

## ▶ Estimating use-case driven iterative development for "fixed-cost" projects

by [Amit Bhagwat](#)

Clearvision CM plc

*Most project-management-related equations treat the project resource requirement as an essential variable. Some of these equations simplify the concept of resource to project budget that can "buy" and therefore substitute, for purpose of calculation, all other resources. The equations lead to a general understanding that the resource requirement for a project will increase -- and, therefore, the cost of the project will go up -- if its coverage, complexity, or quality<sup>1</sup> expectations increase. Alternatively, if the number of features to be implemented and / or their complexity increases through lack of early comprehension or because of changing goals, then the only way to maintain cost is to give quality a good thrashing. Of course, compromising on quality is not an attractive option.*



*Conclusion -- unless we have a project with high stability in terms of both requirements and complexity, there is no way that we can accurately predict its cost, assuming a certain quality of its output. This is the "fixed-cost project paradox." Put simply, fixed-cost projects do not work well for dynamic business system development.*

*Yet, the fixed-cost method is often the preferred choice of project clients, because it allows them to make fixed investments upfront, with a mirage of returns on the horizon. This is even simpler for contractor-delivered projects, in which contractors work out the cost and all the sponsor does is to pick the proposal that appears to be most applicable and to deliver "best value."*

*In this article, I will first recap a well-accepted use-case based estimation technique, then describe a converging range estimation technique, applied*

▶ [subscribe](#)

▶ [contact us](#)

▶ [submit an article](#)

▶ [rational.com](#)

▶ [issue contents](#)

▶ [archives](#)

▶ [mission statement](#)

▶ [editorial staff](#)

*within an iterative development scenario. The two techniques are related in that the use-case based requirement management approach provides logically related requirement packets for implementation in an iterative scenario. Please note the following:*

*1. The concept of converging prototypes is useful not only for use-cased based projects, but also for projects that modularize, trace, and gradually elaborate functional requirements by any other techniques.*

*2. The section "Use Case based Effort Estimation" should seem fairly straightforward to those who have been using use cases for a while.*

*3. If you are new to both use cases and iterative development, if you are using a requirements management technique that modularizes requirements so as to allow iterative implementation, and you would like to venture into planning iterative development without learning the details of the use-case technique, then you may bypass the "Use-case based effort estimation" section, and treat any reference to use cases in later sections as a reference to "modularized requirements."*

## **Project management realities**

Project managers required to work on fixed-cost projects usually employ some kind of safety net to keep the project in line with estimations, without which the project is guaranteed to be executed predominantly by lawyers rather than by developers. They make a detailed list of all features upfront. They then draw a fixed price contract against this list with the caveat that any change to the requirements will have dire consequences regarding cost. Next, they embark on the "80% work," often smoothly, until some change to business objectives, through changed perception or need, requires certain features to be modified. All too quickly the project deadline draws near, and the choice seems to be between 1) continue the work indefinitely, or 2) deliver a system that barely stands on its feet. This is the classic problem with traditional sequential, or "waterfall," project management, well illustrated by independent industrial research, the oft-quoted CHAOS<sup>2</sup> report for example, which puts success rate of typical IT projects well under 30%, with very liberal success criteria.

Anticipating these inevitabilities, the use-case driven iterative approach to software project management offers a better alternative for keeping projects on track. It assumes that requirements are seldom understood completely at the beginning of a project, and that the only certainty in business system requirements is that requirements change. The use-case approach, therefore, allows the detailing of only certain requirements to be implemented for each iteration. This brings down project risk earlier than in waterfall development practice, while allowing teams to detail other requirements for implementation in subsequent iterations.

However, the use-case approach can also mean that the software development organization and the client do not have a detailed description available upfront for all requirements, which means the so-called "accurate costing" is missing from the equation. Many clients are uneasy about this presumed lack of accuracy, and if they are suspicious about the abilities of

the software development team, they may discourage the iterative approach for this reason. Pressured by competitors who claim to offer accurate estimates upfront, a software supplier is tempted to propose a business system project estimated on ideal assumptions with limited upfront feel for the completed system. This is a recipe for disaster.

On the other hand, if the client has some faith in the supplier, or if it is willing to recognize upfront the realities of typical business system projects, then the use-case driven iterative approach can work splendidly. It can provide a much higher rate of success and virtually eliminate complete failure. Yes, the client needs a number to be able to make budgetary provisions, but the client also accepts the following:

1. Not all requirements are known in their entirety.
2. Certain requirements are architecturally more significant than others.
3. Certain requirements are more volatile than others.
4. If requirement depiction is easily understood by the system user, it is more likely to be accurate and leading to a user-acceptable system.

This form of understanding is ideal ground for adopting the use-case driven iterative approach. I have observed the successful, practical implementation of this approach in "larger than pilot" (20 person-month +) projects. The projects borrowed heavily from earlier works related to use-case based estimation and coupled them with the concept of converging estimates.

## **Use-case based effort estimation**

For those who are new to use-case based estimation, I recommend John Smith's paper "The Estimation of Effort Based on Use Cases."<sup>3</sup> Not only does it offer a concise description with a set of formulae, but it also serves as a reference to work by others in this area. Yet, Smith's paper may be considered heavy going for the uninitiated. For this reason, I will provide a few basic principles, formulae, and guiding data from this work (some with modest empirical support, as acknowledged). This summary will give the reader an introduction to use-case based estimation. For those of you who are more curious, I recommend that you consult Smith's paper and the sources listed therein.

A structural hierarchy exists in a software solution, consisting of the following levels:

- 4 - System of Systems
- 3 - System
- 2 - Subsystem Group
- 1 - Subsystem
- 0 - Class

Typically, use cases exist at all levels, except, in most cases, level 0 (i.e., individual classes). Use cases at different levels differ in complexity both as written and as realized. Typically, we do not need hundreds of use cases to describe behavior at any given level. The general approach is not to have premature decomposition for high-level use cases without

adequate architectural work and a model to support it. The method determines typical multiplication factors between adjacent levels in the hierarchy, as well as some other quotients, mainly from practical experience. These are shown in Table 1:

**Table 1: Typical hierarchy of system complexity.**

Operation size	70 slocs
Number of operations per class	12
Number of classes per subsystem	8
Number of subsystems per subsystemGroup	8
Number of subsystemGroups per system	8
Number of systems per systemOfSystems	8
Number of external use cases (per system, subsystem, etc.)	10
Number of scenarios per use case	30

I have personally found these multipliers quite acceptable. They make it possible to arrive at a reference range, or number, for effort per use case. For example, in a simple system of 10 external use cases at each level, assuming other cost drivers to be minimal, this approach arrives at the numbers in Table 2, using Estimate Professional (a tool by Software Productivity Center based on COCOMO II<sup>4</sup> and the SLIM model):

**Table 2: Estimation based on 10 external use cases at each level.**

**Note: "sloc" = "source lines of code."**

[Click to enlarge](#)

Size (slocs)	Effort hrsluse case simple business system	Effort hrsluse case scientific system	Effort hrsluse case complex command and control system
7000 (L1)	55 (range 40-75)	120 (range 90-160)	260 (range 190-350)
56000 (L2)	820 (range 710-950)	1700 (range 1500-2000)	3300 (range 2900-3900)
448000 (L3)	12000	21000	38000
3584000 (L4)	148000	252000	432000

However, we must acknowledge that life is seldom this easy. For example, there may be use cases whose levels are fuzzy -- those, for instance, that are biggish for a subsystem but smallish for a subsystem group. Thus, the approach prescribes overlapping bands in terms of SLOC for permissible size of use cases implemented at each level. In the region of overlap, we work out the position of the use case relative to the typical middle value for use cases in the levels above and below it. We can therefore say that the use case is at fractional distance  $D$  from the next lower level (and therefore  $1 - D$  from next higher level). In such cases it can be represented as  $N$  use cases at the lower level, where<sup>5</sup>

$$N = 8^D$$

It is recommended that a complexity multiplier be introduced, based on the project's specific average use-case length at a given level, as compared to the typical length of a use case at that level.

There is also a correction applied to account for overall effort inertia -- i.e., size of effort relative to the reference point (one described in table above which assumes 10 external use cases at each level). Based on the COCOMO model, with nominal scaling for size S for actual effort and  $S_R$  for a reference case, the effort per use case scales<sup>6</sup> by  $(S / S_R)^{0.11}$ .

Therefore, the effort inertia scaling factor works out to: (sum of size of effort at each level relative to the reference value at designated level) <sup>0.11</sup>

The effort estimate is therefore the sum of effort estimates at each level, which is the sum of product of: number of use cases at the level relative to reference, use case complexity multiplier (average use case complexity at the level relative to reference), the effort inertia scaling factor for the level, and the effort for reference case at that level.

## **Difficulty**

This approach acknowledges that, if we decide to use iterative development process, we must maintain some latitude for correcting the estimate at the beginning of subsequent iterations as the architecture evolves and crystallizes. However it also assumes that a substantial amount of use case writing takes place upfront and that surprises and uncertainties originate predominantly from lack of architectural clarity. Therefore, this approach does not highlight lack of requirement clarity, and more importantly the effect of changing requirements, except to acknowledge that the accuracy of the whole estimation process relies on the estimators' familiarity with the project domain.

One benefit of this process is that it does give sponsors a number that good project teams can realize, provided requirements and architectural evolution do not have many surprises. The drawback is that too many business systems projects suffer from changing requirements; the approach of the use case writers is therefore to state virtually all use cases upfront and then to elucidate them over iterations. Thus, use cases are completed (in every respect, naturally including all scenarios) in an iteration prior to realizing them in a model; this may happen, typically, one iteration prior to implementing and unit-testing the features related to them. In any case, a use case is effectively frozen only when it is realized in a low-level model, typically in an iteration prior to implementation. For these reasons, we must acknowledge that use cases that are assigned to be implemented in later iterations are much more prone to change in dynamically evolving systems.

A method based on early requirements freeze does not set a buffer upfront that dictates the flexibility / resilience of the project. Such a method therefore leaves little room for requirement changes or late discovery of requirements. This means that, while requirements inevitably change given the dynamic nature of the domain, as do the estimates if recalculated at a later stage, the estimates may not follow a set pattern, leaving the client uneasy throughout the project.

## **Solution**

The way out is to acknowledge that the stability of the estimate for use cases implemented in subsequent iterations is low, that the estimate is not a golden number, but rather a conclusion arrived at assuming things will go smoothly. But what if things go badly -- so bad, for instance, we begin to doubt the validity of our use cases? This sort of thing can happen in dynamically evolving business systems. It seems, then, that we are actually in search of two numbers: one for smooth sailing, and another for when the seas grow rough and we consider abandoning ship.

Let's imagine that the smooth sailing estimate, arrived at by the method described above, is  $X$ ; however, given the instability and low comprehension of use cases implemented in subsequent iterations, there will be  $\Delta X$  of additional effort involved.<sup>7</sup> Not only does this give the client a range of estimate between  $X$  and  $X + \Delta X$  and which converges towards a narrower range at the beginning of each subsequent iteration, but it also gives a more direct criterion for the client's acceptance of the chosen iteration plan. This is because, the values of  $X$  and  $\Delta X$ , viewed from another perspective, assure the client that the project will deliver  $(X/(X + \Delta X))$  features, within resource  $X$ , in the worst-case scenario. The project can therefore be considered completed within the estimate  $X$  (i.e., in the "fixed-price"), in virtually all cases, if a fraction  $(1 - (X/(X + \Delta X))) = (\Delta X/(X + \Delta X))$  of features to be implemented<sup>8</sup> represents optional features that can be dropped, still resulting in an acceptable end system. This controlled approach requires labeling expendable features upfront, rather than scrambling at the point of crisis to throw precious cargo overboard, in order to save the ship. The approach also allows restructuring the iteration plan, whereby, among other changes, a different number of use cases may be detailed and effectively frozen upfront, in order to bring down  $\Delta X$ , at the cost of sacrificing greater choice and flexibility in feature implementation at later stages.

## **Finding the effort range for iterative development**

Let me admit here that, since the influence of project variables is different at different times through a project, we do not have a single formula that plots  $X$  against  $I_i$ ,  $I_i$  being the  $i^{\text{th}}$  iteration. Indeed, there have been few systematic attempts towards trying to arrive at range estimates. Perhaps the most noteworthy among these is Boehm's work on the COCOMO II model.<sup>9</sup> The general approach in this model is to try to find as many factors as possible that contribute to overall effort estimate, and divide these factors among:

1. Factors contributing to size
2. Factors linearly varying the estimate for a given size
3. Factors varying the estimate exponentially with respect to the size, thus contributing to economies / diseconomies of scale

The estimation model is then divided into three temporal zones running sequentially along the project timeline. The model to be applied in the first zone is the Application Composition model, characterized by linear scaling. This is followed by Early Design and Post Architecture models with seven and seventeen effort multipliers identified, respectively. The approach

analyzes empirical data collected from multiple sources over several years, in order to ascertain the value range for the multipliers and exponents. The approach predicts  $\Delta X$  as high as  $3X$  at the beginning of the feasibility phase, in the case of software development projects yielding unprecedented products in novel domains.  $\Delta X$  tends to be  $\sim X$  after the feasibility phase is completed and the basic architecture has been visualized. Thus, for projects related to dynamically evolving business systems in familiar or mature domains,  $\Delta X = \sim X$  near the beginning of the project. As the gap between base and pessimistic estimates continually reduces, the  $\Delta X$  tends to reduce at an increasingly slower rate as the project progresses. This may however be approximated to linear reduction beyond the point where the basic architecture is determined and major implementation work begins. This is the area where architectural implementation, detailed design, and complete implementation occur, iteratively and incrementally in our case. This is also the region where "requirements creep" occurs the most, without being declared disastrous.

In the paragraphs that follow, we will concentrate on this area of the project plan, working with variables and evolving them into estimation formulae. This mathematical treatment is inescapable for a project estimator and is indeed our goal. I realize that some readers find formulae tough to work with, so you may want to take a break and return to this discussion with a fresh mind.

We will first establish the relationship between certain estimation variables, then produce basic formulae for a simple scenario, work through them with an example, and then expand them to account for more life-like situations.

## **Relationships among estimation variables**

Let's concentrate on typical projects related to dynamically evolving business systems where requirements creep, and not domain unfamiliarity, is the main issue. Based on empirically established work described above, let's define our project characteristics as follows:

1. Implementation approach: iterative, involving nearly same level of effort in each functional iteration
2. Basic effort in each functional iteration =  $X_i$
3. Basic functional implementation effort =  $X$
4. Total number of functional iterations =  $n$
5. Potential aggregate additional effort at any point, due to changing functional requirements =  $\Delta X$
6. Potential additional effort out of requirement creep for functional iteration  $I_i = \Delta X_i$
7. Variation of  $\Delta X$  with respect to  $n$  = linear, reducing
8.  $\Delta X = \Sigma \Delta X_i$

9. We can also reasonably assume that at the point of entering the next functional iteration, requirements for it are frozen. Therefore  $\Delta X_{\text{next}} = 0$

Thus, at the beginning of first functional iteration ( $I_1$ ) we have:

- Functional iterations remaining = n
- $\Delta X = X$
- $\Delta X_1 = 0$
- Therefore  $\Sigma \Delta X_i (1 \text{ to } n) = \Delta X = X$

At the beginning of the last functional iteration ( $I_n$ ) we have:

- Functional iterations remaining = 1
- $\Delta X = 0$
- $\Delta X_n = 0$
- Therefore  $\Sigma \Delta X_i (n \text{ to } n) = 0$

Following the linear relation for reduction in  $\Delta X$  (which is same as  $\Sigma \Delta X_i (i \text{ to } n)$ , for an iteration  $I_j$ ) going through iteration  $I_1$  to  $I_n$ , we observe a reduction in  $\Delta X = X$  through  $n - 1$  steps. Therefore, each step contributes to reduction of  $(X / (n - 1))$  in  $\Delta X$ . In other words, at the beginning of the iteration  $I_j$ ,  $\Delta X$  will be  $(X - ((j - 1) (X / (n - 1))))$   
 $= X (1 - ((j - 1) / (n - 1)))$   
 $= X ((n - j) / (n - 1))$

We have still not deduced exactly how  $\Delta X_i$ , associated with individual subsequent iterations, will change as we progress through each iteration. This is of no significance if we look at the bigger picture. If, however, we need to know the exact level of uncertainty / potential error slice that each of the subsequent iterations is contributing to  $\Delta X$ , then we also need the relationships between various  $\Delta X_i$  values.  $\Delta X_i$  will be "increasing with value of i" at any point in time, since the farther an iteration from the point of implementation, the higher the grain size of the features defined -- and the more abstract are the requirements associated with that iteration. If again, we were to assume a linear relation (increasing at fixed rate) between various  $\Delta X_i$  values, then at the beginning of iteration  $I_j$ , we have  $\Delta X_j = 0$ ,  $\Delta X_{j+1} = Y$  (say),  $\Delta X_{j+2} = 2Y, \dots, \Delta X_n = \Delta X_{j+(n-j)} = Y(n - j)$ . Therefore in general  $\Delta X_i = Y (i - j) = (\Delta X_{j+1} (i - j))$ .

Therefore, at the beginning of iteration  $I_j$ ,  $\Delta X = \Sigma \Delta X_i (j+1 \text{ to } n) = \Delta X_{j+1} \Sigma (i - j) (j+1 \text{ to } n)$

However, at the beginning of iteration  $I_j$ ,  $\Delta X$  is also  $X ((n - j) / (n - 1))$

Therefore,  $\Delta X_{j+1} \Sigma (i - j) (j+1 \text{ to } n) = X ((n - j) / (n - 1))$

Therefore,  $\Delta X_{j+1} = X ((n - j) / (n - 1)) / \Sigma (i - j) \text{ (j+1 to n)}$

## Examples

We have too many variables (X, i, j, n) and greek letters floating around, and many of us are not exactly in love with algebra and statistics. I suppose an illustrative example will go well here. Say our overall estimate is X (the calculation for which is explained in method revisited earlier) and we want to implement it in six functional iterations -- i.e., n = 6. Therefore, for a typical dynamic business system project, with modest novelty presented by the project domain (where we are justified in approximating the relation between  $\Delta X$  and n to linearity), we have:

At the beginning of iteration 1,

$$\Delta X = X$$

Where

$$\Delta X_2 = X ((6 - 1) / (6 - 1)) / (5+4+3+2+1) = X / 15$$

$\Delta X_3, \Delta X_4, \Delta X_5, \Delta X_6$  will be its multiples, i.e.,  $2X / 15, 3X / 15, 4X / 15, 5X / 15$

Therefore  $\Delta X = \Sigma \Delta X_i \text{ (j+1 to n)} = \Sigma \Delta X_i \text{ (2 to 6)} = ((1+2+3+4+5) / 15) X = X$  as expected.

Likewise,

At the beginning of iteration 2,

$$\Delta X = X ((6 - 2) / (6 - 1)) = 0.8X$$

$$\Delta X_3 = X ((6 - 2) / (6 - 1)) / (4+3+2+1) = 2X / 25$$

$$\Delta X_4 = 4X / 25$$

$$\Delta X_5 = 6X / 25$$

$$\Delta X_6 = 8X / 25$$

At the beginning of iteration 3,

$$\Delta X = X ((6 - 3) / (6 - 1)) = 0.6X$$

$$\Delta X_4 = X ((6 - 3) / (6 - 1)) / (3+2+1) = X / 10$$

$$\Delta X_5 = 2X / 10$$

$$\Delta X_6 = 3X / 10$$

At the beginning of iteration 4,

$$\Delta X = X ((6 - 4) / (6 - 1)) = 0.4X$$

$$\Delta X_5 = X ((6 - 4) / (6 - 1)) / (2+1) = 2X / 15$$

$$\Delta X_6 = 4X / 15$$

At the beginning of iteration 5,

$$\Delta X = X ((6 - 5) / (6 - 1)) = 0.2X$$

$$\Delta X_6 = X ((6 - 5) / (6 - 1)) = X / 5$$

At the beginning of iteration 6,

$$\Delta X = X ((6 - 6) / (6 - 1)) = 0$$

Therefore, we see a narrowing of all  $\Delta$  values:  $\Delta X$  and all  $\Delta X_i$  values. We

also know what maximum variation to expect and are prepared for it in the next iteration. In our example, while iteration 2 is being executed, the worst variation that can occur to the estimation of iteration 3 is  $2X / 25 = 8\%$  of overall estimate; likewise, the worst variation allowed in subsequent iterations would be 16%, 24%, 32%, thus allowing a maximum variation of 80% overall.

One iteration later, if everything has gone well in iteration 2, we have the maximum variation to subsequent iterations as 0%, 10%, 20%, 30%, thus down 8%, 6%, 4% and 2% respectively.

If things had gone as badly as realistically acceptable in the planning of iteration 3, the estimate would have crept up from  $X$  to  $X + (2X/25)$ , still within limit and still narrowing the limit in the process. The maximum variation at the beginning of iteration 3 would still have remained 60% of original estimate in total, divided as 10%, 20% and 30% of the original estimate for iteration 4, 5 and 6. These numbers would have made even smaller fractions of the new estimate (108% of  $X$ ), which would have been: 55.55% in total, divided as approximately 9.26%, 18.52% and 27.78%.

The iteration plan is flexible, with built-in resilience for handling changes on the fly, impacting iterations 2, 3, 4, 5, 6 by 6.7, 13.3, 20, 26.7, 33.3 percent respectively of the total estimate.

Viewed from a "fixed-cost" perspective, the plan guarantees success with cost  $X$ , if features corresponding to the  $(\Delta X / (X + \Delta X))$  portion of the estimate are optional features.  $(\Delta X / (X + \Delta X))$  is 50% at the very beginning, reducing to 44.4% by the time requirements for the second iteration are frozen, reducing further to 37.5, 28.6, 16.7 and 0 percent at the beginning of each subsequent iteration.

## Iterations of different size

You may want to know how our equations would fair if iterations happen to have unequal functional coverage. For example, imagine there are  $n$  iterations planned of functional size  $a_1, a_2, a_3, \dots, a_i, \dots, a_n$

The thinking behind the formula

[At the beginning of iteration  $I_j$ ,

$$\Delta X = X ((n - j) / (n - 1))]$$

will still apply, so long as the linearity between  $\Delta X$ , and project progress, which is established empirically, holds reasonably well.

The formula will therefore appear in its more general form:

$$\Delta X = X (\sum a_{i(j+1 \text{ to } n)} / \sum a_{i(2 \text{ to } n)})$$

Or in words, to make it easier to read:

$\Delta X = X$  ((number of estimation units left to be implemented after implementing the current set of segments) divided by (number of estimation units left to be implemented after implementing the first set of segments))

Our other formula:

$$\Delta X_{j+1} = X ((n - j) / (n - 1)) / \sum (i - j)_{(j+1 \text{ to } n)}$$

also assumes a linear increment in  $\Delta X_i$ . This assumption is based on sparse empirical support, and although essentially correct in terms of trend, it may require refinement of the exact computation method based on further empirical work.

If however, we were to apply this formula as-is for iterations of different size, it will look something like this:

$$\Delta X_{j+1} = (X (\sum a_{i(j+1 \text{ to } n)} / \sum a_{i(2 \text{ to } n)})) (a_{j+1} / (\sum (a_i (i - j)))_{(j+1 \text{ to } n)})$$

Which is  $\Delta X (a_{j+1} / (\sum a_i (i - j)))_{(j+1 \text{ to } n)}$

Or to describe this in words:

Estimation error associated with iteration to be implemented after the current one = (Total estimation error after implementing the current iteration) **multiplied by (ratio of ((units to be implemented in next iteration) to (sum of product of units to be implemented in subsequent iterations and their proximity to current iteration)))**.

Likewise:

The estimation error associated with any iteration implemented after the current one = (Total estimation error after implementing the current iteration) multiplied by **(ratio of (product of ((units to be implemented in that iteration) with (its proximity to current iteration)) to (sum of (product of ((units to be implemented in subsequent iterations) and (their proximity to current iteration))))**)).

## Conclusion

We have attempted here, to make iterative development more client-friendly. In most business systems projects, requirements keep changing. The iterative use-case driven approach has emerged to acknowledge that requirements change, and by documenting requirements in a user-friendly manner and implementing them incrementally, risks associated with the outcome are greatly reduced and better managed. Software project clients are sometimes suspicious of this approach, since estimates often change as use cases become detailed and/or change. The approach described above offers a range of estimate that will converge with successive iterations. It also offers a picture of what proportion of features are guaranteed under a fixed budget scenario. In addition, our method quantifies the trade-off between flexibility / resilience in functional

implementation and precision of project estimates.

I encourage you to apply this concept to different sizes and types of projects, to provide additional empirical data, and to deduce more precisely the relation between the level of potential change / additional work in a subsequent iteration and the position of that iteration in project execution.

---

## Notes

<sup>1</sup> It is generally agreed that improving on quality of project execution process will, more likely than not, reduce its cost. In context of our discussion, however, we are referring to variation in quality of project deliverables, under the chosen project execution approach.

<sup>2</sup> Standish Group, "CHAOS: A Recipe For Success," 2001. The point illustrated in this paper is also consistent with earlier and later CHAOS literature.

<sup>3</sup> John Smith, "The Estimation of Effort Based on Use Cases," Rational Software whitepaper, 1999

<sup>4</sup> From the COCOMO II Web site itself you can obtain a basic estimation tool ([ftp://ftp.usc.edu/pub/soft\\_engineering/COCOMOII/cocomo99.0/c990windows.exe](ftp://ftp.usc.edu/pub/soft_engineering/COCOMOII/cocomo99.0/c990windows.exe)) based on that model.

<sup>5</sup> This is not as obscure as it may appear; the base 8 comes from the multiplier of 8 assumed (and generally found acceptable) for the number of units corresponding to a lower level that relates to a unit at the next higher level.

<sup>6</sup> For those who are new to COCOMO and happen to be baffled by the fractional exponent, here's a very brief explanation: We are assuming scaling factors to be nominal; thus involving slight diseconomy of scaling, the model gives an exponent of 1.11. The catch is in the phrase "Scales by" and not "scales to," where the former works out to  $(1.11 - 1 = 0.11)$ . For example, say, X functionality is implemented through Y efforts, but due to a slight diseconomy of scale, 10X functionality needs more than 10Y efforts. To find out this effort value, we first work out the effort as 10Y (multiplying by the number of functional elements relative to the reference), assuming no economy or diseconomy of scaling (i.e., exponent = 1). We then impose a fractional exponent on the size ratio to account for nominal diseconomy. In this case, for example,  $(S / S_R) = 10$ . So assuming the simplest case and allowing scaling by 0.11, we have final efforts =  $((10Y) (10)^{0.11}) = ((10Y)(1.29)) = 12.9Y$

<sup>7</sup> Very rarely projects do get some unexpected bonuses through newly discovered reusable assets, or through newer, quicker, or more reliable asset production techniques invented between the time of estimation and project deadline. Rest assured, however, that clients will welcome any such productivity boosts in the same way they would welcome winning the lottery.

<sup>8</sup> Here we are assuming the simplest linear relation; we can put in a weight factor to account for a slower pace of execution towards the beginning of project, which may be due to a team's unfamiliarity with the process and / or the team's deliberately addressing the difficult, architecturally significant features early in the project.

<sup>9</sup> You can refer to the model manual at [ftp://ftp.usc.edu/pub/soft\\_engineering/COCOMOII/cocomo99.0/modelman.pdf](ftp://ftp.usc.edu/pub/soft_engineering/COCOMOII/cocomo99.0/modelman.pdf). For those who want to go easy, the book *Software cost Estimation with COCOMO II* by Boehm et. al. is a useful source. For those who particularly love formulae and would like to recap other celebrated works related to software estimation, I can suggest the PhD work of some of Boehm's team members, in particular Chapter 2 of

<http://sunset.usc.edu/publications/dissertations/SChulani.pdf>



---

***For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided.  
Thank you!***