Research paper accepted for EuroSPI 2004 http://2004.eurospi.net/

Project Effort Estimation: Or, when Size Makes a Difference

Oddur Benediktsson[1] and Darren Dalcher[2]

[1] University of Iceland, Hjarðarhaga 2-4,
107 Reykjavik, Iceland
oddur@hi.is
[2] Middlesex University, Trent Park
Bramley Road, London N14 4YZ, UK
d.dalcher@mdx.ac.uk

Abstract. The motivation for this work is derived from the current interest in speeding up development schedules. A key implication of the shift to more rapid development methods is the growing emphasis on fixed time and fixed effort delivered during such projects. However there appears to be little work that addresses the impacts of dealing with bound effort levels. The result of binding time and effort is to deprive project managers of the normal parameters that are used in tradeoffs. The paper attempts to introduce a quantitative analytical framework for modeling effort-boxed development in order to uncover the effects on the overall development effort and the potential leverage that can be derived from incremental delivery in such projects. Models that predict product size as an exponential function of the development effort are used in the paper to explore the relationships between effort and the number of increments, thereby providing new insights into the economic impact of incremental approaches to effort-boxed software projects.

1 Introduction

The popular computing literature is awash with stories of IS development failures and their adverse impacts on individuals, organisations and societal infrastructure. Indeed, contemporary software development practice is regularly characterised by runaway projects, late delivery, exceeded budgets, reduced functionality and questionable quality that often translate into cancellations, reduced scope and significant re-work cycles [8]. Failures in particular, tell a potentially grim tale. In 1995, 31.1% of US software projects were cancelled, while 52.7% were completed late, over budget (cost 189% of their original budget), and lacked essential functionality. Only 16.2% of projects were completed on time and within budget; only 9% in larger companies, where completed projects had an average of 42% of desired functionality [16]. The 1996 cancellation figure rose to 40% [17]

The cost of failed US projects in 1995 was $81 billion, in addition cost overruns contributed an additional $59 billion (so that out of the $250 billion spent on 175,000 US software projects, $140 billion was spent on cancelled or over budget activities)

[16]. In fact, Jones contended that the average US cancelled project was a year late having consumed 200 percent of its expected budget at the point of cancellation [10]. In 1996, failed projects alone totalled an estimated $100 billion [12]. In 1998, 28% of projects were still failing at a cost of $75 billion, while in 2000, 65,000 of US projects were reported to be failing [18].

The Standish Group makes a distinction between failed projects and challenged projects. Failed projects are cancelled before completion, never implemented or scrapped following installation. Challenged projects are completed and approved projects which are over-budget, late and fewer features and functions than initially specified. Most organisations are constantly searching for ways to improve their development practice and reduce the likelihood of failures and the degree to which their projects are challenged.

Typical projects entail a balancing act between the triple constraints of budget, schedule and scope. Trade-offs and adjustments are therefore made by restricting, adding to or adjusting the cost, time and scope associated with a project. Budding IT project managers are introduced to the holy trinity of project constraints at an early stage and are advised to balance the factors in order to define an acceptable compromise.

Indeed the traditional triangle in project management is said to be concerned with finding the balance between cost, time and scope (see Figure 1). For example, the more that is requested in terms of scope (or arguably even the performance or the quality) the more it is likely to cost and the longer the expected duration. If the client needs to have a certain performance delivered very rapidly, this will increase the cost due to the need to work faster and have more people involved in the development. The more features expected from a system, the higher the cost and the longer the expected duration. If the costs need to kept to a minimum, one may need to consider the essential performance, or the overall scope, and compromise there.

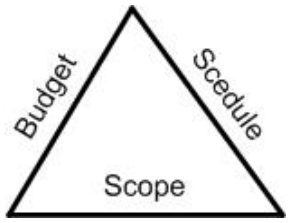

Fig. 1. Budget, Schedule and Scope trade off

The three factors are obviously closely entwined and project managers are expected to balance the **what** (scope) with the **when** (schedule) and the **how much** (budget). The trade-off between time, scope/performance and cost is essential in understanding the tension in a typical project management environment.

Many managers quickly discover that the triangle is not flexible. Performance and scope are often pre-determined prior to the project. Moreover, project managers often inherit the overall budget from the contracting activities that may even have imposed a fixed-price contract structure. A fixed overall budget may also exclude the hiring of

specialists and the addition of human resources. The only remaining scope for leverage is in the schedule. However, this may also be imposed on a project through a fixed date for delivery with little regard to the complexity of the intended system or the risks it embodies. Once both budget and schedule are fixed, there appears to be little scope for compromises and tradeoffs.

The three factors play a key part in determining the degree to which a project is challenged (or even deemed a failure), yet they may be uncontrollable by the project manager. Indeed, Capers Jones observed that the most common constraints encountered are: fixed delivery dates; fixed-price contracts; staffing or team size limitations; and performance or throughput constraints [11] i.e. fixed time, price, staffing level (see Section 2), performance and scope. Many managers are thus looking to control other factors that may alter the outcome of the project, especially as the constraints often occur in concert. An alternative solution is to opt for iterative or incremental delivery of partial, yet useful versions of the system. Such an approach can rely on the success of earlier deliverables and build on them to upgrade the system. This can be achieved within the overall timescale whilst adhering to the intended scope of the system. The approach is useful in trying to mitigate the effect of failures by reassessing the outcome after achieving each increment or deliverable milestone. It also empowers project managers by enabling them to operate and make decisions within the traditional triangle of project management.

2 Working Incrementally

In many modern IT projects, the desire to avoid failures is further exacerbated by the need to deliver almost instantaneous functionality in response to changes. The pressure to collapse the schedule is typically due to the need to respond to opportunities, time-to-market demands, the need to get there before the competition or the implications of technical obsolescence. As organisations race to ensure a functioning presence on the web, the new mode of operation is introducing additional constraints forcing new working patterns. The market is fast-moving characterised by a dynamic environment with high levels of uncertainty and risks. Customers appear more demanding and discerning expecting non-stop service around the clock. Service is being judged according to overall traffic, frequency and duration of visits and loyalty.

The rate of change endemic to this environment leads to small and quick projects with high frequency of release and smaller scope explored in each release (akin to timeboxing). The scope is highly changeable over time to respond to competition and opportunities. Time therefore becomes the critical factor: As the speed of release is measured by getting there before the competition, project execution is measured according to the shortest time to register a presence (often regardless of the quality). Trade-offs can often sacrifice scope, cost, expectations or quality to accelerate the speed of completion.

Most projects in this environment are new, innovative and difficult to estimate and cost. The planning approach thus needs to focus on key milestones and targets, yet remain flexible enough and responsive enough to cope with changing requirements, delivery dates, release deadlines and new opportunities. Projects tend to be evolution-

ary and have short release cycles as they build on what already exists and are often prone to scope re-adjustments in line with shifting knowledge and expectations.

Incremental and iterative development approaches have long been recognised as effective in reducing the risk of failure in such situations as they entail a more controlled approach to development [2]. Incremental approaches emphasise phased development by offering a series of linked mini-projects (referred to as increments, releases or versions) [9]. Work on different parts and phases, is allowed to overlap throughout the use of multiple mini-cycles running in parallel. Each mini-cycle adds additional functionality and capability. The approach is underpinned by the assumption that it is possible to isolate meaningful subsets that can be developed, tested and implemented independently. Delivery of increments is staggered as calendar time progresses. The staggered release philosophy allows for learning and feedback to alter some of the customer requirements in subsequent versions. Incremental approaches are particularly useful when there is an inability to fully specify the required product or to fully formulate the set of expectations under some budgetary control.

Note that the term 'Incremental development' as used here embraces the spectrum of iterative development and delivery (ranging from the full product architecting and design being performed upfront, to evolutionary methods and may also be extended to refer to delivery modes in agile methods with little or no a priori product design).

Indeed in the context of managing projects within the triangle of constraints, the adoption of **incremental development**, **Rapid Application Development or** the use of **agile methods** enables project managers to gain flexibility within the pre-defined boundaries of overall time and cost. These approaches are focused on the need to deliver relevant working business applications quicker and more cheaply. The application is likely to entail delivery in incremental fashion while maintaining user involvement through the application of design teams and special workshops. Projects tend to be small and limited to short delivery periods to ensure rapid completion and are thus particularly suitable as controlled cycles of small implementation released as part of a larger project. The management strategy utilised in incremental development relies on the imposition of *timeboxing*, the strict delivery to target which dictates the scoping, the selection of functionality to be delivered and the adjustments required to meet the deadlines. Rapid Application Development and agile methods are particularly useful in environments that change regularly and impose demands of early solutions.

The motivation for this work is thus derived from the current interest in speeding up development schedules. A key implication of the shift to more rapid development methods is the growing emphasis on fixed time and fixed effort in projects. Going back to the triple constraint, if time schedules are fixed by timeboxes and costs are largely dictated by the availability of personnel to work within these imposed time frames, the main variable is the scope that can be delivered. However, the scope (including the actual delivered and useful functionality and the quality or usability of that functionality) is the driving concern in many projects. Indeed, scope needs to be handled with care as delivering partial functionality (e.g. the release of a sophisticated order recording systems on the web a few weeks prior to Christmas without a corresponding order fulfilment system capable of despatching the ordered goods).

Project managers respond to such constraints by trying to adjust the other parameters of the project. However, assuming that overall cost, time and scope are fixed, which variable remains to be manipulated? Fixed time boxes in fixed time frames, result in *chunks of fixed person-months* required to complete the task. The key question is, given the reality of fixed chunks of person-months, are there any methods for manipulating person-months in an attempt to address project objectives and leverage off different arrangements? More crucially perhaps, we are interested in finding out how much can be accomplished with such a given effort.

The current literature offers very little clues about what can be accomplished. The Standish Group recommended that an ideal (and achievable) project would entail six developers working for six months [16]. This confirms the notion of fixed developer resources imposed on the project. These results are corroborated by a large study of success and failure in over 1,000 UK IT projects which found that 90% of the successful projects were less than 12 months long and 47% had a duration of less than 6 months [20]. Note however, that deploying a fixed team of six developers for six months ultimately gives 36 *PM* as a fixed chunk of key project resources that can be manipulated to ensure that the project does not get out of hand. Recently, the Standish Group revised the figures to suggest an ideal team of four developers working for four months, giving a much reduced total of 16 *PM*. (Note that a person-month is defined in the computing literature as the amount of time one person spends working on a software development project for one month.) Indeed, the Standish conclusions assert that 'shorter timeframes with delivery of software components early and often, increase the success rate' [18]. Moreover, they implicitly confirm that managers now have a new type of resource to manipulate.

The interest in agile methods and smaller and rapid projects is driving down the average duration and fixed level of resources and hence the *project resource envelope* that can be utilised by project managers. Indeed, in Extreme Programming developers often talk about even smaller chunks of development (measured in weeks and therefore totalling 1-4 months) while DSDM recommends timeboxes of three month duration [19]. The figures take us back to the all important question: 'Does size matter?'

Earlier work attempted to explore the relationship between effort and the number of increments [2]. A key conclusion of that work was the realisation that the ratio of work (and hence the overall work) decreases as the number of increments increases. Indeed, using multiple increments appears to offer great benefit compared to monolithic development approaches. This work can now be extended to explore the impact of dealing with fixed time, scope and effort parameters by dividing the fixed *project resource envelope* authorised for a given project into a set of increments.

The rest of the paper will focus on providing a framework for evaluating the relationship between project size and the effort required to deliver it in the context of a fixed effort level. The framework will then be populated with a range of values to provide a sampling of scenarios for different project environments. These scenarios offer an insight into the behaviour patterns expected under different conditions. The paper concludes by drawing a set of observations from the results and commenting about the implications of the results and about the ideal size and complexity relationship, before finally re-assessing the issue of ideal size in order to provide practical answers.

3 Size of the Development Effort

Estimating the development effort for a software system is a long standing problem in software project management. It has generally been noted that the effort (*E*) is strongly correlated to the program size. A great deal of research has been carried out to relate the effort empirically to the product size, with the effort being commonly expressed in Project Months (PM) and the size in Kilo Lines of Code (KLOC) or in Functions Points (FP). Typical models used to express this relationship are of the form:

$$PM = c + a\ Size^{\ b} \quad (1)$$

where a, b, and c in equation (1) are determined by regression analysis using a collection of project outcomes. Some of the better known empirical models are highlighted in Table 1 [3, 4, 14, 21].

Table 1. Effort estimation models

Author	Original formula	Inverted form
Halstead	$PM = 0.7\ KLOC^{1.50}$	$KLOC = 1.27\ PM^{0.667}$
Basic COCOMO organic	$PM = 2.4\ KLOC^{1.05}$	$KLOC = 0.43\ PM^{0.952}$
Basic COCOMO semidet.	$PM = 3.0\ KLOC^{1.12}$	$KLOC = 0.37\ PM^{0.893}$
Basic COCOMO emb.	$PM = 3.6\ KLOC^{1.20}$	$KLOC = 0.34\ PM^{0.833}$
COCOMO II.2000	$PM = 2.9\ KLOC^{1.10}$	$KLOC = 0.38\ PM^{0.909}$
Walston-Felix	$PM = 5.2\ KLOC^{0.91}$	$KLOC = 0.16\ PM^{1.10}$
Bailey-Basil	$PM = 5.5\ KLOC^{1.16}$	$KLOC = 0.23\ PM^{0.862}$
Doty (for KLOC > 9)	$PM = 5.288\ KLOC^{1.047}$	$KLOC = 1.27\ PM^{0.674}$
Albrecht and Gaffney	$PM = -13.39 + 0.0545\ FP$	$FP = 245.7 + 18.35\ PM$
Kemerer	$PM = 60.62 \times 7.728 \times 10^{-8}\ FP^3$	$FP = 59.76\ PM^{0.33}$

Column three of Table 1 is derived from column two by solving for *Size* in terms of *PM*. The models are seen to vary considerably and can be expected to give different results in individual estimation cases. The estimation accuracy is improved by setting a number of parameters to reflect the situation at hand regarding the product, process, platform, and people involved as is done for example in COCOMO II [4].

The terminology used in this paper is that of the COCOMO II model [4]. COCOMO II represents an update to the original COCOMO 81 work [3] to account for contemporary development practices. The paper builds on earlier work [2, 6] that utilises the notation as a quantitative analytic framework for evaluating software technologies, project decisions and the economic impacts in terms of the resulting projects. The focus in this work is on relating the *Size* of the outcome of a development project to the input effort. Column three of Table 1 gives expressions for this relationship. Note that the *Size* (KLOC or FP) is an exponential function of *PM* with the exponent being less than or equal to one in all but the Walston-Felix model. The relation that will be exploited here is of the form

$$Size = c\ PM^{\ d} \quad (2)$$

which is in line with all of the above mentioned models except the Albrecht and Gaffney model which has a constant term.

4 Size in Incremental Development

Suppose that *PM*, the total allocated project effort, is fixed for a given development project (e.g. following the recommendation of the Standish Group or as proposed by the proponents of agile methods and DSDM) and that the nature of the project is such that it can be broken into a sequence of *n* increments. The effort exerted in the individual increments is denoted by PM_1, $PM_{2, \ldots}$,PM_n where

$$PM = \sum_{i=1}^{n} PM_i \tag{3}$$

While incremental development aims towards a target functionality, there has to be some additional effort used for the integration of the different increments. The term used for this in the COCOMO models is *breakage* as some of the existing code and design has to be mended to fit a new increment in. The breakage effect will be accounted for by reckoning that some fraction of PM_i, the incremental effort exerted in increment *i*, is gained as system enhancement. This fraction is termed *Gain Multiplier (GM)*. The net effort is expressed as $GM_i \times PM_i$.

Kan asserted that 20% of the added code in staged and incremental releases of a product goes into changing the previous code [13]. Cusumano and Selby reported that features may change by over 30% as a direct result of learning during a single iteration [7]. In a recent paper the authors argued that the incremental integration *breakage* can be expected to lie in a range from 5% to 30%. This relates to the range in Gain Multipliers of 95% down to 70% in approximate terms [2]. Thus a Gain Multiplier can assume a range of values. The GM will equal *1* when all of the gross work delivers new functionality. Where some gluing, integration or refactoring effort is needed the efficiency will be reduced below *1*. The gain factor can exceed *1* in the case of marked code reuse (thus suggesting negative entropy and a major benefit from the application of reuse). One might ask if the Gain Multiplier can be negative in which case adding an increment is counterproductive – i.e. more code needs to be mended than has been written. This interesting pathological case is outside the scope of this work. It is therefore assumed here that $GM>0$.

The net incremental effort exerted for increment *i* is $GM_i \times PM_i$. The $Size_i$ of a developed increment is then derived from expression (2)

$$Size_i = c_i (GM_i \, PM_i)^{d_i} \tag{4}$$

This relation does not hold if the increments are "too small". For example, the COCOMO formulation stipulates that the size has to be larger than 2 KLOC (which roughly equates with 6 PM). Note also that for one-off development $n=1$ and $GM=1$ which is consistent with expression (2).

When the outcome of the individual increments is accumulated the total developed size $Size_I$ is

$$Size_I = \sum_i Size_i \tag{5}$$

and by combining (4) and (5)

$$Size_I = \sum_i c_i (GM_i \, PM_i)^{d_i} \tag{6}$$

When the development context is uniform (i.e. one and the same team throughout as well as homogeneous project and process properties) then the *c* and *d* parameters in (6) are taken to have constant values across the increments i.e.

$$c_i = c \text{ and } d_i = d \text{ for } i = 1,\ldots n$$

and then expression (6) simplifies to

$$Size_I = c\sum_i (GM_i\, PM_i)^d \tag{7}$$

Expressions (6) and (7) are seen to represent **a general way of estimating the outcome of incremental development projects.**

The following question arises: Does the number of increments matter for the resulting product size? Expression (7) will be explored below in a search for an answer to this question.

First, the one-off case (expression (2)) is employed to normalize expression (7) resulting in what is termed here the *Relative Productivity (RLPR)* that is $RLPR = Size_I / (c\, PM^d)$. By combining (2) and (7)

$$RLPR = \sum_i (GM_i\, PM_i / PM)^d \tag{8}$$

The multiplier *c* in (2) and (7) has been factored out in the normalization. *RLPR* is seen to give clear indications of productivity in incremental development - for $RLPR > 1$ the productivity has increased while for $RLPR < 1$ the productivity has decreased as compared to the one-off case.

Second, suppose that the total effort exerted in the n increments is equally divided. In other words, the work is effort-boxed. In this case $PM_i = PM / n$ for all *i*. Then expression (8) simplifies to

$$RLPR = n^{-d}\sum_i (GM_i)^d \tag{9}$$

As mentioned earlier effort-boxed development is gaining in popularity. Evidence of this trend can be seen as a result of wide adoption and interest in agile methods and extreme programming as well as the use of the effort-box notion in IT projects. Moreover, improved failure statistics in failure surveys is directly attributable to the increasing proliferation of mini projects.

Third, a parameter termed the *Cumulative Gain (CG)* is now defined as

$$CG = 1/n\sum_i (GM_i)^d \tag{10}$$

CG is a function of *n*. For $d=1$ (the linear case) *CG* gives the arithmetic average of the GM_i values. For a constant *GM* across the increments denoted as $\underline{GM}$ then $CG=\underline{GM}^d$. In general CG can be expected to be a complex function of a) the development context of a project, b) the number of increments, and c) the extent of reuse. Expressions (9) and (10) combined yield the following compact expression for the Relative Productivity:

$$RLPR = n^{1-d} CG \tag{11}$$

The parameter *d* is taken to be restricted to $0<d<1$ (see Table 1 Column 3). In the case where *CG* is constant, then expression (11) is seen to be a monotonic increasing function of *n* i.e. the productivity increases with increasing number of increments.

Note that the underlying assumptions in this expression (11) are a) effort-boxed increments and b) uniform development context. This expression will be exploited in

the next section in the quest for gaining additional insight into the behaviour of the incremental development model.

5 Discussion

Table 2 shows sample computations of *RLPR* using expression (11). Relative Productivity values that are less than 1 are italicised. For these values there is no gain in using incremental development in terms of reduced development effort. Below the italicised region, *RLPR > 1* and the productivity increases as compared to the one-off development.

Table 2. Relative Productivity ***RLPR*** for the exponent fixed at ***d = 0.833***

	CG Cumulative Gain			
n	**1**	**0.9**	**0.8**	**0.7**
1	1.00	*0.90*	*0.80*	*0.70*
2	1.12	1.01	*0.90*	*0.79*
4	1.26	1.13	1.01	*0.88*
6	1.35	1.21	1.08	*0.94*
8	1.42	1.27	1.13	*0.99*
10	1.47	1.32	1.18	1.03
12	1.51	1.36	1.21	1.06

The productivity is seen to increase down a column for constant Cumulative Gain as has been observed. However, for a particular project, a planning scenario could possibly yield the incremental alternatives presented in Table 3. The planning alternatives that could be utilised by the project manager are the *(n, CG)* pairs *(4, 0.9)*, *(6, 0.8)*, and *(8, 0.7)* and expression (11) (or indeed Table 2) has been used to find the corresponding Relative Productivity. Here the *n=4* alternative is the most productive case – the one with the fewest elements.

Table 3. A planning scenario with ***d = 0.833***

n	**CG**	**RLPR**
4	**0.9**	1.13
6	**0.8**	1.08
8	**0.7**	0.99

The above discussion sheds some light on one of the highly debated issues of today: "Is XP a better development method than the traditional approaches?" One of the cornerstones of XP is incremental delivery with a large number of increments. Kent Beck [1] states that the technical premise for this is that the cost of fixing defects rises slowly in time. This is reflected in a slowly increasing Cumulative Gain, in which case productivity is expected to rise significantly with an increasing number of in-

crements as deduced from Table 2. However, the traditional view has been that the cost of fixing defects rises sharply with time [3, 5]. If this is the case then increasing the number of increments could possibly reduce productivity as demonstrated with the planning scenario of Table 3.

It is also interesting to investigate the effects that the exponent *d* exerts on the Relative Productivity. Table 4 exemplifies the effect of varying *d* in a case where the number of increments is fixed at 24. (The values of *d* chosen here correspond to the scale factors in the Basic COCOMO model as seen in Table 1 where the *0.833* value corresponds to the embedded model, *0.893* to the semidetached model, and *0.952* to the organic model which can be viewed as a set of representative values for projects).

Table 4. Relative Productivity ***RLPR*** for *n* fixed at ***n = 24***

	CG Cumulative Gain			
d	**1.0**	**0.9**	**0.8**	**0.7**
0.833	1.70	1.53	1.36	1.19
0.893	1.41	1.26	1.12	*0.98*
0.952	1.16	1.05	*0.93*	*0.82*

An interesting pattern emerges here: The lowest values of *d* yield the highest gain in productivity *(RLPR)*. It is to be noted that a low value of *d* signifies a difficult development context with low overall productivity.

The line of Table 4 for *d = 0.952* shows that *RLPR < 1* for the *0.8* and *0.7* values of *CG* indicating that there is a loss in productivity in this case as compared to the one-off development. For the higher values of *CG* there is a productivity gain. High values of *d* (e.g. 0.952) in the COCOMO II model is the result of project parameters of high precedence in project type, flexible documentation, well conducted risk resolution, high team cohesion, and high process maturity [5]. In this type of development instance the Cumulative Gain can be expected to be high as well.

6 Conclusion

IT projects continue to provide challenging tradeoffs for project managers. The tendency to pre-determine many of the key variables (including budget, schedule and scope) prior to project launch leaves managers with few tradeoff options. As a partial response to project failures, many organisations have adopted timeboxing notions to ensure successful completion and delivery. The resulting impact is in creating effort-boxes which project managers are expected to balance and control. These can be likened to a *project resource envelope* which determines the key parameters of a project. To date there has been little effort to understand this phenomenon and to reason about the methods required to control it.

Advice regarding projects is sometimes delivered in the form of the overall effort-box required to increase the likelihood of successful completion (i.e. six people for six months or four people for four months). Table 1 reworks typical estimation mod-

els to reflect size in terms of the required effort so that $Size = c\ PM^{d}$. The required effort can equate to the *project resource envelope*, or the effort-box.

The key implication is that the overall project effort is fixed. In other words, resources are determined for the duration of the project. This represents a shift from organisational and project structures that allow the project managers to call on a variety of skills as and when needed thus implying that whilst specialist resources are perhaps no longer obtainable, the project manager is at least dealing with unchanging resource levels. Fixed resource levels eliminate many tasks associated with resource management. Besides reducing the need for comprehensive resource estimation, calculation, levelling, optimisation and smoothing, which often prove to be time-consuming activities, the new arrangement removes the reliance on external availability of resources suggesting that the focus is on the internal project team. The remaining flexibility can be derived from dividing the given total fixed effort into segments for incremental delivery as described in Expression (3) and in trying to maximise the output values made possible through this division.

The paper investigates the implications of working on projects with a fixed-effort level where increments may be utilised to obtain additional, or perhaps the only leverage, for project managers. The productivity gain from incremental development in fixed effort situations comes when *relative productivity RLPR>1* as this is where the productivity is increased compared with a single delivery project.

The most difficult and complex environments with high rates of change and associated low productivity rates provide the best candidates for incremental development. The lowest values of *d* yield the highest gain in productivity *(RLPR)*. High level environments are less likely to show a direct benefit in terms of productivity from adopting an incremental approach (possibly even indicating a loss of productivity compared to the one-off model).

The analytic framework explored in this work provides a new method of reasoning in the increasingly common situations where effort is fixed (along with time, cost and expected scope). The framework enables project managers to explore the impact of adopting incremental delivery practice thereby enabling them to obtain insights and as a result derive leverage from using increments as part of the limited tradeoffs possible under such fixed constraints.

Most of this work makes the simplifying assumption that a uniform development context is maintained throughout the duration of the project. Indeed, significant changes to the context or to the development environment will invalidate most estimation and planning methodologies.

The work reported in this paper can be extended in a number of directions to explore a range of potential avenues. In particular, the authors are interested in addressing a number of emerging challenges that include:

- Collecting empirical data on incremental development to validate the postulated formulation
- Analysing the relationship between incremental delivery and team size
- Relating the findings to factors expressed in alternative estimation models
- Utilising a richer estimation model to conduct deeper analysis of the cost/duration implications of incremental development

- Investigating the relationship between incremental development and the 'evolvability' of systems
- Providing a basis for reasoning about continuation and cancellation of projects (or deliverables) and conducting trade-offs between time, effort and increments in dynamic projects

It is therefore hoped that the work put forward in this paper will serve as an opening for further discussion and investigation. The opening of a new perspective can also stimulate additional insights needed to bridge the gap in the analysis of the economic impact of non-conventional development approaches on the product, process and project.

References

1. Beck, K., 1999. *Extreme Programming Explained: Embrace Change.* Addison-Wesley.
2. Benediktsson, O. and Dalcher, D. 2003 Effort Estimation in Incremental Software Development. IEE Proc. Softw., Vol. 150, no. 6, December 2003, pp. 351-357.
3. Boehm, B.W.1981. *Software Engineering Economics*. Englewood Cliffs: Prentice Hall.
4. Boehm, B. W., Abts, C., Brown, A. W., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D., and Steece, B. 2000. Software Cost Estimation with COCOMO II. Prentice-Hall.
5. Boehm, B. W. and Turner, R. 2003, Balancing Agility and Discipline. Addison Wesley.
6. Clark, B., S. Devnani-Chulani, and B.W. Boehm. 1998. *Calibrating the COCOMO II Post-Architecture Model*. in *Proceedings of ICSE 1998*. IEEE Press.
7. Cusumano, M.A. and Selby, R.W. 1995. Microsoft Secrets: how the world's most powerful company creates technology, shapes markets, and manages people, Free Press, New York.
8. Dalcher D. 1994. Falling down is part of Growing up; the Study of Failure and the Software Engineering Community, *Proceedings of 7th SEI Education in Software Engineering Conference*, New York: Springer-verlag, pp. 489-496.
9. Dalcher, D. 2002. Life Cycle design and Management in Stevens M. (Ed.): Project Management Pathways: A practitioner's Guide, High Wycombe: APM Press.
10. Jones, C. 1994. *Assessment and Control of Software Risks*. Englewood Cliffs, New Jersey: Prentice-Hall.
11. Jones, C. 2000. *Software assessments, Benchmarks and Best Practices*. Upper Saddle River, New Jersey: Addison-Wesley.
12. Luqi and. Goguen, J.A. 1997. Formal Methods: Promises and Problems. *IEEE Software*. Vol 14(1), pp. 73-85.
13. Kan, S.H. 2003. Metrics and Models in Software Quality Engineering, 2nd ed, Pearson Education.
14. Pressman, R.S. and D. Ince, Software Engineering: A Practitioner's Approach. 5 ed. 2000, Maidenhead: McGraw-Hill
15. Royce, W. 1998. *Software Project Management: A Unified Framework*. Addison Wesley.
16. Standish_Group. 1995. *Chaos 1995*. Standish: Dennis, Mass.
17. Standish_Group. 1997. *Chaos 1997*. Standish: Dennis, Mass.
18. Standish_Group. 2000. *Chaos 2000*. Standish: Dennis, Mass
19. Stapleton, J. 1997. *DSDM Dynamic Systems Development Method*. Addison-Wesley.
20. Thomas, M. 2001. IT projects Sink or Swim, British Computer Society Review.
21. Vliet, H.V. 2000. Software Engineering: Principles and Practice. 2 ed., Chichester: Wiley.

