

1.4 Software Metrics and Effort Estimation

In section two of the literature survey, an overview and analysis of the procurement process was described and its weaknesses were highlighted. By one definition, all project failures are failures of estimation. There is only a complaint when a project is delivered late; it is only late because it has been predicted to be finished inaccurately. This section investigates the methods that are in use for the calculation of effort estimates, and highlights the difficulties in producing them, and the problem with verifying they are *reasonable*. When a tender response is received with a quoted price, it is seldom clear how the price has been calculated, therefore it is unclear how much confidence the customer should have that this is the price they will actually pay.

Effort estimation is the weak link in software project management. Often a task delegated to a project manager, the resulting estimates, made very early in the development lifecycle, are notoriously inaccurate. The time it takes to build a piece of software is an intractable question. Typically, not enough time is devoted to the calculation of effort, as the time spent is typically not funded. The individuals doing the job may not have the necessary experience to perform it well. Estimates based on inadequately described requirements are bound to be incorrect, and it has been shown that the expression of requirements is, itself, notoriously poor [1]. Requirements statements too often consist of the 'when' and not the 'what' [2] of project definition. Understanding the relationship between time, size and cost in software development is fundamentally complicated and insufficiently understood. For instance, limiting the time available to a project may be seen as a strategy to reduce risk, when in fact it has the effect of reducing efficiency and hence driving costs up [2]. It is reasonable to consider cost in terms of software development to refer to the cost of labour, hence project duration and human resources allocated are the primary cause of rising costs when a project is delivered late.

Estimates are needed throughout a project's lifecycle; from preliminary estimates when bidding for a contract, to multiple interim estimates required to measure progress against targets. Yet, inevitably, preliminary estimates are the most difficult to obtain, and are often the least accurate because too little is known about the project in the early stages. Once begun, more detailed estimates help in project planning, but can, embarrassingly, show the initial estimate to be far off the mark [3].

The history of unrealistic bids has created distrust between suppliers and customers [2]. In a study by Mohanty the cost of one software development project varied by almost 800% when estimated using 12 different models [4]. There is clearly a lot of work needed in the field of software cost estimation, however, it is not clear that results will ever match expectations.

Commercial pressure

Where projects are subject to competitive bidding, a bid that is too high compared with competitors will invariably result in the loss of the contract, whereas a bid that is too low will normally result in a financial loss; sometimes to the supplier, sometimes to the customer, and sometimes to both.

Bidding to win, and yet still to make a profit, is a difficult balancing act. It may be further exacerbated by the commercial need of an organisation to win a large contract because they are short of work. This may be a foolhardy basis upon which to choose a winner, if the lowest bid corresponds to the most desperate supplier. Sales people are rewarded based on what they sell, not on the profitability of work [2] which can act as an internal pressure for suppliers to bid low prices that are plainly inadequate to the task at hand. This approach sets the conditions for later disputes. As power shifts from the customer to the supplier over the lifetime of a project, a situation may develop whereby the supplier demands more funds to continue work using the threat that unless more money is forthcoming, work will cease [5, 6]. Suppliers may try to exploit ill-defined requirements to support their case for an increase in budget. This may render the original 'lowest' price, more expensive than other of the more realistic original bids, thus making a mockery of the competition [2]. Although strictly outside a discussion of effort estimation methods, it is important to recognise the reality of commercial pressure that results in an approach to predicting costs that is driven by a philosophy of 'price to win' regardless of the feasibility of completing the project for the price quoted [3]. The prevalence of this practice is hard to quantify, although empirical evidence [7-9] suggests it may be common. Unfortunately it is the customer who often pays the price of this strategy; a strategy that could be described as immoral, were morals to have any place in the discussion. The problem is made worse when the customer is the government and the funds being spent are raised from taxation, as increasing costs may not be sufficiently resisted by a customer who feels under political pressure to deliver what has been announced and where the impression may prevail that they have access to an inexhaustible source of funds from the public purse. This may be a consequence of a more general tendency of Government

projects where there is no discernible project owner [10] and no board of directors or stock holders whom can be held to account. What is required is an evaluation on the part of customers as to whether the price they are being quoted is, in fact, realistic [11].

The Nature of Estimation

Effort estimation is a prediction of the cost (predominantly labour) involved in delivering software to a certain specification. There is an assumption that the specification is *sufficient* to allow an estimate to be made, however, this is not always the case, nor is it necessarily the norm. An estimate is theoretically defined as a target that falls within a range of values predicted to occur with a high degree of probability. It is a mistake for a supplier to quote a value that is in fact at the very lowest end of a range of possibilities [12]. Consider a situation in which an estimate is required from a set of requirements. Were it possible to record the actual completion times of a large number of projects, all trying to implement the same set of requirements, a graph of the probability density function to time (t) could be constructed. In reality this distribution is not known and cannot be known, unless an experiment was held that resulted in many suppliers building versions of the same software; a situation that is unlikely, in the extreme, to occur. Leaving this practicality aside, an estimate can be defined as the median of a distribution. Software engineers often misconceive an 'estimate' as the smallest defensible amount of time in which a project can be completed. This interpretation points not to the median, but rather the minimum value, of (t) that has a non-zero probability. The minimum value is the shortest possible time in which the project can be finished, the probability of such, by definition, being low [3]. Therefore, an estimate is a triple, featuring the most likely value (median) plus the upper and lower bounds. Probability theory terms these bounds 'confidence intervals'.

At the time of competitive tendering for a project, very little is known about the details of what is actually required. For this reason, ideally, early lifecycle estimation should be avoided because it is notoriously inaccurate [13], however, in a world of competition, it continues to be a fact of life. Suppliers have developed their own *ad hoc* practices to deal with this conundrum, and those with a more academic bent, have worked to develop models that may be applied with different levels of utility. All methods have suffered from lesser accuracy the earlier in the lifecycle they are applied. An algorithmic method for calculating effort estimates during the early lifecycle stage would be extremely useful, even if it were only capable of producing gross estimates [13]. Considering these serious

impediments to accurate estimation, a sensible improvement would be for customers to modify their procedure where instead of expecting a fixed price estimate they demanded to know the upper and lower bounds of the given estimate and the confidence level attached.

Heuristic methods

Heuristic methods of estimation are essentially based on the experience that exists within a particular supplier organisation where past projects are used to estimate the required resources necessary to deliver future projects. A convenient sub-classification is to divide heuristic methods into ‘top-down’ and ‘bottom-up’ approaches. These approaches are the *de facto* methods by which estimates are produced and as such they are implicated in being poor reflections of the actual resources that are required as evidenced by the failure of projects to be estimated accurately. There is little in the literature on these *ad hoc* approaches, whose chief advantage is that they are easy and quick to apply [14].

Top-down approaches to effort estimation may rely on the opinion of an expert within an organisation who seeks to draw upon analogy between new requirements and historic projects that have been delivered that are considered to be similar [1, 14, 15]. A variation of this approach, sometimes referred to as a Delphic approach, is to seek the opinions of a group of experts and to work to seek consensus between them. By definition this approach is subjective and unstructured even though it is the most commonplace method in use by practitioners [16, 17].

Bottom-up estimation is the process by which time needed to code each identified module is estimated based on the discrete tasks that must be performed, such as analysis, design and project management. Historical records, if these are available, may be used to perform checks to ensure estimates do not vary wildly from standard percentages to ensure the time allocated appears to be reasonable. This approach, if not conducted systematically, runs the risk of common tasks being overlooked such as integration, quality and configuration [1]. Task summation is however, reportedly a common method of estimation employed by practitioners [18].

On one hand it has been suggested that organisations do not have the resources to develop and analyse a large enough historical database of their own projects [17] to ensure checks can be undertaken. Other authors suggest the task is not necessarily so onerous recommending that, in the beginning, a database of historical projects need only contain the details of 10 projects before being of utility [12, 15]. One

priority is to identify which productivity factors are the most important in determining the differences between projects. Additionally, the complication exists that these factors may vary over time within a particular company. In one study, the main factors found to affect productivity were size of the project, application category, programming language, reliability, storage constraint, programming practices and the use of tools [17] however, the details of exactly what metrics should be collected and at what frequency are not reported. It is unlikely that a supplier organisation can muster much enthusiasm for collecting metric data unless they are convinced that the effort expended in doing so will be worthwhile. This tendency hinders improvement, because experts, who presumably base their estimates on past experience, do not have access to any meaningful historical data. Therefore the organisation cannot compare estimates with known 'actuals'.

Effort estimation is predicated on the assumption that the target organisation has a repeatable process in place, and that efficiency does not vary wildly between one project and another. This may be an erroneous assumption according to a survey undertaken by the Software Engineering Institute at Carnegie Mellon University which shows that over 75% are stuck at level 1 of their Capability Maturity Model [19-21]. Level 1 is the first of five possible levels that may characterise an organisation as having *ad hoc*, or possibly chaotic processes [22]. In this environment, managers may simply guess to arrive at an estimate, relying on their intuition. This practice is, unsurprisingly, highly positively correlated with large projects that overrun their estimates [23]. There is no evidence to suggest that heuristic methods are any worse than other more formal methods [14] at producing accurate estimates; faint praise given the woeful level of accuracy for both.

Algorithmic methods

Two prominent algorithmic methods appeared in the late 1970's. The motivation for new methods came from the military's need to understand the very large sums of money they were being asked to spend, and the equally perplexing problem of where that money was going. Boehm worked at the defence contractor, TRW, and contributed the Constructive Cost Model (COCOMO), the original algorithmic method based on lines of software code (LOC), described in his well-known book [24]. Albrecht, while working at IBM, in a bid to address criticisms of COCOMO with respect to its validity across different programming languages, invented the notion of *Function Points*. The discipline of Function Point Analysis (FPA) was born [25].

With respect to algorithmic methods of estimation, the basic approach is to first estimate the size of the application under consideration, and to then modify the basic size calculation to take into consideration technical and project (environmental) factors that will affect the complexity of delivering the software. Once a measure of size (and a unit of size depending on the method selected) has been calculated and any adjustments made, a number of man-hours can be applied to the satisfaction of each unit of size to arrive at an estimation of labour. The ability of a particular organisation to deliver a notional unit of program size may initially be taken from the literature or, over time, from the company's own records. A perceived advantage of adopting a formal method to produce estimates is that customers may more readily accept them if they know an established method has been used in their production [26].

Based on an agreed metric (LOC, Function Points) the hope was that algorithmic approaches would be more accurate and more transparent. Function Points (FP) tend to be more appropriate to the consideration of data-driven applications whereas LOC have been considered more appropriate in computationally complex, embedded and real time systems development [1].

COCOMO

Introduced in Software Engineering Economics [24] CONstructive COst MODEL (COCOMO) is a model that measures functionality through the lines of programmer's code required to deliver it. According to this approach, required time is a factor of the type of application and the environment in which it is built. Boehm defines three development modes; organic, embedded and semi-detached. An organic project is defined as being developed in a familiar stable environment where the system under construction is relatively similar to those that have been previously developed. The system is relatively small and requires little innovation. An embedded project is subject to tight, inflexible constraints and requires a lot of innovation. An example of a system that would be considered embedded is a real time system with timing constraints and customised hardware. Finally, a semi-detached project is one that falls somewhere in between an organic and an embedded project.

COCOMO is defined as being capable of application at different points in the lifecycle. There is a basic model that is said to be suitable for early rough estimates. It is reported to be accurate to within a factor of two, 60% of the time. The intermediate model is a refinement that employs specific cost drivers such as execution time and storage constraints (*see*: Table 1.4.1), to reportedly produce a more accurate model that is within 20% of the actual time required, 68% of the time.

Table 1.4.1: A list of identified factors identified in COCOMO that are considered to have an effect on effort estimation.

Application attributes	Personnel + Project attributes
Reliability	Analyst capability
Database size	Applications experience
Complexity	Programmer capability
Computer attributes	Virtual machine experience
Execution time	Language experience
Storage constraints	Modern programming practices
Virtual machine volatility	Use of software tools
Turnaround time	Required development schedule

In the detailed model of COCOMO even greater claims are made for the method's accuracy, wherein product and project attributes are factored differently according to their importance at different times in the development lifecycle. For instance, the effect of good programming staff is presumed to have a greater impact during construction, but may have no discernable effect during analysis. The approach has been subject to revision over the intervening years since first published, such as the introduction of COCOMO II [27]. A full description of COCOMO and its variants is outside the scope of this discussion.

It is difficult to test the claims that are made for COCOMO, as there is little in the literature in the way of case studies to support or refute the claims that are made for it [14]. Some doubt has been expressed as to the general usefulness of the approach because of the differences in data collected, project types and environmental factors among sites that may mean models are generally only valid within the organisation in which they are developed [28]. In addition, aspects of development such as planning, design, and testing do not produce LOC, but do consume effort which the method does not explicitly address. Most seriously, a good estimate of the number of LOC must be produced before the model can be used. So the estimate is based on an estimate rather than being strictly based on a measure of functionality [29].

“The major problem with using cost estimation models early in the life cycle (e.g. during bid assessment and/or initial project planning) is that the input data

required by existing models is not measurable early in the lifecycle. Most cost estimation models use system size as an input parameter and models such as COCOMO ... require that size be measured in LOC. It is obvious that LOC cannot be measured from a document such as a requirements document or an invitation to tender.” [30]

COCOMO is a good *theoretical* model, but liable to be a poor model in practice [1]. Lines of code do not necessarily say anything about the amount of functionality that is being delivered *per se*. For instance, two teams, both efficient, each deliver a software solution that features radically different quantities of code, yet which both address the same problem [30]. There is no consideration as to which solution is better; for instance one may be highly configurable, and maintainable. In this respect, it is useful to distinguish between the size of the problem and the size of the solution and to be clear about the distinction.

Function Point Analysis (FPA)

Due to the perceived limitations of measuring software size with LOC, another approach was suggested that provided a measure of software size through a construct termed ‘function points’ (FP) [25]. Starting with the question “what does the application do?” function points are an abstracted unit of functionality. The basic premise of FPA is to identify, count, weight and sum FPs to arrive at an unadjusted function point count (UFC). The UFC is then subject to a technical complexity factor deemed appropriate to the specific project at hand. FPA draws from COCOMO in this respect and attempts have been made to qualify it against LOC as a means of verification. For example, one function point has been equated to 110 lines of Cobol code [24]. Jones reports there is some correlation between estimates produced using FPA and COCOMO and that one can be converted into the other through a process known as ‘backfiring’ [31].

Identification of function points cannot start until the system boundary has been identified. It is important to identify the characteristics of the user domain and the other systems that will be expected to work together. There are two kinds of FP, those that are concerned with the management of stored data, and those that describe user inputs and system outputs. To apply FPA, an organisation must understand how much effort is required to deliver a single function point’s worth of behaviour. This is a measure that is liable to calibration to reflect the efficiency of the particular organisation.

The different types of FP are identified as external inputs, external outputs, external inquiries, and external interface files. External inputs (EI) identify data that crosses the system boundary from outside to inside and may be identified from data input screens or communications with other applications. External outputs (EO) represent data that goes from inside to the outside of the system under consideration. External inquiry (EQ) function points are both input and output components that result in data retrieval. Internal logical files (ILF) are user identifiable groups of logically related data that reside entirely within the application boundary and that are maintained through external inputs. External interface files (EIFs) represent FPs that are user identifiable groups of logically related data that is used only for reference purposes. These definitions, along with a very large body of supporting data, are intended to allow the FP practitioner to identify and make a reliable and repeatable FP count. It is claimed that different people can count and come up with the same results within a reasonable margin of error [1].

One perspective on FPA is that it has a relationship to the basic database management functions known as create, retrieve, update and delete (CRUD). This bias has tended to make FPA more suited to database-driven applications than other types of software [13]. This bias led Symons, in 1983, to introduce FPA Mark II [32] which included an expanded set of adjustment factors. In 1986, the International Function Point Users Group was established as a non-profit organisation to help promote FPA. A large collection of projects and accompanying metrics has been collected of over 20,000 projects. Jones is convinced of the popularity of FPA [31], although there is evidence to suggest he is misguided [18].

Caper Jones states that estimation is a 'parametric' activity, where accurate results will depend on a complex interplay of variables (parameters) [31]. These parameters are both intrinsic and accidental and collecting the correct metrics in a reliable manner is recognised as being difficult. Human factors, such as sheer reluctance to systematically collect data, may play a significant role in making estimates unreliable. Staff members are simply reluctant to keep accurate records, and as a consequence even if organisations are committed to doing so, they may be frustrated. Researchers looking into a historical database may exclude projects where data is incomplete thereby ruling out a potentially significant *tranche* of data. Requirements may be incorrect in the first place, rendering the original FP count inaccurate. The activity selection, production rate, critical path, staff build up and application of technology adjustment factors may well be incorrect. In addition, the pressure of the workplace may

introduce problems such as that of unpaid overtime, which is subsequently not recorded and serves to distort the records. All of these issues have led to the quiet dropping of the technical complexity factor from the original method as it could not be shown that it led to better estimates.

FPA dates from the 1970's; a time before the introduction of graphical user interfaces, client-server, and three tier architectures, when applications were typically command line affairs interfacing with mainframes which were written in low level procedural languages. It is unclear what broad utility FPA offers to the task of effort estimation in a different age of three tier applications and object oriented methods [32].

FP counting is inherently subjective. There is not enough information contained in a typical RFP to allow the method to be applied in early lifecycle estimation [30]. In fact, a great deal of information is required to undertake a FP count, so much so that it may not be possible to make a count until the project is well underway (or indeed finished) [32]. One author has gone so far as to describe the process of performing FP counts as downright tedious [29] and it is a matter of conjuncture as to whether the work of one individual performing a count could be replicated in the work of another.

Originally the FPA method suffered from considerable hype as the industry rode a wave of enthusiasm about something other than COCOMO and LOC as a metric useful in estimation. FPA later suffered from a 'trough of disillusionment' that Symons describes as an inevitability of being swept along as part of the Gartner (an industry research consultancy) 'hype curve'. Today it is not evident that FPA has had a discernable impact on the way effort estimation is typically performed in industry [33].

The Use Case Points Method (UCPM)

The theoretical basis of this method is based on use cases as a basic notation for the representation of functionality. From the outset, it was considered an advantage that the raw data necessary to make an estimate might be more readily available earlier in the project lifecycle than that required by either COCOMO or FPA. Indeed the method, as introduced by Gustav Karner [34], draws heavily on the method described in FPA and some work has been reported that relates use cases to FP [35]. In this approach, the first step is to calculate a measure called the unadjusted use case point (UUCP) count. To the UUCP count is applied a technical adjustment factor, as per FPA, albeit the factors themselves have been changed to reflect the different methodology that underpins development with use cases. In

addition the Use Case Points method (UCPM) also defines a set of environmental (project) factors that contribute to a weighting factor that is also used to modify the UUCP measure.

To apply the UCPM, the practitioner must first start with the identification of actors. Once actors are identified they are ranked according to a scale of simple, average or complex. Characterising the identified actors, Karner suggests that other systems to which the system under consideration must communicate should be ranked as 'simple', or as 'average' if the communication is via a protocol rather than an application programming interface (API). A human actor, interacting through a graphical user interface, is rated 'complex'.

The next step in the application of the UCPM is to identify the relevant use cases. It is then necessary to rate the use cases according to the simple, average, and complex ranking.

To calculate the Unadjusted Use Case Points (UUCP) score, the values of the actors and the identified use cases are summed together.

$$UUCP = \sum_{i=1}^6 n_i * w_i$$

In this formula n_i is a particular actor or use case, and w_i is the weighting that corresponds to the simple, average or complex scale. The weightings that Karner recommends are reproduced in the following table.

Table 1.4.2: Weights attributed to use cases and actors based on their complexity ranking.

Actor type	Actor weights	Use case type	Use case weights
simple actor	1	simple	5
average actor	2	average	10
complex actor	3	complex	15

In order to assign a weight to a use case, Karner recommends allocation according to the number of 'transactions' contained in each use case, although the word 'transaction' is not explicitly defined. In a database context, a transaction is a set of activities that are either performed entirely, or not at all [36]. Another term for transaction, in a use case sense is a 'scenario'. In this example, a scenario is an instance of a text case. A use case is ranked 'simple' if there are 3 or fewer transactions, average if there are between 3 and 7 transactions, and complex if there are more than 7 transactions. Another suggested approach is to equate use cases with the number of analysis classes necessary to realise the behaviour contained in the use case, but this approach is not well articulated. From this description it can be seen that to apply the UCPM, use cases must be reasonably well articulated, at the level of 'focused' on the Kulak and Guiney [37] scale of use case detail. This is a level of detail that is not available at the time of the drafting of a requirements statement typically used in RFPs, although it is an improvement over FPA in that the required artefacts are available earlier and the basis upon which they are ranked is less ambiguous.

Use Case Points Method worked example

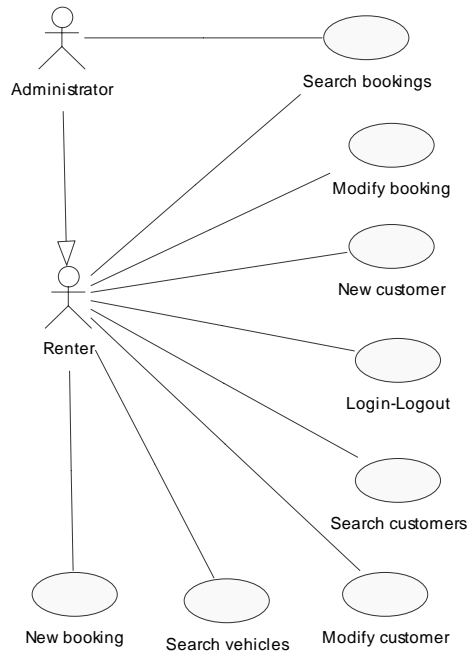


Figure 1.4.1: An example of a use case model for the rental of vehicles at the façade (graphical) level of detail represented at the level of user goals.

Table 1.4.3: An application of the UCPM to the use case model presented in Figure 1.4.1. The Value column is calculated by multiplication. Weight * Number. The Total UUCP score is calculated as the sum of the Value column.

Component	Weight	Number	Value
Simple Actor	1	0	0
Average Actor	2	0	0
Complex Actor	3	2	6
Simple use case	5	1	5
Average use case	10	7	70
Complex use case	15	0	0
Total UUCP score			81

Table 1.4.3 takes the model represented in Figure 1.4.1 and illustrates an application of the UCPM. If there is no more information about the implementation, then Karner states the UUCP score can be used

to make an estimate. He further states that it is possible to get a better estimate if the technical complexity factor (TCF) and environmental factors (EF) are applied to the calculation.

Karner states that the TCF, whose calculation is illustrated in Table 1.4.4, is intended as a factor that reflects the technical difficulty in constructing the application. It is directly based on the Function Point method, with some factors added and others removed to better reflect the experience at of those who originally worked with the method at Objectory Systems in Sweden. To the observer the basis of the weights that have been assigned are not obvious; neither are they described in any of the other algorithmic methods that make use of weighted influencing factors.

Table 1.4.4: Technical complexity factors identified in the UCPM. Each factor is given a rating by the estimator of between 1 and 5.

F_i	Factors contributing to complexity	Weight: (W_i)
F ₁	Distributed system	2.0
F ₂	Performance in response or throughput	1.0
F ₃	end user efficiency	1.0
F ₄	complex internal processing	1.0
F ₅	code must be reusable	1.0
F ₆	installation ease	0.5
F ₇	operational ease	0.5
F ₈	portability	2.0
F ₉	changeability	1.0
F ₁₀	concurrency	1.0
F ₁₁	special security features	1.0
F ₁₂	direct access for third parties	1.0
F ₁₃	special user training facilities	1.0

A factor that is given a value of 3 is neither crucial nor irrelevant. If all factors were to be valued at 3, then the TCF would be 1 and hence it would have no effect on the overall calculation. The calculation of the TCF is determined according to the following formula that follows FPA. The justification for the weights is not given.

$$TCF = C_1 + C_2 \sum_{i=1}^{13} F_i * W_i$$

Where $C_1 = 0.6$ and $C_2 = 0.01$

The objective of the EF, outlined in Table 1.4.5, is to estimate the efficiency of the project. It is based on the same form as the TCF. Each factor is given a rating by the estimator of between 1 and 5. The assignment of the weights to factors is not discussed by Karner.

Table 1.4.5: Environmental complexity factors identified in the UCPM.

F_i	Factors contributing to efficiency	Weight (W_i)
F ₁	familiar with Objectory	1.5
F ₂	part time workers	-1.0
F ₃	analyst capability	0.5
F ₄	application experience	0.5
F ₅	object oriented experience	1.0
F ₆	motivation	1.0
F ₇	difficulty of programming language	-1.0
F ₈	stable requirements	2.0

A factor that is given a value of 3 is neither crucial nor irrelevant. If all factors were to be valued at 3, then the EF would be 1 and hence it would have no effect on the overall calculation. The assignment of weights (C₁ and C₂) is only cursorily discussed by Karner.

$$EF = C_1 + C_2 \sum_{i=1}^8 F_i * W_i$$

Where C₁ = 1.4 and C₂ = -0.03

Karner states that the value of the constants are based on early estimates, that “seem to be reasonable, based on interviews with experienced Objectory users at Objectory Systems” [34]. More explanation would be very helpful, the lack of which casts their veracity into doubt.

To arrive at a final Use Case Point (UCP) score, the following calculation is performed:

$$UCP = UUCP * TCF * EF$$

Once the number of UCP has been determined an effort estimate can be calculated by multiplying the number of UCP by a fixed number of hours. Karner suggests a figure of 20 hours/UCP based on his preliminary figures taken from a small sample of three reported projects. Other authors report the method has merit [26, 38-40] and that its main benefit is that it can be performed very early in the

lifecycle, although this may be with a low level of precision and it is important to revise any estimate as more information becomes available.

To continue with the worked example, as no information is available for the calculation of the TCF or the EF, these values are both set to 1.

$$\text{UCP} = \text{UUCP} = 81$$

$$\text{Hours per UCP} = 20 \text{ (following Karner)}$$

$$\text{Estimate} = 81 * 20\text{hrs} = 1620 \text{ hours}$$

Intuitively, it seems it should be possible to make an estimate of effort from the use case model [40] although it is unclear how this is possible unless the use cases have been 'levelled' so that it can be shown they are comparable. It is critical to know which use cases are going to be counted in the calculation. The consequences of not doing so are that use cases counts will be imprecise and subjective. In Karner's thesis paper [34] his work was intended to continue solely with Objectory customers (*see*: Modelling Languages and the "Method Wars" who might have been expected to produce comparable use cases as they were under the tutorage of the company and expected to decompose use cases to the same notional level of abstraction. It seems inevitable that some decomposition of use case expression must take place, and it must occur in a systematic manner [40]. The IEEE Std 1220 (Standard for Application and Management of the Systems Engineering Process) states in section 6.3.1 that decomposition is performed to understand clearly what the system must accomplish and that generally one level of decomposition is sufficient. There is merit in considering the notion of external use cases as the basis of the identification of what should be counted, and what should not. (An external use case is one with a direct association to an actor.) In this respect, FPA has the advantage of being better established in that there are international standards for counting FP even though it remains problematic. By way of example, consider the case of CRUD (create, retrieve, update, delete) use cases and whether these should most profitably be considered as a single instance, or as a group of four individual use cases [13]. It may well be that the answer to this question is less important than consistency of application.

One notable advantage of the UCPM over FPA is the adoption of a single notation for the representation of the basic input to the algorithm. Whereas the FP method does not require source

documents to conform to a particular standard, use cases, if not exactly standardised, are more formally defined. Counting use cases is not liable to be such a tedious [26] or subjective task as that of counting FP. Combining estimation methods such as expert analogy and the UCPM is liable to be a compatible and practical strategy [38].

The suggestion has been made that the UCPM can be used successfully to measure software size and that its application can normally be done within a couple of hours [40]. These attributes make it a good candidate for an early lifecycle approach to effort estimation. The other algorithmic approaches that have been reviewed are not useful in the early lifecycle. Additionally, the method is easy to learn and apply [39] although assigning values to the TCF and EF suffers from the same subjectivity as in the other methods described. Obviously, there is no reason to apply modification factors where they do not result in better estimates. Far more important is the metric of the number of hours that should be allocated to the satisfaction of a single UCP worth of notional function. This value is liable to be different for each software organisation, and will reflect the relative efficiency of one over another. The job of uncovering the hours/UCP metric however, is not trivial, and it will be intimately linked to the hierarchical goal abstraction at which the use cases are described. It may take some conviction to instigate a programme of estimation based on UCP in the first instance, however, over time the method can be verified with reference to source code, or the number of classes at the end of a project as a database of projects with the correct metrics grows [26]. As Ivor Jacobson himself believes, the UCPM has enough promise to be worthy of further investigation [41].

Effort Estimation Summary

There are no short cuts to accurate estimates [3, 12, 30]. If the client has not described their need at a sufficient level of detail, the supplier cannot be expected to know what they want, and consequently cannot estimate with accuracy [2]. In practice, the models described (with the exception of the UCPM which is too new to be implicated) have performed very badly [30]. Disturbingly, the deviation between actual effort and estimates of effort shows that only 25% of the estimates were within 25% of the actual size. This would appear to markedly contradict claims made for COCOMO and FPA, but seems borne out by the unpopularity of both these methods shown in studies [18, 29]. Equally, it seems unlikely there is a simple model of the relationship among cost parameters which is universally valid. DeMarco in [12] believes there are hundreds of parameters which affect development effort. Watson

and Felix [42] settle on the figure of 29 factors, Bailey and Basili believe there are 21 factors [43] whilst Vosburgh *et al* have identified 14 factors [44]. Yet Kitchenham is of the opinion that there are no factors at all that can be applied universally to arrive at better estimates [30]. One problem is that identified factors are treated as being independent when they are not [17] and the allocation of weights to factors is fundamentally subjective and therefore resists systematic testing and refinement. The suggestion that it is organisational differences between suppliers that accounts for the biggest differences seems most likely. What has been clearly demonstrated is that productivity decreases with increased team size and project duration [17]. Apart from this simplistic summary, very little else has been conclusively demonstrated.

Where in other professions, the repeatability of the task is used to inform future projects, in IT development, the popular consensus has been that the next project is likely to be very different from the last thereby invalidating comparisons [3]. For this reason, practitioners may justify their reluctance to collect metrics that might inform their future estimates; although equally it may be they are unable to identify the appropriate metrics that should be collected. It is natural that unless a metric can be shown to have value that the time required to collect it should be considered an administrative overhead that cannot be justified. Yet good software cost estimation depends on the availability of accurate records of past projects and using a repeatable, defined development process, resulting in a tension that has yet to be reconciled [30].

More recent work by Boehm *et al* [45] treats schedule as an independent variable of the project as a means of responding to an initiative by the U.S. DoD. The essence of an approach that is intended to make a success of this initiative is summarised as relying on the effective management of stakeholder expectations, the prioritization of features, the establishment of a coherent set of core capabilities, planned increments and the careful management of change. The key to this approach reaping benefits is dependent on a sufficiently mature relationship existing between the customer and the supplier so that the scope of the requirements can be made to fit the timetable, rather than assuming that too much can be delivered in too short a space of time. This new work from the originator of COCOMO may be evidence that it has become necessary to move away from the original model because it has not been effective. Unfortunately, this approach cannot be applied to early lifecycle estimation, and Boehm has now become an advocate of a process that is at odds with the general model as it exists in Europe whereby opportunities are publicly published and bids are expected against a fixed statement of

requirements. Public tender would appear to have been abandoned by the U.S. military establishment as a method of acquiring software. There may be a lesson to be learned with respect to the importance of establishing a pre-procurement phase undertaken in an atmosphere of cooperative discussion, yet before this approach is more widely adopted, the military would have to demonstrate that their new approach delivered substantively better results.

A sensible recommendation for improvement in the sphere of effort estimation is that it should be treated in a more systematic fashion that separates the estimation function from the rest of application development. The role of the estimator should be specialised and should not be left to the opinion of project managers who may lack the requisite skill and be subject to partisan pressure. Estimators may well seek to formalise their function were they to be held responsible for the results they produce [12].

One conclusion might be that, where projects go according to plan, this is more likely to be a consequence of late nights and the sustenance of pizza [2] than it is sound engineering practice. Though a pragmatic solution to an intractable problem is elusive, in a review of 100 papers on the subject of effort estimation, Jorgensen has synthesised the recommendations of many authors [14] into ensuring the independence of the estimation, using multiple methods, measuring accuracy against results, and providing training. Another conclusion might be that in the field of software effort estimation, both formal and informal methods exist, they are far from perfect, and much work remains.

- [1.] Agarwal, R., Kumar, M., and Mallick, S., *Estimating Software Projects*. ACM Sigsoft, (2001). **24**(4): p. 60, ACM Press
- [2.] Smyth, P., *Bidding for Software Projects - Verifying Estimates*, (2002), Price Systems L.L.C., Mount Laurel, N.J. USA, www.pricystems.com/downloads/pdf/smyth.pdf
- [3.] Fenton, N. and Pfleeger, S.L., *Software Metrics - A Rigorous & Practical Approach*. (1996), London, Thomson Computer Press
- [4.] Mohanty, S., *Software Cost Estimation: Present and Future*. Software Practice and Experience, (1981). **11**: p. 103-121, Wiley Publishing, NY
- [5.] staff, *The Libra project HC 327*, 2003, National Audit Office (NAO), London http://www.nao.gov.uk/publications/nao_reports/02-03/0203327es.pdf
- [6.] Collins, T., *Public Accounts Committee's Criticism of Libra Has Lessons for the NHS*, 11/11/2003, Computer Weekly, <http://www.computerweekly.com/Article126317.htm>
- [7.] Committee of Public Accounts, T.U.K.P., *Improving the Delivery of Government IT Projects (HC65) - Annex A: Information Technology Projects Examined by the Committee of Public Accounts and the Comptroller and Auditor General*, 2000, London <http://www.parliament.the-stationery-office.co.uk/pa/cm199900/cmselect/cmpublic/65/6509.htm>
- [8.] Collins, T., *Fujitsu Won £800m Post Office Contract After Misrepresenting Itself, says Lawyer*, 29/4/2003, Computer Weekly, www.computerweekly.com/Article121343.htm
- [9.] Collins, T., *Fujitsu Bankruptcy Risk Over Courts IT Contract*, 12/02/2003, Computer Weekly, www.computerweekly.com/Article119287.htm
- [10.] staff, *Gateway Review Leadership Guide*, 2001, Office of Government Commerce (OGC), Norwich www.ogc.gov.uk/embedded_object.asp?docid=704
- [11.] staff, *The Green Book - Appraisal and Evaluation in Central Government*, 2003, HM Treasury, The Stationary Office (TSO), London www.ogc.gov.uk/sdtoolkit/reference/ogc_library/related/Green_Book_03.pdf
- [12.] DeMarco, T., *Controlling Software Projects*. (1998), Upper Saddle River, N.J., Prentice Hall
- [13.] Probasco, L., *What About Function Points and Use Cases*, Rational Edge, Rational Corporation, Cupertino, CA, www.therationaledge.com/content/aug_02/t_drUseCase_lp.jsp
- [14.] Jorgensen, M., *A Review of Studies on Expert Estimation of Software Development Effort*, 2002, Simula Lab, Kysaker, Norway www.simula.no/photo/expertssubmitnovember2002_copy.pdf
- [15.] Shepperd, M., Schofield, C., and Kitchenham, B., *Effort Estimation Using Analogy*. IEEE Proceedings of International Conference on Software Engineering (ICSE'96), (1996). **18** IEEE Computer Society
- [16.] Heemstra, F., *Software cost estimation*. Information and Software Technology, (1992). **34**(10): p. 627-639, Elsevier Science
- [17.] Maxwell, K., Van Wassenhove, L., and Dutta, S., *Performance Evaluation of General and Company Specific Models in Software Development Effort*. Management Science, (1999). **45**(6): p. 787-803, Emerald Management Reviews
- [18.] Taylor, A., *IT Projects Sink or Swim*. The Computer Bulletin, (2001). **42**(1): p. 24-26, Oxford Journals http://www3.oup.co.uk/combul/hdb/Volume_42/Issue_01/
- [19.] Paulk, M., *A Comparison of ISO 9001 and the Capability Maturity Model for Software*, 1994, Software Engineering Institute (SEI), Carnegie Mellon University, <http://www.sei.cmu.edu/publications/documents/94.reports/94.tr.012.html>
- [20.] Herbsleb, J., D., Z., Goldenson, D., Hayes, W., and Paulk, M., *Software Quality and the Capability Maturity Model*. Communications of the ACM, (1997). **40**(6): p. 30-40, Association for Computer Machinery (ACM)

- [21.] Gibbs, W., *Software's Chronic Crisis*, September 1994, p. 72-81, Scientific American, Scientific American Publishing, <http://www.sciam.com>
- [22.] Marciniak, J., *Encyclopedia of Software Engineering*. Vol. 2. (1994), New York, NY, Wiley Publishing
- [23.] Lederer, A. and Prasad, J., *Nine Management Guidelines for Better Cost Estimating*. Communications of the ACM, (1992). **33**(2): p. 51-59, American Computing Machinery (ACM)
- [24.] Boehm, B., *Software Engineering Economics*, Advances in Computing Science. (1981), Upper Saddle River, N.J., Prentice Hall
- [25.] Albrecht, A.J.: *Measuring Application Development Productivity*. in *IBM Applications Development Symposium*. (1979), p. 83-92, Monterey, CA: International Business Machines (IBM)
- [26.] Anda, B., Dreiem, H., Sjoberg, D., and Jorgensen, M.: *Estimating Software Development Effort based on Use Cases*. in *4th International Conference on the Unified Modeling Language*. (2001), 487-502, Toronto, Canada: Springer Verlag <http://www.idi.ntnu.no/emner/sif8080/docs/faglig/uml2001-anda.pdf>
- [27.] Boehm, B. and Horowitz, E., *Software Cost Estimation with Cocomo II*. (2000), Pearson Education
- [28.] Bailey, J. and Basili, V.: *A Meta-Model for Software Development Resource Expenditures*. in *5th International Conference on Software Engineering (ICSE81)*. (1981), p. 50-60, San Diego, CA
- [29.] Pengelly, A., *How long is a piece of string?*, Computer Bulletin, British Computing Society (BCS), <http://www.bcs.org/BCS/Products/Publications/JournalsAndMagazines/ComputerBulletin/OnlineArchive/mar98/forum.htm>
- [30.] Kitchenham, B., *Estimation*, in *Software Metrics - Rigorous and Practical Approach*, p. 132 N. Fenton, Editor. (1991), Chapman and Hall, London.
- [31.] Jones, C., *Estimating Software Cost*. (1998), N.Y., N.Y., McGraw-Hill
- [32.] Symons, C.R., *Function Point Analysis : Difficulties and Improvements*. IEEE Transactions on Software Engineering, (1988). **14**(1): p. 2-11, IEEE Computer Society <http://portal.acm.org/citation.cfm?id=630951&dl=ACM&coll=portal>
- [33.] Symons, C.R., *Come Back Function Point Analysis (Modernised) - All Is Forgiven*, Software Measurement Services, accessed: 2003, Edenbridge, Kent, <http://www.gifpa.co.uk/library/index.html#papers>
- [34.] Karner, G., *Resource Estimation for Objectory Projects*, 1993, Masters thesis, Linköping Institute of Technology, Linköping, Sweden, LITH-IDA-Ex-9344:21,
- [35.] Fetcke, T., Abran, A., and Nguyen, T.-H.: *Mapping the OO-Jacobson Approach into Function Point Analysis*. in *Technology of Object-Oriented Languages and Systems*. (1998), p. 192-202, Santa-Barbara, California: IEEE Computer Society <http://user.cs.tu-berlin.de/~fetcke/en/publications.html#Fetcke1997>
- [36.] Connolly, T., Strachan, A., and Begg, C., *Database Systems - A Practical Approach to Design, Implementation and Management*. (1995), Harlow, Addison Wesley Longman
- [37.] Kulak, D. and Guiney, E., *Use Cases - Requirements in Context*. (2000), Upper Saddle River, N.J., Addison Wesley Longman
- [38.] Anda, B., Angelvik, E., and Ribu, K.: *Improving Estimation Practices by Applying Use Case Models*. in *4th International Conference on Product Focused Software Process Improvement, Rovaniemi, Finland, December 9 - 11, pp. 383-397, LNCS 2559, Springer-Verlag*. (2002), Profes, Finland http://www.simula.no/publication_one.php?publication_id=500
- [39.] Ribu, K., *Estimating Object-Oriented Software Projects with Use Cases*, 2001, Masters thesis, University of Oslo, Oslo, Norway, www.stud.ifi.uio.no/~kribu/oppgave.pdf
- [40.] Smith, J., *The Estimation of Effort Based on Use Cases*, November 1999, The Rational Edge, <http://www.rational.com/products/whitepapers/finalTP171.jsp?SMSESSION=NO>
- [41.] Jacobson, I., *Use Cases: Yesterday, Today and Tomorrow*, IBM Developerworks, accessed: 2003, www-106.ibm.com/developerworks/rational/library/775.html
- [42.] Watson, C. and Felix, C., *A Method of Programming Measurement and Estimation*. IBM Systems Journal, (1977). **16**(1): p. 54-73, International Business Machines (IBM)

[43.] Bailey, C. and Dingee, W.: *A Software Study Using Halstead Metrics*. in *ACM workshop/symposium on Measurement and evaluation of software quality*. (1981), p. 189-197, University of Maryland: ACM Press <http://portal.acm.org/citation.cfm?id=807928&dl=ACM&coll=GUIDE>

[44.] Vosburgh, J., Curtis, R., Wolverton, B., Albert, H., Malec, H., Hoben, S., and Liu, Y.: *Productivity Factors and Programming Environments*. in *International Conference on Software Engineering (ICSE)*. (1984), 143-152, Orlando, Florida: IEEE Computer Society <http://portal.acm.org/citation.cfm?id=801963>

[45.] Boehm, B., Port, D., Huang, L., and Brown, W., *Using the Spiral Model and MBASE to Generate New Acquisition Process Models*. Crosstalk - The Journal of Defense Software Engineering, (2002) Department of Defense (DoD) www.stsc.hill.af.mil/crosstalk/2002/01/boehm.html