheating spiral described above is triggered as soon as the project's "fat" reserves are consumed without producing appreciable results. This behaviour is all too frequent, as shown by the project accounts discussed in the statistics gathered in the field. Therefore, to reduce the risks of failure due to these causes, it is necessary to act to improve the systems and capacities of both project management and of quantitative estimation; the latter is an important subset of the former.

3.1. What is to be estimated?

To correctly assign resources to a software project, the main variables to be estimated are costs, working efforts, deadlines, and the "amount" of software to be developed. These variables are not independent of one another, but are highly related. In fact, a project's costs are basically working costs, and therefore highly dependent on the unit costs of the professional resources used and on the amount of working effort needed for each professional resource. Deadlines also depend on the amount of work to be delivered and the number of resources that can be used in parallel. Lastly, work effort per professional resources depends on the amount of product to be developed, and naturally on the productive capacity that the project group manages to put into play. Based on this

chain, the amount of software to be developed is assessed. Error in this estimation spreads like an oil spill to all the others.

Currently, the choice is widely being adopted around the world to measure software based on the functionality it offers its users, which is to day, in the final analysis, based on its use value. The most widespread functional metrics is certainly IFPUG Function Points [5], whose reference document provides a set of rules for counting software applications, whether newly developed or enhanced. The main problem of Function Point Analysis is that, in order to apply the standard practices provided for in the reference manual, it is necessary to explain and document the detailed functional specifications of the software application being measured. In other words, it is necessary to know all the output traces, input masks, and the structure of the logical archives down to the elementary field level. This detail is almost always non-existent at the moment of the decision as to assigning resources to a project. In lucky cases, this decision is made after a more or less accurate feasibility study. Therefore, we have the so-called estimation paradox: “An estimate is most needed when we lack the elements to perform it (project startup), and when we are able to do it with absolute accuracy (project release), we no longer need it.” The **Early Function Point Analysis (EFPA)** technique was created to solve this dilemma.

3.2. Early Function Point Analysis

The Early Function Point estimation method was developed with the planning constraint of having to maintain complete compatibility with IFPUG standard practices: EFPs are not metrics alternative to Function Points (that is, a variant), but an early indication of the Function Point count that would be obtained for the project or application being measured, if we had the right level of detail (or enough time) to perform the standard count. Early Function Point Analysis [6] [7] [8], a public domain method, has been presented at a number of international conferences, and successfully used by many organizations. It focuses on some fundamental principles:

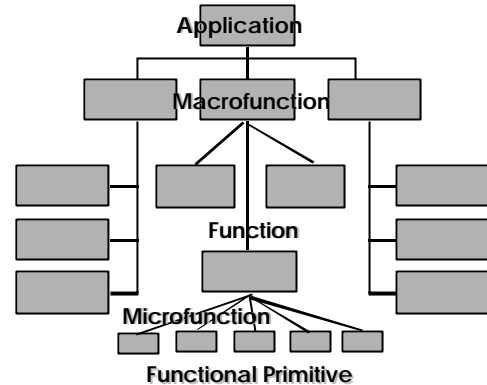
- **classification by analogy:** based on the similarity between new software objects and already-known software objects;
- **structured aggregation:** based on uniting a certain number of objects in one level into one object of a higher level;
- **function/data approach:** no hypotheses on the empirical relationship between data and functionality (how many elementary processes there are, on average, for each archive) are needed, because both components may be assessed autonomously;
- **multilevel approach:** “If you have details, don’t discard them; if you don’t have details, don’t invent them!”
- **use of a derivation table:** each software object at the various levels of detail in the classification taxonomy is assigned a FP value, which is its contribution to the estimation, based on an analytically derived table.

Therefore, the method is based on analogy and analysis. The former leads the estimator to discover similarities between a new “piece” of a software application and similar “pieces” encountered in other software applications and already classified with respect to how the functionalities or data provided for by the method are aggregated. The latter guarantees a certain grounding for the estimate, since the weights of the various software objects are not assigned based on empirical considerations regarding the data gathered by actual projects (and therefore subject to calibration and variation), but are established conceptually, as they are connected with the way in which the various software objects in the classification structure are constructed.

Logical Data Groups (LDGs or just LDs) correspond with logical files in IFPUG Function Point Analysis, but without the distinction between internal and external logical data (ILFs and EIFs), and with two additional levels of complexity. In fact, when information has a low degree of detail, a logical archive identified in an initial instance as unique may later break down into two or more distinct logical archives.

Functionalities may also be identified at different levels of detail or aggregation. A functional primitive is the “smallest correlated set of actions that is self-contained and meaningful to the expert user.” Functional Primitives, if properly identified, correspond with FPA’s Transactional Function Types (EI, EO, EQ). They break down into: input primitives, output primitives, and dialogue primitives.

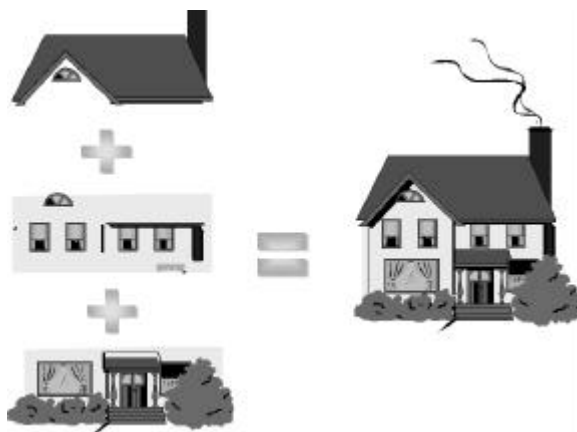
Micro-functions and functions are aggregates of primitives. Macro-functions, on the other hand, are aggregated into a number of functions. The method provides the criteria to appropriately classify functional aggregates based on their forecasted internal numerosity.



To obtain an estimate of the FP count of the software application to be developed or enhanced, it will be enough to have a list of functionalities and data to be implemented that may contain non-homogeneous levels of detail. Knowledge of similar software objects (functions and data) will make it possible to assign the right level of classification to each element on the list, and therefore to derive the FP contribution. Each contribution provided by higher-level aggregated structures is associated with a margin of uncertainty that takes into account the forced lack of detail. Uncertainty will be greater for the higher levels of aggregation, and expressed with a trio of values: minimum – more likely – maximum, with reference to the FP estimate.

If, based on the historical analysis of a sufficiently long set of estimations, a systematic error should be found (over- or underestimate), an additional multiplicative adjustment may be considered to obtain the proper FP count.

To put it extremely briefly, the method’s innovation is to work on recognizing not only the elementary bricks underlying the construction (EI, EO, EQ, ILF, and EIF), but also the pre-fabricated structures on a higher level (macro-functions, functions, micro-functions, multiple archives).



The findings available so far indicate that the method’s accuracy is good, and the average error is held below 10%, thereby permitting above all up to a 4/5 savings in effort over a standard count. In fact, the EFPA method can be used not only when details for the standard count are lacking, but also when we simply cannot or do not wish to commit the resources needed to apply the complete method.

4. An integrated risk, requirements, and estimation management process

If an improvement action performed separately in the areas discussed above guarantees results that are already significant for reducing project risk, a combined action synergistically involving both aspects yields even greater benefits. As shown above, there is a close mutual influence between the quality of the process to define project requirements and the estimation of necessary resources.

This final part of the document is therefore dedicated to describing a framework of reference to integrate the processes of SAFE risk management, of ASK requirements management, and of EFPA early estimation, which has been operatively trialled using the commercial tools Requisite Pro from Rational Corporation, and SFERA from D.P.O. Srl.

The underlying idea was that proper requirements definition and management performed right from the initial project phases, plus the ability to quickly estimate resources based on these requirements, could make it possible to more quickly and effectively hold our course, making the appropriate manoeuvres to avoid the shoals that can run the project aground for lack of resources. The link between these processes is simple and immediate. If each functional requirement were to be given a classification based on the taxonomy provided for by EFPA, we could easily use this information to generate a FP estimation of the required application, and from there proceed to assess the times and costs needed for the project. This will make it possible to allocate resources consistently with what has to be done, thereby reducing the risk of failing to comply with the established constraints. On the other hand, a variation in the requirements may be quickly analyzed with regard to the impact that it has on the project, highlighting the need to assign it more resources, for example. Should, despite indications that the budget needs to be expanded or the deadlines extended, the decision be made to maintain the original ones, the techniques will enable us to reasonably assess the increased risk. This will be described in the final paragraph.

4.1. Evaluating the impact of changed requirements

How can a reasonable assessment of the impact caused by modifications to a project's requirements be made when they are required during the work? Joint use of the techniques described in this presentation, along with a work progress evaluation technique typical of project management, called Earned Value, may yield - and has already yielded- interesting results.

It must first be specified that this work does not intend to deal with the technical aspects regarding changes in structure or function of the software application, but with the consequences in terms of resources use and project risk that the variation demands may bring.

Let us then assume that, after an initial definition of the requirements for a software project, the functionalities and data are classified using the EFPA method. The result of applying this technique is therefore a Function Point estimate of the functional size of the software application to be developed. Using appropriate productivity and cost models connected to the specific development environment, we can forecast the project's effort, duration, and cost. Budget and deadlines can then be assigned in line with requirements, and the project can start on its way.

However, after a certain period of time, the stakeholders require important changes to the requirements, which necessitate revising both the functional and technical design and the project's work plan. At this point, how are we to evaluate whether resources have been correctly allocated or, if the original budget is maintained, whether the associated risk is increased or diminished?

4.2. First option: budget revision

Having established an automatic link between requirements and EFPA method, it will be easy to obtain a new Function Point estimate for the new requirements configuration. Using this estimate

with the same productivity models employed before, we obtain new time, effort, and cost forecasts. These forecasts, however, will regard a project as if it were starting from zero. Actually, since the requirements were modified while the work was in progress, the project has already committed a portion of those resources and yielded a portion of its results. The problem, therefore, is to obtain an assessment of the project's so-called "forecasts to complete," which is to say an estimate only of the resources needed starting from the moment the variation is made. The Earned Value (EV) technique is used to obtain this result.

EV, a value that can be calculated at any time in the project's lifetime, is a cumulative indicator, which means that it outlines within itself the project's past history. It integrates the measurements of purpose, cost, and scheduling. Its proper definition is as follows: Earned Value is the budget value of the work actually accomplished at the date of reference (generally termed Time Now – TN). To use Earned Value properly, we must introduce the following terms [9]:

- The budgeted cost, also called the budgeted cost of work scheduled (BCWS), is that portion of the approved cost estimate planned to be spent on the activity during a given period.
- The actual cost, also called the actual cost of work performed (ACWP), is the total cost incurred in accomplishing work on the activity during a given period.
- The earned value, also called the budgeted cost of work performed (BCWP), is a percentage of the total budget equal to the percentage of the work actually completed.

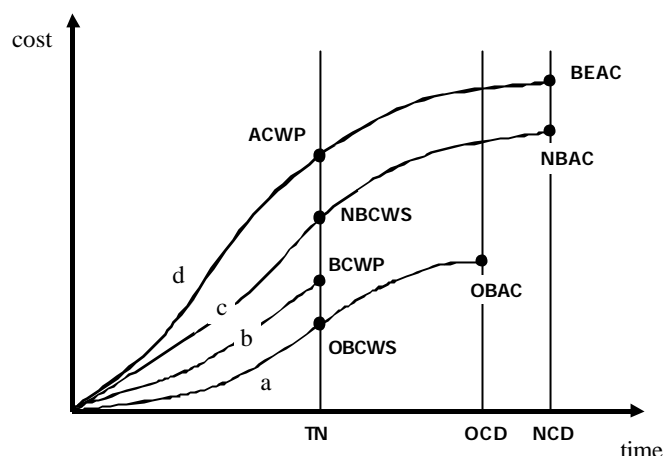
These three values are used in combination to provide measures of whether or not work is being accomplished as planned. The most commonly used measures are the cost variance ($CV = BCWP - ACWP$), the schedule variance ($SV = BCWP - BCWS$), and the cost performance index ($CPI = BCWP / ACWP$). The cumulative CPI (the sum of all individual BCWPs divided by the sum of all individual ACWPs) is widely used to forecast project cost at completion. In some application areas, the schedule performance index ($SPI = BCWP / BCWS$) is used to forecast the project completion date."

In substance, EV or BCWP represent the amount of resources in the original plan that have theoretically been consumed because they regard activities already performed or under way, regardless of how many resources that have actually been consumed, that is of the actual data. Comparing the BCWP and ACWP may permit us to know, in brief, whether the project is spending more or less than forecast, accomplished work being equal. On the other hand, comparing BCWP and BCWS may enable us to know, in brief, whether the project is early or late with respect to the original plans. It is not our purpose to illustrate this technique in detail, which can be done elsewhere. The important thing is to observe that if we remove the BCWP calculated at the moment of revision from the new estimate we have obtained following the variation of requirements (New Budget At Completion: NBAC), we will have a forecast "for completion" for the resources needed for project, which may be "corrected" with an indicator of the degree of efficiency reached up to that moment ($ACWP/BCWP$). In effect, we must also consider that, of the work already accomplished (BCWP), a portion may still be valid even after the change in requirements, while another will be entirely useless because it will regard requirements that have been abandoned or modified. This can be taken into account by multiplying the BCWP by a percentage, R, which indicates the possible degree of reuse of the work already accomplished. To estimate R, we may ask ourselves the following question: "How much of the work already accomplished can be reused even after the requirements are changed?" If we can give this question a percentage value (for example, we believe

that the work is 80% recoverable), the value to be subtracted from the new total forecast to obtain the final forecast will be 80% of the BCWP.

The following graphic may be of assistance for illustrating the proper analysis procedure. The symbols used in the graphic are as follows:

a is the curve for absorbing resources provided for in the original plan; b is the earned value curve; c is the curve for the new budget assessment after the requirements are changed; and d is the curve for the budgets and forecasts for completion.



- ACWP represents the expenses actually incurred up to TN.
- NBCWS (New Budgeted Cost of Work Scheduled) is the new forecast of resource use at TN – that is, after the requirements are revised.
- BCWP represents the Earned Value calculated at TN, with reference to the original plan.
- OBCWS (Old Budgeted Cost of Work Scheduled) is the original forecast of resource use at TN.
- OBAC (Old Budget At Completion) is the total expense forecast of the original budget.
- NCD(New Completion Date) is the new project completion date.
- OCD (Old Completion Date) is the old project completion date.

Now then, the final forecast (Budget Estimated At Completion: BEAC) is given by the following formula:

$$BEAC = ACWP + (NBAC - BCWP * R) * (ACWP / BCWP)$$

The budget may then be suitably revised upon implementing the new requirements and for the productive behaviour shown up by the project up to that point.

4.3. Second option: risks revision

What happens if the budget or deadlines cannot be modified, and we wish to maintain the validity of the initial cost forecast (OBAC) even while accepting the changed requirements? Let us imagine a fixed-payment tender, for example. In this case, we shall have to consider how the project's risk will be influenced.

In effect, the quantity (BEAC-OBAC) represents the percentage of the new budget that should be compressed by a demand for greater efficiency (with respect to average productivity) from the working group. We may then analyze the riskiness linked to failure to meet the original budget in the following manner.

First we assess the damage caused by failure to meet the project's budget on a scale of 1 to 10, in which 10 is project failure and 1 a slight inconvenience. This assessment may vary, since there are projects that must obligatorily be performed, and for which meeting the budget is important but not vital, and other projects that can be cancelled in the event of excessive consumption of resources.

For the sake of simplicity, let us assess the likelihood that the project will manage to meet its budget, considering that it is:

- directly proportional to the remaining duration of the project $P1=((NCD-TN)/NCD)$ – the more time there is to recover, the higher is the likelihood of managing to do so;
- directly proportional to the ratio between tied up and forecast budget, $P2=(OBAC/EAC)$ – the higher the ratio and lower the costs are to be recovered, the easier it will be to do so;
- directly proportional to the Cost Performance Index (CPI) – the higher the CPI and the more efficient the project has been to that time, the higher is the likelihood that it will recover.

If these indices are all lower than 1, we may think of adopting their numerical product $Pyes=P1*P2*CPI$ as the value of the likelihood of managing to maintain the budget. The opposite event, that is failure to respect the budget, will then be: $Pno = 1 - Pyes$.

The risk associated with the event – the budget not having been met – is therefore the product of the score on a scale of 1 to 10 attributed to the damage and the probability Pno . After this assessment, we may decide on an appropriate action plan consisting of prevention, monitoring, and combat, as provided for by the SAFE method.

5. Conclusions

This work has examined three fundamental processes that may greatly contribute towards reducing a project's risk, particularly if they are made to interact with one another in a coordinated fashion. We have presented the SAFE risk management methods, the ASK requirements management methods, and the EFPA Function Point estimation methods. Lastly, an example has been provided of using these techniques to assess the impact of modified requirements on the budget and on the project's risk.

6. References

Some of the papers referenced may be downloaded from:

<http://web.tin.it/dpo> or <http://www.dpo.it>

- [1] Gluch, David P., "A construct for describing software development risks", Technical Report CMU/SEI-94-TR-14, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, 1994
- [2] Meli, Roberto "SAFE: a method to understand, reduce, and accept project risk", ESCOM-ENCRESS 98 - Project Control for 2000 and Beyond - May 27-29, 1998 - Rome, Italy.
- [3] <http://www.lvdg.com/gma/gmagopp.htm>
- [4] Sommerville, Ian and Sawyer, Pete, "Requirement Engineering: A good practice guide", John Wiley & Sons, 1997 –Baffins Lane, Chichester, West Sussex England
- [5] International Function Point Users Group: "Function Point Counting Practices Manual", Release 4.0, 1994
- [6] R. Meli, "Early Function Points: a new estimation method for software projects", ESCOM 97, Maggio 1997, Berlino
- [7] R. Meli, "Early and Extended Function Point: a new method for Function Points Estimation", IFPUG Fall Conference, 15-19 settembre 1997, Scottsdale, Arizona USA
- [8] L. Santillo, R. Meli, "Early Function Points: some practical experiences of use", ESCOM-ENCRESS 98, Maggio 1998, Roma
- [9] PMI Standards Committee, "A Guide to the Project Management Body of Knowledge", Project Management Institute, Upper Darby, PA 19082 USA