iMAPS: Collecting Data for Software Costing

Gina Kingston, Martin Burke and Peter Fisher

DEPARTMENT OF DEFENCE
DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

iMAPS: Collecting Data for Software Costing

Gina Kingston, Martin Burke and Peter Fisher

Information Technology Division
Electronics and Surveillance Research Laboratory

DSTO-GD-0090

ABSTRACT

This paper discusses the iMAPS Software Costing conjectures, and documents the data required to calibrate and validate the models. It discusses issues related to the collection of the data, including the benefits to participants, and the significance of this research to the Australian Defence Organisation.

APPROVED FOR PUBLIC RELEASE

DEPARTMENT OF DEFENCE

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION

Published by

DSTO Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury. South Australia, Australia 5108

Telephone: (08) 259 7053
Fax: (08) 259 5619

AR-009-686
April 1996

APPROVED FOR PUBLIC RELEASE

iMAPS: Collecting Data for Software Costing

Executive Summary

Software is an increasingly important element in modern Defence systems. A large proportion of the Australian Defence Organisation's (ADO) budget is currently committed to the procurement and maintenance of software based systems. Despite this, the process of estimating and monitoring Software Costs for ADO Projects is ad-hoc. This can result in project Costs exceeding the original budget, or systems being delivered with sub-optimal functionality. Moreover, current approaches to Software Costing have limited applicability to Defence systems.

The integrated Measurement, Assessment and Prediction of Software (iMAPS) task is motivated by the need for a systematic approach to Software Costing at all stages of the Defence acquisition process. This document describes the iMAPS Software Costing conjectures and the data collection activities required to support the research.

A two-phased approach to Software Costing is proposed. The result of the first phase, Cost Slicing (see Section 2.5), would be a coarse initial estimate which could be used early in the acquisition process when comparing alternative methods of obtaining a specific Defence Capability, and as input to Defence Force Capability Options or Capability Proposals. This phase would be followed by an iterative process, Progressive Refinement (see Section 2.6), to refine the Cost estimates during the remainder of the acquisition process, including both before and after contract negotiation.

In order to analyse and calibrate (see Section 4) this approach, software cost data will be collected from the ADO. The quality and quantity of the data provided will directly effect the accuracy and precision of the predictions made by the approach.

The confidentiality of the data is of utmost importance. Only aggregated results, which in no way disclose the source of the data, will be published. Furthermore, the data will not be used to evaluate or compare organisations or projects. The names of organisations and projects which contributed information will only be released if express permission is given. Even in these circumstances, the names of the organisations will not be associated with data they supplied.

The data will be collected with the assistance of a facilitator, who may use a combination of interviews, questionnaires and collation of existing data to collect information from the Developers, Clients and product Users. The Questionnaire in Appendix A provides a record of the information to be collected. However, the data requested falls into two main categories (Group 1 and Group 2) and it is possible that most of the information for the larger group will be already stored electronically and will merely require copying. Automated support for the collection of Group 2 information should be available for on-going projects. It is anticipated that a person familiar with the project will require about one hour to answer the questions for Group 1. A small reduction in the effort required for data collection could be achieved by restricting the information collected to the compulsory questions for each group.

All participants in the data collection activities should benefit through early and preferential access to the Software Costing models developed using the data. Industrial participants who collect related data for their own use should quickly be able to customise these models for their organisation. Such customisation has already been shown to improve the accuracy and precision of Software Costing models. Early benefits to the Australian Department of Defence should arise from the creation of a database of Defence specific projects which could be used for informal Costing of similar projects.

Contents

Page No.

1. Introduction

Objective

This document describes the iMAPS Software Costing conjectures and the data collection required to support the research. This document focuses on the kinds of information which will be required in the short term, and indicates the provisions which should enable these early data collection efforts to benefit longer term analyses as well. It identifies different types of data, and the mechanisms by which each type of data could be collected.

Motivation

Software is an increasingly important and costly component of many Defence systems. Methods for estimating the Development Costs of such systems tend to be inaccurate and imprecise. As a consequence, the software acquisition process tends to be high risk for both the Australian Defence Force (ADF) and the industries involved. The iMAPS Software Cost Prediction research aims to reduce the risks for all parties. However, relevant data must be obtained to develop Costing models suitable for Defence Projects. Organisations which choose to supply data should benefit from early access to the results of this research and increased awareness of their Software Development Process.

Context

This work forms part of the DSTO iMAPS task DST 93/349 [Burke, 1995], a 3 year DSTO task which aims to provide an integrated approach to the description, measurement, assessment and prediction of software attributes. A general introduction to the iMAPS approach is given in [Burke, 1994].

Research on the Software Cost Prediction foci of the iMAPS task was started in mid-1994 and an initial statistical investigation using public domain data has already been undertaken [Kingston et al, 1995c] through a Co-operative Education Enterprise Development (CEED) agreement [Kiermeier, 1994] with the University of Adelaide. Current work is focusing on the development, evaluation and refinement of a new model for Software Development Cost Prediction and will contribute towards Gina Kingston's PhD studies.

Assumptions

The Cost associated with a software product has many elements including: the cost of the hardware it will be developed on, travel costs, and Software Development and Maintenance costs. The iMAPS Cost Prediction thread will focus on just the Software Development Cost. Development Cost is assumed to be primarily due to the cost of labour and may be thought of as Development Effort multiplied by the average cost per unit Effort.

Intended Readership

This document has been prepared for use by the iMAPS team at DSTO and external parties, such as TTCP partners, Science Policy Division (SP) and, specific ADF and industry project managers, who may be able to assist in data acquisition. Sections of it may also be of use to other members of the Software Engineering Group and ITD's executives.

Layout

Potential data suppliers are advised to read at least Sections 3.1 and 3.2. They may also wish to view the preliminary Costing questionnaire in Appendix A and the information on Collection Mechanisms in Section 3.3.

Potential facilitators (see Appendix B) are advised to read at least Sections 2 and 3 and both Appendices, although Section 2 is more technical. Readers wishing to familiarise themselves with the conjectures should read Sections 2.3 to 2.6 and those requiring an appreciation of the scientific context of the work should read Sections 2 and 4.

The iMAPS team are advised to read all sections of this paper.

Section 2 provides a brief background to Software Costing, with particular emphasis on those established and new ideas on which data will, or may, be collected for the iMAPS task. The new conjectures which will be investigated using this data are summarised at the end of the section.

Section 3 describes the data collection requirements and indicates how the data may be captured. Section 3.1 should be read by anyone interested in the type of data being collected. Section 3.2 describes some of the considerations for potential participants in the data collection. Section 3.3 discusses mechanisms by which the data may be collected and constraints on the data.

Section 4 describes the analyses which are to be performed on the data, and indicates how various subsets of the data could be used during the different analyses.

Section 5 summarises the current status of the data collection work and the way ahead.

A preliminary Costing questionnaire and guidelines for data collection facilitators are included as Appendices.

2. Background

A substantial amount of literature is available on Software Cost Explanation techniques dating from the 1960's. (See [Kemerer, 1991] for a summary of approaches commonly used.) Most of this work relates some measure of Software Size with Effort. While the correlation between Cost and Size is statistically significant [Kitchenham, 1992], it is not strong enough to provide useful Cost Estimates. Therefore, adjustment factors are usually added and linear regression is reapplied to try to improve the model. In addition, most of the work focuses on Effort Explanation, rather than Effort Estimation or Prediction (See Section 2.1).

Literature surveys on Software Costing and Software Sizing Techniques are currently being conducted and papers presenting the results are being prepared [Kingston et al, 1995a; Kingston et al, 1995b]. The conclusions so far, are that the results given by the current state-of-the-art techniques are still of limited accuracy and rarely attempt to quantify precision. Correlation efforts [Kitchenham, 1992; Kingston et al., 1995c] have shown that, out of all measures suspected to influence Software Cost, measures of Software Size have the greatest correlation with Software Cost. They also show that most of the adjustment factors used in current models have little or no correlation with Development Effort. The only adjustment factors which appear to have statistical significance are the Development Environment and language.

Section 2.1 explains some of the differences between the terms Estimation, Explanation and Prediction. Section 2.2 reviews different established measures of Software Size. The measures described here include the two most commonly used measures, Lines Of Code and Function Points. These measures will be used in the evaluation of the new approach. A more complete review is currently being prepared for publication [Kingston et al, 1995b]. Section 2.3 introduces a new measure conjectured as part of the iMAPS task. Section 2.2 and 2.3 examine some of the benefits and limitations of using the different Size measures for Predicting Software Development Effort and Cost. Sections 2.3 to 2.5 examine the conjectures proposed by the iMAPS Cost Prediction team.

2.1 Estimation, Explanation and Prediction

The terms Estimation, Explanation and Prediction are often used inconsistently and incorrectly in the Software Costing field. This section provides definitions of these and related terms as they are used throughout this and related iMAPS documents. Figure 1 depicts the relationships between the terms.

Definition: Costing

Costing is the generic term used to describe all work on developing models for Software Development Costs, regardless of their use, testing, or how they were developed. In other contexts it may also be used to refer to models of Software Maintenance Costs.

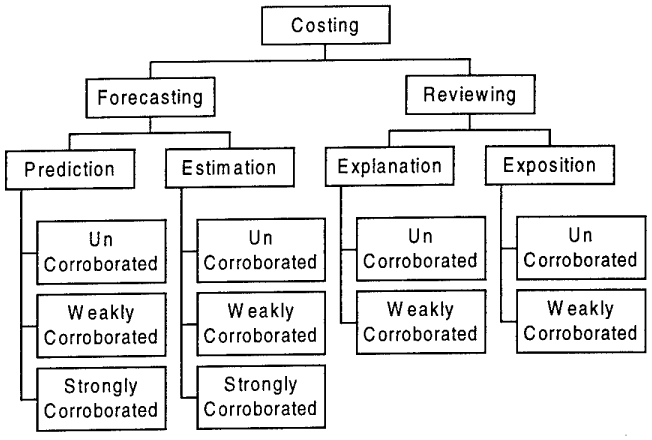

Figure 1: Costing Terminology

Definition: Reviewing
The first stage in developing a Software Development Costing model is normally to explore the potential model using historical data. Models which use data obtained after project completion will be termed Review models.

There are two types of review models - Explanation and Exposition. Explanation models are the simplest.

Definition: Explanation
Explanation models are the first (stage in) models developed to explain the behaviour of past projects. They contain estimates of their accuracy; that is, they have been checked for biases. Most models in the literature only deserve to be called Explanation models even though they are often claimed to be useful for Estimation.

Definition: Exposition
An Exposition model is an Explanation model which has been enhanced by a mechanism for determining the precision of the model, such as Prediction Intervals [Matson et al., 1994].

Definition: Forecasting
Unlike Review models, which are retrospective, those which are to be used for Costing a project during its Development should be developed from data obtained before and/or during the Development.

There are two types of such Forecast models- Estimation and Prediction models. Estimation models are the simplest.

Definition: Estimation
A model is termed an Estimation model if it is a Forecast model whose accuracy has been determined.

Definition: Prediction
A model is termed a Prediction model if it is an Estimation model for which Prediction Intervals [Matson et al., 1994] have been determined.

The validity of Software Costing models may be checked using a variety of techniques and this method can be used to further classify the models.

Definition: Un-Corroborated
Un-corroborated models have their accuracy (and precision) checked for biases using only the information used to develop the model. Checking techniques include statistical checking such as statistical checks for differences between models, Mean Residual Error checks and heteroscedastity checks (tests for changes in error with Size). All types of Costing models may be Un-Corroborated.

Definition: Weakly Corroborated
When Weakly Corroborated models are developed, a randomly chosen subset of the available data is set aside for testing. After the model has been developed from the remaining data, from one or more projects, Effort estimates are obtained from the test data and compared to the Actual Effort. Accuracy is measured by checking for biases in these estimates. Precision can be determined from the absolute errors in the estimates. All types of Costing models may be Weakly Corroborated.

Definition: Strongly Corroborated
Strongly Corroborated models are similar to Weakly Corroborated models, in that the model is checked using data which was not used for the development of the model. The difference is that Strongly Corroborated models are tested on new projects where the estimates are used during the development process, not just on projects for which data was initially collected. Therefore, only Forecast models can be Strongly Corroborated. Models may be Strongly Corroborated to check against biases in the initial data collection, and biases introduced by the estimation process. For example, the development of many of the existing models only used data from successful projects.

2.2 Current Sizing Techniques

This section looks at several different measures of Size. The first two, Lines Of Code (LOC) and Function Points, can be found in the Cost-Estimation literature [Kemerer, 1990]. The other measures are specific to large Ada projects. The Capacity measure being developed by the iMAPS Software Cost Prediction team as an alternative measure of Software Size is discussed in Section 2.3. A more detailed review of existing Software sizing measures and their application to Software Costing is discussed in a paper currently being prepared [Kingston et al, 1995b].

Lines Of Code
One of the most commonly used Size measures is Lines of Source Code. It can be readily measured from the source code, provided a sensible definition of a line of source code is used. This definition must account for the inconsistent use of "white space", including

carriage returns, as most programming languages allow it to be arbitrarily included or omitted at many locations within the code.

Benefits:

- Simple concept.
- Can be automated.
- Widely used.
- Contains no uncertainty when measured directly provided clear definitions (or automated calculation methods) are used.
- May be useful for predicting maintenance and/or testing costs (although complexity measures may also be necessary, or may be superior).

Limitations:

- Language dependent.
- Can only be measured after the completion of coding.
- Can be, and is, counted in many ways using different definitions. This introduces uncertainty into comparisons between different systems.
- Does not consider the complexity of the code.

Function Points

Function Points are another widely used measure of Software Size. Function Points were developed for Transaction Processing systems and have become established in this area.

Function Points were designed to:

- Be measured before coding commences (cf Lines of Code).
- Be measured after functional and data decomposition have been performed.
- Capture the transactions which occur at the interface between the software and both the users of the software and other systems.

The Function Point count for a system is determined by calculating the raw function point (or base) count and applying 14 adjustment factors. The base count is determined by considering the user requirements according to five categories: Inputs, Enquiries, Outputs, Internal Logical Files and External Logical Files.

Benefits:

- Can be measured earlier than Lines of Code.
- Language and technology independent.

Limitations:

- Original choice of components and weights in the definition was arbitrary.
- Best measured after system design.
- Does not handle uncertainty.
- Applicable to a limited range of applications or systems - in particular transaction based systems. Most of the functionality of these systems is tied to the five categories (often broken down into data and transaction categories) on which Function Points are based. Function Points do not provide a suitable measure of Size for systems where there are large amount of functionality which are not tied to transactions, such as real-time, scientific, and other systems with a high degree of internal processing.

- Definitions are subject to different interpretations.
- Counting cannot be fully automated.
- Time consuming.

Function Points and Lines of Code have been correlated for a number of different languages [Albrecht and Gaffney, 1993]. This suggests that conversions can be made between Function Points and Lines of Code. This has two main purposes. The first is so that Function Point Counts can (effectively) be used as the input to Cost Estimation tools based on Lines of Code. (Correlation between different Size measures is seen as an important issue by many researchers, although this may preclude measures which are "better" in some sense which do not correlate with other Size measures.) The second is so that historical data, based on both Function Points and Lines of Code, can be used to calibrate tools. However, there are a number of problems with this approach including:

- Function Points were designed to capture Software Size for Transaction Processing systems and are not always accurate for other types of systems. Lines of Code are a valid measure of Software Size for all types of systems. Thus correlations between them would only be valid for Transaction Processing Systems. This is only a problem if the counts are misused, as in the example below.

  If the correlation is used in inappropriate circumstances, a Function Point count could be used to derive a meaningless Lines of Code count. If the original Function Point count is then thrown away, the Lines of Code count could be treated as an accurate estimate of Software Size.
- Different adjustment factors are generally used when correlating Function Points and Lines of Code with Cost. It is difficult to see if and how these should be taken into account when doing the correlation.
- Function Point counting is generally performed by decomposing the system into sub-systems based on related (functional) areas. If Function Points are measured before the system is designed, the designers may make use of this decomposition. This may influence the design of the system and may affect the eventual number of lines of code.
- Errors are introduced into estimating processes which use historical data obtained using the correlation, as the process is not perfect.

Ada Specific Measures

A number of Ada specific measures are also described as they are relevant to the systems of interest to the ADF. It is not intended to restrict our research on Software Costing to Ada applications and alternative Size measures may be provided for applications in other languages. In addition, it is our intention to collect information on any Size measures which are currently being used.

Previous Cost Explanation models have tended to be developed using small to medium sized applications typically written in Cobol, C or C++. By contrast, Defence projects are typically large, Ada projects. Because these systems are large, they are generally broken down into a hierarchy of *sub-systems*. The sub-systems themselves consist of a collection of *Ada library units*, which are the smallest pieces of Ada code which can be separately compiled and stored in the Ada library. The only mechanism of encapsulation within the

library unit is the *Ada package*. These three constructs – sub-systems, Ada library units, and Ada packages – are our Ada specific measures.

Sub-system Counts

Benefits of sub-system counts:

- Available before detailed design.
- Can be automatically obtained from some design tools and Development Environments.

Limitations of sub-system counts:

- Not available before system design.
- Cannot be determined from the source code as there is no language construct to support the sub-system abstraction.
- Subjective. They are determined by the system designers, sometimes in accordance with loose guidelines. Therefore, they can be of arbitrary Size.

Ada Library Unit Counts

Benefits of Ada library unit counts:

- Can be automatically obtained from code and some design systems.
- Available before coding.

Limitations of Ada library unit counts:

- Design dependent. They are determined by the system designers, sometimes in accordance with loose guidelines. Therefore, they can be of arbitrary Size.
- The concept of library units is specific to the Ada language.
- Ada "separate"s can be treated in two ways. (Ada "separate"s are files which contain sections of code which are logically part of a library unit, but which can be re-compiled separately [Barnes, 1993].) They may either be counted towards the library unit count (as they may have been separated because they are large, complex or volatile components) or they may be ignored as they are not actually a library unit.

Ada Package Counts

Benefits of Ada package counts:

- Can be automatically obtained from code and some design systems.
- Available before coding.

Limitations of Ada package counts:

- Design dependent.
- Language dependent although similar concepts exist for other languages.
- Only available after detailed design. They are determined by the system designers, sometimes in accordance with loose guidelines. Therefore, they can be of arbitrary Size.
- Variations can exist between the counts of a given system as special types of packages, such as *Generics*, can be treated in several ways. A well defined counting mechanism would eliminate this problem.

2.3 Conjecture 1: Capacity

Capacity is proposed as a new candidate Software Size measure. It was conjectured after considering the benefits and limitations of existing measures of Software Size. Capacity

only captures what the software system does (will do) and does not consider how it does (will do) it. It looks at what the software directs the computer to do. It is asserted that computers can perform two main activities - communicating with external devices and processing information (both symbolic and numeric). Software can make computers appear "clever" by making them do many of these types of activities quickly. Capacity is a measure of how much the software makes the computer do - that is, how much of communicating and processing the software controls. Appendix A, Part C contains a table which gives possible examples of Capacity levels and indicates the amount of device control and processing which may be present at each level. An ITD report describing the concept of Capacity in more detail is being prepared [Kingston and Burke, 1995d].

> **Definition (Preliminary): Capacity**
> The Capacity, C, of a Software System is a measure of its Size or Functionality. It is defined as an increasing function of the number of Basic Manipulations (BM) which must be performed by the Software System to deliver its functionality.
>
> A Basic Manipulation is a Basic Data Transfer or a Basic Data Transformation. In practice, these Basic Manipulations are not measured directly, but are combined into Basic Computing Functions which are easier to count.

Alternative methods of determining a system's Capacity are also being investigated.

Capacity is designed to have very different properties to the preceding three measures of Size. Capacity's properties include:

- Capacity can be naturally quantised in broad categories.
- Capacity can be used without fine-grained knowledge of the project.
- Capacity can be measured very early in the Software Development - after the initial requirements have been captured.
- Capacity is currently evaluated by comparison with reference systems which may be local to the Development Environment or global standards. This tends to identify bands of Capacity around the reference systems.

As the practice of making Capacity evaluations matures, it should be possible to define increasingly precise bands of Capacity. Capacity measures of increasing precision could then be used (with other measures) to refine Cost estimates during the Software Development.

It is hoped to eventually develop a scale for Capacity. (In the same way as scales for temperature were developed from the concepts of hotter and colder, the development of a scale for Capacity first requires investigation into differences and similarities in the Capacities of Software Systems.)

Benefits of the initial concept are:

- Can be determined early in the Development.
- Uncertainty is handled by quantising levels.
- Quick.

Limitations of the initial concept are:

- Subjective.
- Cannot be automated.
- Currently only one very broad breakdown of Capacity is available (see Appendix A, Part C).

Some of these limitations will be addressed during the first phase of the investigations into Capacity.

The main hypothesis of the iMAPS Cost Estimation Team is that Software Capacity and Difficulty (described later) will correlate well with Cost quanta.

2.4 Conjecture 2: Difficulty

A large number of fine grained adjustment factors have been proposed as modifiers for Software Cost estimates. They are typically applied by giving a ranking to the influence of the factor (eg on a scale of 1 to 5), combining the factors using a weighted arithmetic sum, and multiplying the Size by the resultant number. This approach has several problems including:

- Most of the factors are not statistically significant
  (According to [Kitchenham, 1991] and [Kingston et al; 1995d] the only currently considered adjustment factors which are significant, are Development Environment and programming language.)
- The determination of the ranking is subjective.
- The factors are not independent but, by using a weighted arithmetic sum, they are treated as if they are.

However, Software Size alone does not correlate well with Software Cost [Jeffery and Low, 1990]. Boehm's Basic COCOMO relates Cost to Size using an exponential relationship [Boehm, 1984]. However, when used on the 63 project data set on which it was developed, it is said to be accurate to within a factor of 2 only 60% of the time [Heemstra, 1992]. It is expected that the results could be significantly worse on other data sets.

Difficulty

We have conjectured that there are two factors which can be determined early in the Software Development, which can be used to predict the Development Cost. The first is the measure of Software Size we call Capacity. The other factor, called Difficulty, is a measure of the effect the Development Environment and product constraints have on the Effort required to develop a software system. These factors would be used in the Cost Slicing Method (see Section 2.5) to obtain a coarse Prediction of the Software Development Effort which would then be refined (see Section 2.6) to obtain more precise estimates later in the Software Development.

Definition (Preliminary): Difficulty

The Difficulty, D, of a Software Development is defined by the equation $E = DC$, where E is the Effort required for the Development and C is the Capacity of the Software System developed.

One measure for the Difficulty concept is a function of the three factors: Process, Product and Resource. (These are described below.) The following example (which uses the terminology introduced below) shows that we cannot assume that the factors are independent.

Example:

1. To develop a simple product (where errors are not very likely) costs *less* when there are less "checks" in the process.
2. To develop a complex product costs *more* when there are less "checks" in the process (because it tends to cost more to correct errors found late in the development process).

Thus, an increase in the number of "checks" in the process may either increase or decrease the Cost of the project depending on the complexity of the product.

Difficulty is a complex function, which, for the purpose of obtaining coarse Software Development Cost Predictions, may never need to be investigated in detail. Therefore, an approximation is needed. One commonly used approximation technique is to consider functions as a weighted arithmetic sum of their inputs. From the example above, it can be seen that this approach would not be sufficient for Difficulty. The initial approach to be taken would involve two steps:

- Quantising Difficulty, so that it can only take a discrete range of values.
- Using a look-up table to define the mapping from Process, Product and Resource to Difficulty.

This should be sufficient for these preliminary investigations and would also allow Process, Product and Resource to be quantised.

Process

The Process attribute is a coarse measure which should capture the rigour of the method used to develop the process. It is provisionally assumed that Process can take three values:-

- Ad hoc: Poorly defined and controlled process.
- Controlled: Well defined process, with some control.
- Intensive: Well defined process, with strict controls.

Things which should be taken into account when considering how rigorous a process is include: the method used to handle changes to the requirements, what information is measured and how it is recorded and used, and the independence and nature of software evaluations.

Product

The Product attribute is a measure of the constraints placed on the product due to the environment in which it is to be developed, maintained and operated. Constraints include things such as: Availability, Reliability, Maintainability, Safety, Security, Storage and Timing requirements. It is provisionally assumed that there would be three grades of Product:

- Easy: No constraints or demands.
- Intermediate: A few compatible constraints or demands.
- Difficult: Many or conflicting constraints and demands.

Resource

The Resource attribute is a measure of the availability of human, financial, temporal, and computing assets during the project Development. Resource is also a quantised attribute. The definition given below gives the coarsest possible quantisation, which will be assumed in this study.

- Ample: Majority of resources readily available at a suitable or better level.
- Constrained: Majority of resources at a low level of availability or suitability.

2.5 Conjecture 3: Cost Slicing

A two phase approach to Software Cost Prediction is proposed where a coarse-grained prediction would be obtained early in the Software Development, such as at the feasibility analysis stage, which could be refined as the Development proceeds (see Section 2.2.4). Cost Slicing is conjectured as the method of determining a coarse-grained Cost Prediction. It should have the following features:

- Quick and easy to use.
- Determines Cost estimates as intervals.
- Can be used early during Software Development.
- Can be used in several ways.
- The Cost component of interest is assumed to be correlated with the development Effort.
- Assumes that Capacity and Difficulty are the two main factors which determine Software Development Costs.

Figure 2 shows one possible relationship between Cost (Effort), Capacity and Difficulty.

An alternative way of viewing this relationship is shown in Figure 3. In Figure 2 the quantisation of Difficulty is clear, but it is not clear that Capacity is also quantised (as lines connect the distinct capacity values). In Figure 3 the quantisation of both Capacity and Difficulty is clear. In this figure Cost appears to be quantised. However, it is likely that the Effort (or Cost) will not be quantised in the initial investigation. Quanta may be determined from the investigation.

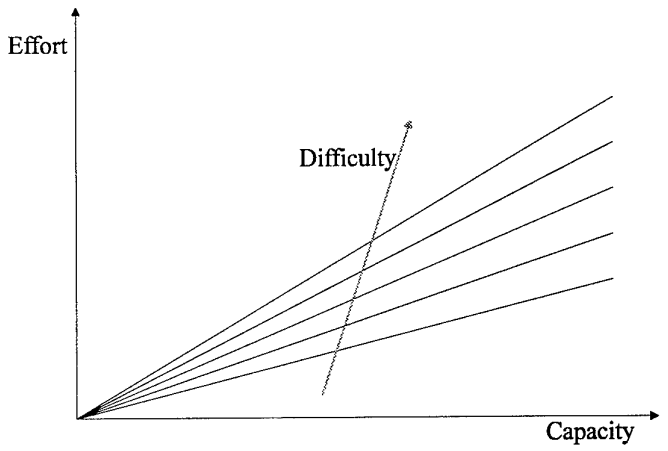

Figure 2: Effort Estimates Changing with Difficulty

The Cost Slices as shown in Figure 2 and Figure 3 could be used in a number of ways:

- To determine the local Difficulty given a project of known Cost and Capacity.
- To determine the Capacity which can be delivered for a fixed Cost given a particular Difficulty.
- To determine how much improvement is required in the Difficulty to achieve a given Capacity for a given Cost.
- To determine the Cost of Software Development with a given Capacity and a given Difficulty.

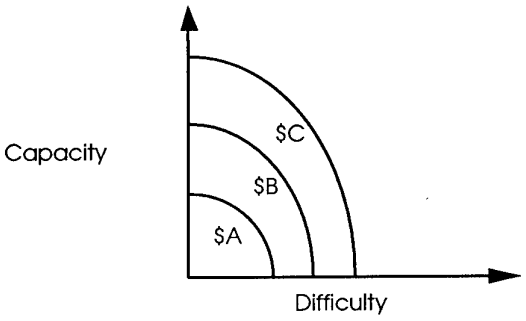

Figure 3: Cost Slicing Model

2.6 Conjecture 4: Progressive Refinement

Progressive Refinement is the conjectured method for refining Cost Predictions. It is the second phase of a two phase approach to Cost Prediction which commences with Cost Slicing (see Section 2.5). It incorporates:

- updated information (eg Effort estimates due to a change in the scope of the project),
- more detailed information of existing types (cf Effort estimates based on actual rather than estimated KLOC),
- new types of Cost information (eg Effort estimates based on FP and KLOC) and actual Costs for the Development stages completed.

It will use methods for:

- refining estimates,
- identifying patterns in changing estimates and,

- identifying high-risk situations where refinements either:
  - don't conform to an existing pattern, or
  - conform to a pattern of escalating Costs etc,
- determining the precision of the estimates in all these circumstances.

The Cost of developing a software system cannot be determined precisely early in its Development. However, as more information becomes available during the Development, the Costs should be increasingly precisely and accurately determined. Fortunately, in most circumstances, while a Cost estimate is required early during the Software Development, it is not required to be precise.

This motivated the development of the notion of Progressive Refinement, where an initial imprecise estimate would be made and then, during the Development of the software, this estimate would be refined. While the details of the Progressive Refinement approach still need to be developed, the data required to investigate the conjecture and the assumptions it relies on have been determined.

One assumption which is useful when considering Progressive Refinement is: A stable process exists which can be regarded as a series of stages, with the ratio of Effort in each stage being approximately constant between projects.

Given this assumption, the estimate can be refined when:

- A Development phase is completed and the Cost for that phase is known. For example, when the Requirements or Specification Phases are completed.
- When additional information becomes available. For example, when Function Point Counts or Lines of Code counts become available.
- When different builds of the software are complete. For example, at the end of an iteration for a product being developed using a Spiral Development Process.

The circumstances when the estimates could be refined depend on the process being used to develop the software and the desired rate of refinement. If the assumption is violated, and the process is ad-hoc, then the estimates could only be refined when new deliverables were produced, or on a 'regular' basis.

When a well-defined and controlled process is in place, the ratio of the Costs (Effort) for each stage of the Development should be similar between different Development projects. (A stage is a combination of Development phases and builds.) An example, where the process has been broken into five stages, is shown in Figure 4. It shows how the ratios a, b, c, d and e can be obtained by normalising the total Effort. The curved line shown is the observed Effort (normalised) over time. A step-wise approximation to this function is also shown. In practice, the ratios would be determined from a number of historical projects.

Thus at the end of each phase or build, or both (depending on the process being used) the Cost could be re-estimated using:

- The Cost of the Development to date, and
- New information uncovered during the previous stage.

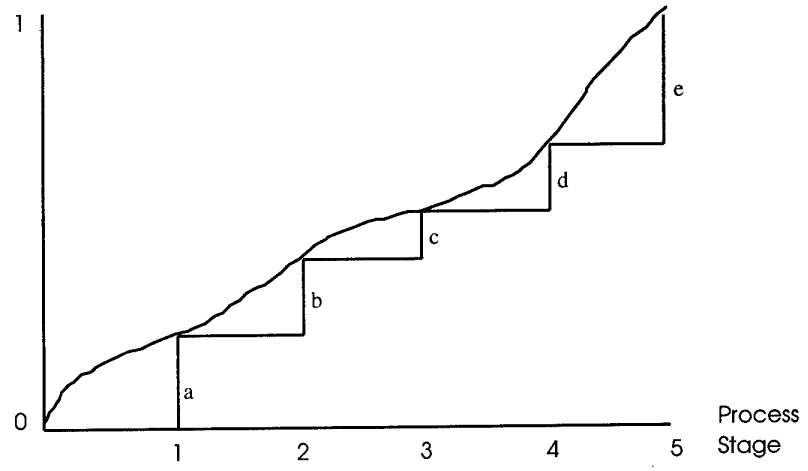

Figure 4: Ratios of process phases.

Figure 5 shows an example of how the estimates could change over time using Progressive Refinement.

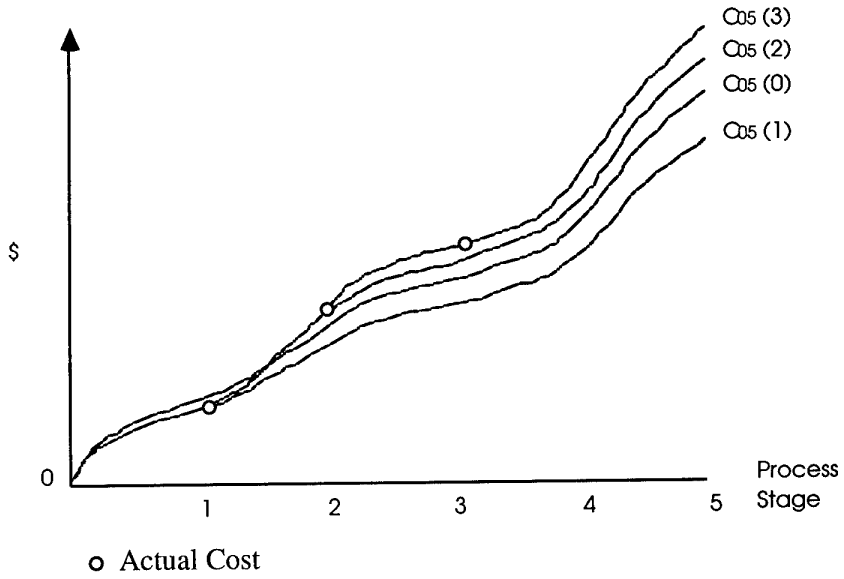

Figure 5: Refining Cost Estimates

The authors have determined a simple mechanism for constructing new estimates from the actual Costs C_{ij} for each stage :

$$C_{05}(0) = \text{Initial Estimate}$$

$$C_{05}(1) = \frac{C_{01}(1) - D_{01}(1)}{a} + D_{01}(1) + D_{15}(1)$$

$$C_{05}(2) = \frac{C_{02}(2) - D_{02}(2)}{a+b} + D_{02}(2) + D_{25}(2)$$

where

- $C_{ij}(k)$ is the Cost for the completion of phase j including only Costs from the completion of phase i (i<j) and determined at the end of phase k. These are estimates where $k < j$ and they are observed values when $k \geq j$.
- D_{ij} is the deviation to the expected Cost for the period between phase i and phase j, which is due to known or anticipated deviations from the standard process. These are estimates where $k < j$ and they are observed values when $k \geq j$.

More complicated mechanisms which detect trends in these changes and allow the integration of new information, such as Function Point Counts are currently being developed. These will be documented in [Kingston and Burke, 1995].

3. Data Requirements

The next stage in the Software Cost Prediction component of the iMAPS task will involve the statistical analysis of data to investigate the conjectures of Sections 2.3 to 2.5 to determine if there are any correlations between the input factors and Development Effort, and therefore Cost.

This Section discusses the data requirements for the iMAPS Software Cost Prediction research and focuses on the requirements for the initial investigation of the conjectures as proposed in Sections 2.3 to 2.6. Section 3.1 identifies the collections of input factors which may be separately analysed and the data required for this analysis. Section 3.2 discusses how organisations participating in this data collection exercise will be affected and Section 3.3 discusses alternative mechanisms which may be used to collect the data.

3.1 Data Categories

The conjectures to be investigated were described in Section 2. However, these correspond to a large set of factors which would require a substantial data set for its analysis. These factors can be grouped according to the conjectures they help explore.

The first group contains information available early in the Software Development, the second group contains information generated through the Software Development and the third group contains miscellaneous information. It is likely that the data collection mechanism (see Section 3.3) will be different for each group.

Each group contains a subgroup of information which is essential to the current conjecture, and additional subgroups which provide more detailed information which allow variants of the conjectures to be explored. Each subgroup corresponds to a section of the Questionnaire in Appendix 1 and the labels of the subgroups indicate the appropriate section of the Questionnaire.

Background information on the project (Part B of the Questionnaire), and the organisation (Part A of the Questionnaire) which conducted the project, is desirable. This allows duplicated projects to be identified and additional contact to be made. It could also be used to obtain information on the definitions used by different organisations for pre-existing measures, such as Lines-Of-Code and Function Points.

GROUP 1

Part C: Cost Slicing Information

This includes the minimum information needed to investigate the Capacity, Difficulty and Cost Slicing conjectures.

Part D: Capacity and Difficulty Information
This includes additional information which could be used to modify, tune or enhance the definitions of Capacity and Difficulty measures and their relationship in the Cost Slicing model.

GROUP 2
Part E: Progressive Refinement Information
This subgroup contains the minimum information required to investigate the Progressive Refinement conjecture in its simplest form. That is a process description, and the Effort for each of the phases in the process.

Part F: Alternative Size Measures
This subgroup contains information which can be used to investigate how estimates of Effort based on measures of Size other than Capacity can be used in the Progressive Refinement technique as they become available. In addition, this information can be used in later analysis to compare the refined, conjectured techniques to other techniques for Software Development Cost Prediction.

Part G: Detailed Size Measures
This subgroup contains more detailed Size information which can be used with that in Group 2 Part H to further investigate the concept of Progressive Refinement.

Part H: Detailed Cost Breakdowns
This subgroup contains more detailed Cost information for finer analysis of the Progressive Refinement conjecture. It includes Effort information on the basis of individual modules, as well as process stages (builds or phases).

Part I: Alternative Cost Information
This subgroup contains the information necessary to perform calculations using currently accepted Costing Models. While it will not be used for the current analysis, it can be used with Group 1 Part C, and Group 2 Parts E and F to compare the refined, conjectured techniques with other techniques for Software Development Cost Prediction.

GROUP 3
Part J: Other
This subgroup allows for the identification of information on possible causes of Cost anomalies not covered by the other areas.

3.2 Considerations for Participants

The data obtained through the collection activities outlined in this document is intended for use in the development and investigation of Software Cost Prediction models. It will not be used for the evaluation of organisations or the individual projects from which data is collected. The data will not be used for the purpose of evaluating the processes used by the organisations. No information on the quality of the final products is being requested and participating organisations are not expected to supply the source code for their projects.

Benefits

The participating organisations are assured of early access to the results of the iMAPS Software Cost Prediction research. This has the potential to improve software cost prediction and the risk management of software projects. In addition, organisations which initiate a data collection program which captures all the information in the questionnaire in Appendix A will be able to customise the model to their local environment. Such customisation has already been shown to improve the precision and accuracy of Software Costing models.

In addition, the organisations may benefit from tools, such as SEE-Ada, which may be introduced to support the data collection process. SEE-Ada [Vernik et al, 1991] is a Software visualisation tool which allows metrics information to be overlaid on a base representation of the system structure. It provides facilities for incorporating information from other tools and directly from the user. In addition to its role as a data collection tool, SEE-Ada could be used to display more information about participants' software, including quality aspects, than is necessary for Software Cost Prediction. This information would not be requested for the iMAPS Software Cost Prediction research.

Effort

While the questionnaire in Appendix A may appear long, it is anticipated that where data already exists in electronic form it could be easily extracted. It appears likely that most of the information requested for Group 2 (See also Section 3.1 and Section 3.3) will either be available electronically (or be able to be generated using automated tools), or not be available at all. Therefore it is expected that only the questions for Group 1 will need to be answered manually. It would be useful, although not essential, for the questions for Group 1 to be answered by the Developers, Clients and the Users of the applications. It is anticipated that this will only take about one hour per application, per person. Participating organisations will be given more detailed information on the Effort required after trials have been undertaken.

Confidentiality

The identity of organisations from which data is collected will be treated as Commercial-In-Confidence. However, it is desirable for the iMAPS task that permission is given to publish the aggregated data in a wide forum. Part of this work is also contributing towards Gina Kingston's PhD studies and it is a requirement that data used in these studies be made available to the University of New South Wales. This data would still remain confidential. Under all circumstances the source of the data would not be divulged. Confidentiality requirements in addition to these would need to be noted. The contribution of participating organisations will be recognised in all documentation. However, only those wishing to be identified will be named.

Flexibility

The current data collection is aimed at initial exploratory investigations of the conjectures given in Section 2. While the nature of future investigations has been foreshadowed, and an attempt has been made to include future requirements in this investigation, it is

possible that additional data will be required for these analyses. It is hoped that the organisations approached during this data collection exercise will be willing to provide additional information, should the need arise. To reduce the impact of such changes in data requirements on the sources, updates will request only the additional information required. The initial data collection will allow for free-form information to be collected. Where the sources have supplied additional information this information will not be requested from them in any updates.

The Software Cost Prediction techniques which arise from this research will be of most benefit to those organisations which, should the need arise, supply additional information in requested updates.

3.3 Collection Mechanisms

It is hoped that wherever possible the data collection can be automated. It is unlikely that full automation will be possible through general purpose tools. However tools, such as spreadsheets, which have been customised for a particular organisation should serve the purpose. It is likely that this approach will work for the data intensive groups (Parts E, G and H), for information on code-based measures of Size (sections of Parts F and G), and possibly for Part I. Collectively, these Parts form the second Group (Section 3.1) of information collected. It may also be possible to automate the collection of information for Groups 1 and 3 where large databases on past projects, which contain the required information, exist. However, it is most likely that the information for Groups 1 and 3 will be collected through interviews.

Accuracy

Because of the nature of this investigation, the researchers require an indication of any potential anomalies in the information supplied. Without this information the results of the research could be invalid. It would be useful to be able to approach the organisations supplying data to determine the nature of possible errors in the supplied information and means of correcting them.

For example, Effort information is often recorded inaccurately: overtime may not be recorded or Effort may be attributed to the wrong task. A related concern is the extent to which information from different organisations reflects the same attributes. For example, there are various definitions of Function Points, and Effort information can be restricted to certain types of people (managers or users may or may not be included) or it may be restricted to certain phases (post-design or post-requirements). It would be useful to be able to investigate these, and additional concerns which arise during the data analysis, through interaction with the sources after the initial data collection.

Collection Mechanisms

The different nature of the data in the various groups given in Section 3.1 indicates that more than one data collection mechanism is likely to be employed. Possible mechanisms include tools like SEE-Ada [Vernik et al, 1991] for supporting Lines of Code measurements and possibly other information for projects where data is collected through the development of the project. It may be possible to extract Effort information

on past projects from existing databases or spreadsheets. Much of the other information will have to be entered by hand, however electronic mechanisms, such as spreadsheets or email may be possible.

The data collection tools MERMAID Mark 1P, M-Base DCSS and Metricate were investigated but appear to offer little benefit for data collection over common spreadsheets.

The annotated questionnaire provided in Appendix A is a preliminary paper-based mechanism. It indicates the types of data required for Groups B and A, and provides detailed questions for Groups A. Guidelines for the questionnaire, which include descriptions of the information requested and definitions of some of the terms used in the questionnaire, are also given in Appendix A. The Questionnaire and Guidelines will be refined through two case studies into their effectiveness.

Amount of Data Required

This investigation requires data to be collected from a large number of projects so that models developed will not be biased towards particular projects, organisations or applications. It requires information from multiple projects developed by the same organisation under the same Development Environment as well as information of projects by different organisations so that models can be developed within and across organisations. Information on individual projects from different organisations will also be useful. It is hoped that data on at least 40 applications will be available for these initial investigations. This number was derived by considering:

- The need for about 10 applications per organisation to develop models within organisations to support the second phase of the approach.
- The desire for at least two application domains to be considered.
- The desire for data from at least two organisations to be available for each application domain.

It should be noted that this data will probably be sufficient for statistically significant results to be obtained if all of the collected information is to be included in the models. It is likely that more data will be required to evaluate the refined models. However the amount of data required will depend on the nature of the refined models and the number of variables they contain. As the initial analyses will only be exploratory it is not considered necessary to obtain sufficient data for a statistical analysis of the more complicated models.

4. Analysis

The analyses of the Software Costing data collected for this research will be broken into two phases.

The first analysis is to identify the strengths and weaknesses of the approach to Software Cost Prediction conjectured in Sections 2.3 to 2.6. This should be undertaken as soon as the initial model is sufficiently well developed and the results should be used to further develop the model. This analysis is likely to be a statistical analysis of the ability of the approach to explain the actual observed Cost. This document is primarily concerned with the data required to undertake this initial analysis.

The second analysis would evaluate the approach and allow comparisons to be made to other approaches. It would be undertaken when the approach has been refined after the first analysis. It is likely to be a statistical analysis of

a) The ability of the approach to explain the observed Costs.
b) The ability of the approach to predict Cost at different points in the Development.
c) The ability of other approaches to predict Cost at different points in the Development.
d) The ability of other approaches to explain the observed Costs.
e) Comparing the results from a) and b) with c) and d) respectively.

The first analysis will consist of ad-hoc analyses which will be determined once the data is available and pre-determined analyses. Both of these analyses will use standard statistical techniques such as linear regression [Moore and McCabe, 1987; Venables and Ripley, 1994]. An outline of these pre-determined analyses is given in Section 4.1

4.1 Exploratory Analyses

The analyses in this section are discussed according the phase of the model they are associated with. The two phases are labelled as the Initial Stage and the Refinement Stage.

Initial Stage

This is the phase where an Effort Prediction is determined using the Slicing model from Capacity and Difficulty measures. The later components in the analysis could identify weaknesses in the initial components requiring several iterations of the early components. The components of this analysis are to:

a) **Investigate how Capacity should be determined from its components.** Initially this would be performed intuitively using a subjective measure of the project's Capacity and the information on the conjectured components. If an intuitively appealing model was found, it would be tested statistically. However, as the Capacity measure would have been subjectively determined, the model would not necessarily be rejected if the relationship was not statistically valid. Under these circumstances any differences would be noted with a view to refining the Capacity concept and its components. The data would also be analysed to determine if any of the components of Capacity were

highly correlated, which would indicate redundancy in the components. This analysis would be undertaken with the Group 1, Part C data.

b) **Investigate how well Capacity correlates with Effort.** This would be a statistical analysis using both the subjective Capacity measure as well as that calculated using the relationship derived in Part a). The components of Capacity would also be compared with Effort to determine if another combination of the components would provide a better correlation with Effort.

c) **Investigate if Difficulty improves the correlation with Effort.** The first step in this component would be to determine potential Process, Product and Resource (PPR) measures. This would involve comparing the subjective questions on Process, Product and Resource to the objective questions. Once this had been determined, Effort models would be developed for each possible PPR combination. Outlying points would be identified to determine if they had any answers to the PPR questions which distinguished them from other projects. This would then be used to refine the method of determining PPR measures. The Effort correlations would then be adjusted accordingly and the PPR combinations assigned a Difficulty value according to the relationship. Once difficulty values had been assigned to each PPR component, a combined analysis of Effort against Difficulty and Capacity could be investigated. If this offered no improvement over the Capacity correlation then further investigation would be required. Alternative combinations of the PPR measures and their components would also be investigated.

d) **Investigate if Capacity can be improved using additional information.** This would use the additional Capacity information in Group 1 Part D to determine if the Capacity measure could be improved by including additional information. This would involve analyses similar to parts a)-c) and would initially use the Difficulty models determined in part c) to determine if a better Effort Explanation model could be developed.

e) **Investigate if Difficulty can be improved using additional information.** This would involve analyses similar to part c) and would initially use the Effort Explanation model developed in part d).

f) **Investigate the Cost Quanta concept.** This would be an intuitive investigation of Cost Quanta using the models developed in part e).

g) **Determine if the desired properties of Capacity, Difficulty and Cost Slicing** are still present and identify any limitations in this area.

Refinement Stage

This is the phase where the model would be progressively refined based on new information as it becomes available.

a) **Investigate refinement based on phased Costs.** The first step in this component would be to determine if there is a ratio between the phases for each process used. Where the phases of the process are based on builds, this may need to depend on the modules in the builds. The average, and standard deviation of the ratios would be determined. At some stage any projects which did not conform would be identified, and the reason why determined where possible - eg one phase out, all out, different language used etc. Analyses would consider all projects and the consequences of leaving out outliers and/or the project of interest.

b) **Investigate refinement based on additional Size measures.**
Two possible approaches exist, the first would be to develop alternative models for software Effort for each of the Size measures obtained and to consider the use of an Effort Prediction (an Effort value and the Prediction Interval around it) from any models and to determine how they could be combined. This would effectively determine how to combine different models for Effort. The second approach would be to consider each Size estimate separately and consider how it may be directly incorporated into the model. The statistical implications of these options would need to be considered before the analysis approach could be confirmed. However, both have different properties. The first would allow alternative Size measures to be used, without redeveloping the model. The second would allow consideration of properties of specific Size measures.

c) **Investigate refinement based on phased Costs and additional Size measures.**
This analysis would be similar to that of b), but would combine the information obtained in a). Differences between the Effort Predictions available at different stages of the development would be considered. If a) or b) are not successfully completed, this would not be attempted.

d) **Investigate refinement based on module Costs.** This would be the same as a) but at a lower level of granularity. If part a) was not successful, it is unlikely that this would be successful either. A suitable measure of Size which could be used early in the Development would eventually be necessary to use this work, but for this analysis which is concerned with Effort Explanation, Lines Of Code or any other available measures would be used.

e) **Investigate refinement based on module Sizes.** This would look at refinement based on differences between the anticipated and actual Sizes on the modules and would be closely related to d). This would require a method for Estimating the module Sizes early on, but initial work might rely on an "average size" based on other projects performed by the organisation.

5. Summing Up

Software is an increasingly important element in modern Defence systems. A large proportion of the Australian Defence Organisation's budget is currently committed to the procurement and maintenance of software based systems. Examples which highlight this include the Submarine, ANZAC Frigate, Jindalee and Nulka projects.

The process of estimating, monitoring and controlling Software Costs in Projects is ad-hoc, which can result in project Costs exceeding the original budget or systems being delivered with sub-optimal functionality. These shortfalls are generally met by both the ADO and the software contractors. Current approaches to Software Costing tend to focus on Business, or Transaction-based Applications and have limited applicability to Defence systems. Furthermore, no concerted effort has been made to coordinate the collection and analysis of Software Cost data from Defence Projects or to provide guidance to the Projects on best practice in this field.

This document describes the data collection requirements for a systematic approach to Software Costing being developed by the iMAPS Software Cost Prediction team. The approach consists of two phases and has the potential to support decision-making in a reasoned, risk-managed way at all stages of system acquisition. The result of the first phase, Cost Slicing, would be a coarse initial estimate which could be used early in the acquisition process when comparing inputs to Defence Force Capability Options or Capability Proposals. This phase would be followed by an iterative process, Progressive Refinement, which could be used to refine the Cost estimates during the remainder of the acquisition process, including both before and after contract negotiation.

6. Acknowledgments

The authors thank DSP-FDI, Dave Saunders and SP-DI, Kevin Bly for their feedback on early drafts of this document. The authors also thank HSE, Stefan Landherr and HIAP, Peter Calder for their input in related discussions.

7. References

A. J. Albrecht and J. E. Gaffney Jr, 1993 "Chapter 8: Software Function, Source Lines of Code and Development Effort: A Software Science Validation", Software Engineering Metrics, Volume 1, Measures and Validations. Ed: M. Shepperd, McGraw Hill, pp 137-154.

J. G. P. Barnes, 1989 Programming in Ada, Third Edition, Addison-Wesley.

B. W. Boehm, 1984 "Software Engineering Economics", *IEEE Transactions on Software Engineering*, Vol SE-10, No 1, pp 4-20.

M. M. Burke, 1995 *iMAPS Task Plan*, DST 93/949, Issue 2.

M. M. Burke, 1994 *iMAPS General Introduction*, ERL-0826-GD.

F. J. Heemstra, 1992 "Software Cost Estimation", *Information and Software Technology*, Vol 34, No 10, pp 627-639.

D. R. Jeffery and G. Low, 1990 "Calibrating estimation tools for software development" *in Software Engineering Journal,* July, pp 215-221.

C. F. Kemerer, 1991 "Chapter 28: Software Cost Estimation Models", Software Engineer's Reference Book, Ed: J. A. McDermid. Butterworth-Heinemann, 1991.

A. Kiermeier, 1994 *CEED Project: Project Proposal. Software Cost Prediction: A Statistical Approach*, Project Number 94705.

G. Kingston, M. Burke and R. Jeffery, 1995a *iMAPS: A Review of Software Cost Prediction Techniques*. Report in progress.

G. Kingston, M. Burke and R. Jeffery, 1995b *Software Sizing for Effort Estimation.* Report in progress.

G. Kingston, A. Kiermeier and M. Burke, 1995c "On the Statistical Significance of Productivity Factors in Software Development Effort Explanation". To be published in *ACOSM'95*.

G. Kingston, A. Kiermeier and M. Burke, 1995d "On the Statistical Significance of Function Point Technology Factors in Software Development Effort Explanation". In progress.

G. Kingston and M. Burke, 1995 *iMAPS: A New Approach to Software Cost Prediction*. Report in progress.

B. A. Kitchenham, 1992 "Empirical Studies of assumptions that underlie software cost-estimation models" *in Information and Software Technology Journal,* Vol 34, No 4.

J. E. Matson et al., 1994 "Software Development Cost Estimation Using Function Points." *IEEE Transactions on Software Engineering* 20(4): pp 275-286.

D. S. Moore and G. P. McCabe, 1989 Introduction to the Practice of Statistics, W. H. Freeman and Company.

W. N. Venables and B. D. Ripley, 1994 Modern Applied Statistics with S-Plus. New York, Springer-Verlag.

R. J. Vernik et al, 1991 "Automated Support for Assessment of Large Ada Software Systems". *TRI-Ada'91.*

Appendix A: Preliminary Software Costing Data Collection Questionnaire

This questionnaire was designed for the collection of data for the development and refinement of Software Costing techniques developed by the iMAPS team. The results of this research will be available to participants.

Instructions

The chart below shows the relationship between the different Parts of the Questionnaire. A given Part should only be answered if all the Parts above it in the chart are also answered.

One copy of Part A should be completed. This contains an organisation or source number which also appears on the top of all other sheets and will be used to refer to your organisation in all documentation of the research. The remaining information in Part A will only be used to contact you if we require clarification of any of your answers, or if we require additional information for further development of the models.

One copy of Part B should be completed for every project and/or application (Software Configuration Item) for which data is submitted. A number should be assigned to each project and application. These numbers should also appear at the top of all sheets submitted for that project.

The remainder of the Questionnaire is broken into three groups. The first two groups address the two phases of the conjectured approach. While it is preferred that the questions from both groups are answered, the answers to only one of these groups are necessary.

The groups are further broken down into parts and the dependencies between the groups are shown in Figure A-1. Where possible, all the questions of a Part should be completed.

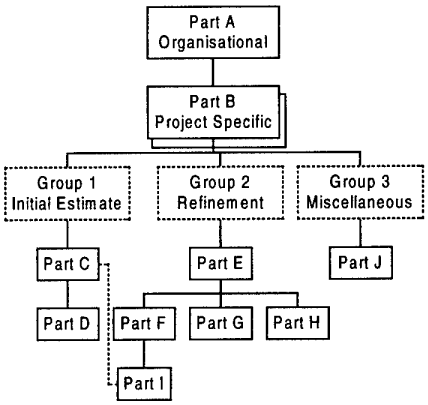

Figure A-1: The Relationship between different sections of the Questionnaire

The questions in Group 1 should be completed as if the project had just commenced. Where a project consists of multiple applications (or Software Configuration Items), information may be given either for the entire project or for each application. It is preferable that the Group 1 questions are answered at both the project and the application levels.

Guidance on how to answer the questions and the terminology used is interwoven with the questions and is given in Helvetica font. The questions always appear on even (left-hand) pages and have the word 'Questionnaire' in the left-hand column. Some guidance also appears on even pages, but without the word 'Questionnaire' in the left-hand column.

It is anticipated that Group 1 with take 1 to 2 hours to complete and that much of the information for Group 2 will already be available electronically.

Approximate answers, with an appropriate error margin, would be appreciated for any questions for which detailed information is not known.

QUESTIONNAIRE CONTENTS

Part A. Organisational Information

1. Organisation Number (Given)

2. Organisation Name
3. Address

4. Country

5. Contact name
6. Phone
7. Fax
8. Email

9. Size of Organisation
   - Total Number of Staff
   - Total Number of Software Development and Related Support Staff
10. Type of Organisation
    - Government - Defence
    - Government - Other
    - Software Development (non-Government)
    - Other Commercial (Specify)
    - Other (Specify)
11. Number of Organisation Sites
12. Parent Organisation (if relevant)

9. The total number of staff should be indicated. Where both part-time and full-time staff are employed, please indicate the number of full-time and of part-time staff separately. Related support staff are all those staff who's main function is to support the software development staff. This includes secretaries, typists and pay-clerks. Where such staff also support other staff, their numbers should be averaged over the number of such staff they support. Staff on help desks, or involved in software maintenance, should not be included.

11. Number of Organisation Sites

    If the organisation is distributed over a number of sites, indicate the number of geographically dispersed sites.

12. Parent Organisation

    If the Organisation is part of a larger Organisation, give the name of the Parent Organisation.

Questionnaire

Part B. Project Information

1. Organisation Number (Given)

2. Project Number
3. Project Title
4. Project Description

5. Year Commenced
6. Year Completed
7. Level of Completion
   - Completed on Time and on Budget
   - Completed
   - In Progress
   - Not completed

8. Team Leader's Name
9. Contact name (if different from 8)
10. Phone
11. Fax
12. Email
13. Address (if different from that of the organisation given in Part A)

14. Number and type of application(s) (Indicate all that apply)
    - Scientific
    - Real-Time
    - Information Processing System
    - Control System
    - Command and Control System
    - Embedded
    - Other (Specify)
15. Purpose of application (s)

2. Project Number

   This is a number which can be used to distinguish information on projects submitted by the same organisation. Where the number of projects for which data is being submitted by an organisation is known, these numbers will be supplied by the iMAPS team. However, in order to allow the questionnaire to be filled in with the minimum disruption, where numbers cannot be pre-allocated the number used may be allocated by the organisation completing the questionnaire, or it may be the number used internally to identify the project. Alternatively , the organisation may request a number from the iMAPS team. (This may be necessary for geographically dispersed organisations).

4. Project Description

   A brief, high-level description of the project should be provided. (The purpose should be described under question 15).

7. Level of Completion

   Indicate if the project was

   FULLY COMPLETED - that is on time, on budget and with the full functionality initially proposed;

   COMPLETED - that is a project was delivered, over time, over budget or with reduced functionality (indicate which);

   IN PROGRESS - that is the project is still undergoing development, or

   NOT COMPLETE - that is the project was terminated before delivery.

8. Team Leader

   The person in charge of the development of the software.

14. Type of application

    Indicate if the application is scientific, real-time, information system etc. Where the project consists of more than one application, indicate the number of applications of each type.

15. Purpose of application(s)

    Briefly describe the purpose of the project and each application in the project. Associate a unique number with each application. (The combination of organisation, project and application numbers while be used to uniquely identify applications within the data set).

The next three questions are provided for cross-referencing questionnaires completed by different organisations for the same project.

16. Client
    a) Organisation Name
    b) Address
    c) Business Area
    d) Contact Name
    e) Contact Phone

17. Users
    a) Organisation Name
    b) Address
    c) Business Area of the Prime Developer
    d) Contact Name
    e) Contact Phone

18. Developer and Sub-Contractors
    a) Organisation Name
    b) Address
    c) Business Area
    d) Contact Name
    e) Contact Phone

19. Staffing
    a) Number of Project Software Development Staff (See guidelines opposite)

Staff	**Activity**																		**Total**	
	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M
Designers																				
Programmers																				
Testers																				
QA Personnel																				
Metrics Personnel																				
Software Engineers																				
Project Managers																				
Total																				

    b) Number of Project Support Staff
    c) Number of Project Management Staff

16. Client

The client is the organisation or organisational representative who is paying for the software.

17. Users

The end-users are the organisation or organisational section who will use the software.

18. Developer

The organisation who developed the software system.

19. The staff column indicates categories of staff. Several lines are left blank so that additional categories can be entered. The activity columns are for the activities performed at various stages of the development. For example, the following activities are used in the example below.

Requirements
Design
Coding
Component Testing
Integration Testing
DOcumentation.

Please supply staff numbers for the entire project and where possible categorise them as indicated in the table. If this information has not been collected in detail, then please supply the Total values. The columns indicate the Average, or Maximum number of staff of each category for each activity. Where possible, please supply both averages and maximums.

Note that the Total Row contains the sum of the columns, and that people may participate in activities other than those dictated by their roles. Averages less than 1 have not been given in the following example, and it is assumed that some averages could not be determined.

Staff	**Activity**																				**Total**	
	R		*D*		*C*		*CT*		*IT*		*DO*											
	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M	A	M
Designers			2	3							1											3
Programmers					3	5	3	5														5
Testers									3	3												3
QA Personnel		1		1		1		1		1												1
Metrics Personnel		1		1		1		1		1												1
Software Engineers	1	1																				1
Project Managers	1	1	1	1	1	1	1	1	1	1											1	1
Analyst/Programmer			1	1	1	1																1
Technical Writer											1	1										1
Total	2	4	4	7	5	9	4	8	4	6	1	1									-	17

Questionnaire

Part C. Cost Slicing Information

(Group 1 Information as described in Section 3.1)

1. Organisation Number (Given)
2. Project Number
3. Application Number (if applicable)

4. What is the Capacity level of the project or application? (See the table opposite)

A
B
C
D
E
F
G
H
I
J
K

3. If this Part is being completed for the entire project then leave this blank. Otherwise identify the application with an application number. Use this number consistently within the questionnaire.
4. The capacity level of the project (or application) should be chosen using the following table. Choose the Standard which is closest to your program in the amount of functionality it provides and use the Usual Characteristics to check your choice. Note that the scale is not linear.

Level	Usual Characteristics				Standards
	Input	Output	Interfaces	Processing	
A	None	Basic, Fixed	1 Output Device	None	Write 5 to output Write "Hello World"
B	Basic, Single type	Basic	1 Output 1 Input	None	Echo a number Echo a message
C	Basic, Single type	Basic	1 Output 1 Input	Basic, Single Unit	Sum Hello 'X'
D	Basic	Basic	1 Input 1 Output	Basic, Many Units	Calculator Line Editor
E	Basic	Detailed	1 Input 1 Output	Basic, Many Units	Graphical Calculator Graphical Editor
F	Basic May be higher	Detailed	1 Input 1 Output May be higher	Medium - Detailed	Scientific Calculator Word Processor Distributed Messages
G	Several types	Detailed	Several I/O	Medium - Detailed	Navigation Aid Distributed Comms
H	Several types May be higher	Detailed	Several I/O May be higher	Medium - Detailed May be higher	Satellite Navigation Satellite - Ground Comms
I	Many types	Detailed	Many I/O Restricted Types	Detailed	Satellite Motion System Satellite Communications
J	Many types	Detailed	Many I/O Diverse Types	Detailed	Satellite Control System
K	Detailed	Detailed	Many	Detailed - extensive	"Star Wars"

5. How many hardware interfaces does the project require?
6. How many software interfaces does the project require?

7. What is the processing level of the project?

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S

5. How many hardware interfaces does the project require?
   A hardware interface is a method which can be used to communicate with a (type of) device. Input, output and sophisticated error streams are each counted separately.
   - Different communications mechanisms for the same (type of) device have one interface each. For example, a monitor may have a text-based and a windows-based interface giving a count of two hardware interfaces.
   - Multiple devices of the same type count only once, provided they are communicated with in the same way. For example, two keyboards count only once.
   - Similar devices, eg two types of monitors, count only once, provided they have similar operating protocols. Conversely two types of CPU would normally count twice.
   - If files are read or written by the program than the interface(s) to the disks should be included. Operations performed by other software should not be included.
6. How many software interfaces does the project require?
   Two software interfaces are required for each pair of software applications which communicate with each other. Only one interface is required if the communication is one-way. For example, if the application uses a (separate) database application to store and retrieve data then two interfaces should be counted. If the database is only used to store information (and there is no error checking) only one interface should be counted.
7. The processing level of the project (application) should be chosen using these tables:

Level	Usual Characteristics		
	Operations	Objects	Areas
A	No processing		
B	Basic	Few	1
C	Basic	Few	2-6
D	Basic	Few	7-8
E	Basic	Similar	1
F	Basic	Similar	2-6
G	Basic	Similar	7-8
H	Basic	Diverse	1
I	Basic	Diverse	2-6
J	Basic	Diverse	7-8
K	Complex	Few	1
L	Complex	Few	2-6
M	Complex	Few	7-8
N	Complex	Similar	1
O	Complex	Similar	2-6
P	Complex	Similar	7-8
Q	Complex	Diverse	1
R	Complex	Diverse	2-6
S	Complex	Diverse	7-8

Basic operations are those which consist of a few simple steps. Complex operations, consist of multiple steps and may rely on other such operations.

Acts involve the movement of external system components.

Areas of Processing
Calculation
Manipulation
Obtain/Retrieve
Store
Transfer
Present
Monitor
Act

8. What is the impact rating of the development environment and product constraints on the project or application?

1
2
3
4
5
6
7
8
9
10

8. Rate the impact of the work environment and the product constraints on the project (or application) (1-10)

This is a subjective measure of the impact of the environment (which may be dispersed and involve many development processes) and the constraints (such as memory and timing constraints). It may be used when considering how to determine a measure which captures the impact of the work environment in an objective manner. It should indicate the rating which would have been given PRIOR to project (application) development. Intermediate values may be used where they can be justified. Those shown in grey are not likely to be given for projects where development commenced.

Rating	Meaning The development environment and constraints on the project (application) ...
1	were likely to *enable* the product to be produced for *minimal Effort.*
2	were likely to *enable* the product to be produced for *a relatively small Effort.*
3	were likely to *enable* the product to be produced for *a modest amount of Effort.*
4	were likely to *enable* the product to be produced for *a reasonable amount of Effort.*
5	were likely to *(slightly) increase* the Effort required for the project.
6	were likely to *increase* the Effort required for the project to a *relatively high level.*
7	were likely to *increase* the Effort required for the project *and the likelihood of failure.*
8	The development environment and constraints on the project (application) gave the project a high likelihood of failure.
9	The project (application) appeared to be nearly impossible to complete given the development environment and constraints on the project. (It may still not have been completed if, for example, management intervened.)
10	The project (application) appeared to be impossible to complete given the development environment and constraints on the project.

9. a) Has a well-defined development method being used?
   b) Has it been customised for local use?
   c) Are these changes well-defined?
10. Is the development method stringent?
11. What programming language(s) is (are) being used?

Ada83	C	Fortran 11
Ada95	C++	Fortran 66
APL	Objective C	Fortran 77
Pascal	Eiffel	PL/1
Modula-2	Smalltalk	PL/S
Prolog	Cobol 74	Algol 68
Lisp	Cobol 85	Algol W
Miranda	Jovial	ANSI Basic
Forth	Mumps	Visual Basic
SQL	Chill	Pearl
AWK		

Scripting Languages (Specify)

Other (Specify)

12. What level is the language(s)? (Indicate the number of languages used at each level.)

1GL	Machine Language
2GL	Assembly Language
3GL	High Order Language
4GL	Fourth Generation Language (eg Database Language)
5GL	Fifth Generation Language (eg Spreadsheet or Graphical Language)

13. Is software reuse being attempted in the development of this project (application)?

Function Reuse
Object Reuse
Sub-system Reuse
System Reuse

14. Are there many constraints on the software? (Y/N)
15. Are the constraints conflicting? (Y/N)

16. Are there adequate resources available? (Y/N)
17. Is the project (application) novel for the development team? (Y/N)
18. Are suitable people available for all necessary software development activities? (Y/N)
19. Is the hardware available and mature? (Y/N)

9. Development method.
   A method is well-defined if it is documented or there are procedures to automate it.
10. Is the development method stringent?
    That is, does the process include many checking steps, such as requirements tracking or Independent Validation and Verification.
11. What programming language(s) is (are) being used?
    Where more than one language is being used, all languages should be listed and their approximate percentage usage. It should be noted how these percentages were determined (ie based on LOC, functionality etc).

13. Is software reuse being attempted?
    Reuse should be counted whether or not the components require modification.

14. Are there many constraints on the software?
    A list of possible constraints in given in Part D, Q10.
15. Are the constraints conflicting?
    For example, are time/space trade-offs or safety versus user friendliness trade-offs required.

16. Are there adequate resources available?
    This is a subjective question to be used to determine how well the questions below map to the Resource concept. Time, Money, Personnel and Computing resources should all be considered.
17. Is the project (application) novel?
    Is the project (application) a new application type for the development team?
18. Are suitable people available?
    That is, does the development team contain people with appropriate skills (through training or past experience) for each of the activities in the development phase, and are they going to be available at the appropriate stage in the development.
19. Is the hardware available and mature?
    If the hardware is under development, or the hardware is being refined or it is the first time the hardware is being used, answer no. If the target hardware will not be available when the coding commences or access to the hardware will be limited, answer no. Otherwise answer yes.

Questionnaire

20. a) Are support tools being used? (Y/N)
    b) Are they mature? (Y/N)
21. Was there pressure to complete the project (application) within a restricted time or budget? (Y/N)
22. a) Is there a reuse library? (Y/N)
    b) Is there a project library? (Y/N)

23. What was the total Effort required to develop the project (application)?

Development Effort	( )
Support Effort	
Overtime	
Sub-Contractors	

24. Phase at which Effort recording was commenced
25. Phase at which Effort recording was completed

20. Support tools

Support tools include project planning and tracking aids, configuration management tools, integrated development environments and automatic test generators.

Mature tools are stable and relatively error-free.

21. Was there pressure to complete the project within a restricted time or budget?

A 'yes' answer to this question implies a fixed time or budget (or at least hard limits).

22. a) Is there a reuse library?

That is, is there a repository of reusable components which the development team will have access to.

b) Is there a project library?

A project library should include design information and documentation of major decisions as well as the source code for the system being developed. It may exist for the project as a whole or for individual applications.

23. What was the total Effort (including overheads) required to develop the project (application)?

In the brackets () indicate the units in which Effort (time spent by staff) was recorded. The preferred units are staff-hours. If alternative units are used, please indicate the conversion rate to obtain staff-hours.

Information should be recorded for the entire project, or the application, for which this Part is being completed (see Question 3). The Development Effort (in the units stated) is the Effort of the project staff from the organisation completing the questionnaire. The Support Effort is the Effort of support and managerial staff whose time is not booked to a particular task.

The Overtime label should be used to record if all time (including Effort outside normal working hours, or Overtime) worked by the organisation's project staff was included in the recorded Effort. If Overtime was not included, this label should be used to indicate either the actual Overtime Effort or the estimated percentage of Overtime.

The Subcontractors label should be used to indicate the Effort from sub-contractors and the % Overtime assumed if this is not included in the Effort.

If this questionnaire is being completed by the client or user indicate the Effort spent by your organisation separately.

24. Phase at which Effort recording was commenced

In particular, indicate if Effort recording was commenced at project conception, at the start of the project specification, after contract negotiations, or after delivery.

Questionnaire

Part D. Capacity and Difficulty Information

(Group 1 Information as described in Section 3.1)

1. Organisation Number (Given)
2. Project Number
3. Application Number (if applicable)

4. Mark the hardware interfaces required by the project (application):

Monitor - Graphics	Keyboard
Monitor - Text	Mouse
Monitor - Windows	
Another CPU - Output	Another CPU - Input
	Another CPU - Error Stream
Modem - Output	Modem - Input
Printer - Text	
Printer - Graphics	
Plotter	Scanner
Speaker	Microphone
Hard Drive - Output	Hard Drive - Input
Floppy Drive - Output, 3 1/2in	Floppy Drive - Input 3 1/2in
Floppy Drive - Output, 5 1/4in	Floppy Drive - Input 5 1/4in
	CD-Rom

Other - list all devices and the type of interface

eg. Faxes - Error Handling, Outgoing and Incoming Fax Interfaces

Other Output Devices	Other Sensors

5. Mark the software interfaces required by the project (application). Where interfaces to more than one tool of the same type are required, or the communication is two-way, indicate the number of interfaces required.

Database
Operating System
Spreadsheet
Geographical Information System
Word processor
Email
Graphics Tool
Network Software (List all types)
Other (List)

3. If this Part is being completed for the entire project then leave this blank. Otherwise identify the application with an application number. Use this number consistently within the questionnaire.
4. Mark the hardware interfaces required by the project (application):

   Other interfaces are combinations of the device, and the mechanism used to interface to it. Devices which can be used as inputs and outputs count at least twice. Devices which have complex error reporting mechanisms have an extra count. Devices which may conceptually be controlled in more than one way (for example monitors with text, graphics and windows interfaces) have multiple counts.

5. Mark the software interfaces required by the project (application). Where interfaces to more than one tool of the same type are required, indicate the number of interfaces required. Indicate separately if the communication is one-way or two-way.

6. Complete the following tables. The tables opposite describe the information required for the first table. The second table should be used to describe the required characterists of the software.

OPERATIONS:

	(Maximum) Operation Level	Rank of Operations (of Max Level)	Rank of Objects (of Max Level)	Rank of Domains (of Max Level)
Calculation				
Manipulation				
Obtain/Retrieve				
Store				
Transfer				
Present				
Monitor				
Act				

CHARACTERISTICS:

C	M	O/R	S	T	Pr	Mo	A	**Handles**
								Boundary conditions
								Over/underflow
	na	na	na	na		na	na	Round-off Errors
na	na					na	na	Device Errors
								High or varying accuracy
		na	na	na				High or varying precision (granularity)
								Partial Solutions
			na					Uncertain information
								Incomplete information
		na			na			Back-tracking (eg undo)
								Special cases (eg near singular matrices)
								Other Errors - Correction
								Other Errors - Fail safe

6. Operations

Operation Level	Types of Operations
0	Not used
1	Single, isolated, operations
2	Simple combinations of operations performed on simple types
3	Operations performed on simple objects
4	Operations performed on complex objects
5	Operations performed on collections of very similar objects
6	Operations performed on collections of objects

The preceding table provides a generic description of the levels of operations. The maximum level for a particular type of operation is the highest level at which operations exist for that type of operation. The different types of operations, and the standard objects they operate on are given in the following table. This should be used with the generic descriptions in the previous table to determine the maximum level for each type of operation.

Operation Level	1 & 2	3	4	5	6
Calculation	Integers, Reals	Vectors	Matrices etc (Fixed Size & Dimension)	Matrices etc (General, Regular)	Other
Manipulation **O**btain/**R**etrieve **S**tore **T**ransfer **P**resent **Mo**nitor	Single Values	Linear Objects	Non-linear Objects (Common Components)	Non-linear Objects (General Components)	Non-linear Objects (Varied Components)
Act ("Movement")	Switches	Fixed moves	"Linear"	2-D	3-D

The following table describes how to rank the number of operations, objects and (application) domains the project, or application (see Question 3) crosses. An object is any type of entity which may be manipulated by the system. For example, customers and accounts are entities of banking finance systems. A domain is a unified collection of specialised information including facts and procedures. For example in a banking system, loans, evaluations, credit checking, and savings accounts would require specialised procedures and therefore would be separate domains. Where the exact number is know, it should be placed in brackets after the rating.

Rank	Operations (Objects, Domains)
0	None
1	Single operation/object/domain
2	Few operations/objects/domains (2-5)
3	Low number of operations/objects/domains (6-10)
4	Medium number of operations/objects/domains (10-15)
5	High number of operations/objects/domains (15-20)
6	Large number of operations/objects/domains (20-30)
7	Complete library of operations/objects/domains (specific)
8	Complete library of operations/objects/domains (generic)

7. Mark the following criteria which apply to your software development process.

It is defined by:

- Documented (internal source)
- Documented (external source)
- Supported by tools
- Common practices passed on verbally
- Other (Describe)

The process (standard) is:

- Preferred (developers are encouraged to follow it)
- Used (consistently by developers)
- Checked for major discretions (by developers management)
- Justification must be given for deviating from it
- Enforced (by management)

- Reviewed and Refined (regularly)
- Other (List)

8. Mark the following activities which are included in <u>your</u> development process.

- Defect Tracking
- Metrics Tracking
- Metrics Collection
- Independent Quality Assurance

- Requirements Tracking
- Requirements Elicitation
- Requirements Change Management
- Prototyping

- Risk Management
- Independent Validation and Verification
- Subcontracting
- (Formal) Training

- Other (List)

8. Mark the following activities which are included in your development process.

Defect Tracking

Allows defects to be tracked as to their origin and resolution. It can form a basis for estimating the remaining defects in a system.

Metrics Tracking

This includes schedule and budget tracking against product size or development activities. Its objective is to determine how the development of the product is progressing. It also includes tracking software quality throughout the development.

Metrics Collection

Manual or automated methods for collecting software metrics.

Independent Quality Assurance

An independent team, not necessarily from a commercially independent organisation is used to check the quality of the product and determine limitations and their potential causes from the development process.

Requirements Tracking

Allows requirements to be tracked from the requirements document to the source code.

Requirements Elicitation

Methods used for determining and refining the user requirements.

Requirements Change Management

A mechanism for handling changes to requirements. This can vary from not allowing changes to requirements, to producing new cost estimates for each requirement change and updating any contracts etc accordingly.

Prototyping

A development phase where a throw-away product is developed to help determine the user requirements or the feasibility of the more difficult or novel parts of the software development.

Independent Validation and Verification

A team from a commercially independent organisation which are used to check the code, often for safety and security issues.

Risk Management

A mechanism for determining the risks associated with the project (application) and means by which they can be eliminated, reduced, or detected early should they arise. It is not just a mechanism for dealing with problems after they arise.

Subcontracting

Are subcontractors used in the project development.

(Formal) Training

Is there a formal training process in place which identifies the training needs of the staff, including training in new tools or techniques used in the application/project or updating skills or which ensures that all staff have a base level of competency in their required skills. (This does not mean that all programming staff have a degree in Computer Science since different degrees focus on different aspects of computing).

Other

It is not intended that all activities in the development process be listed here. Those that relate to design/code/test type activities should not be listed here. This section is meant to contain those activities which relate to the management of the development, however other activities can be included if it is felt they have a significant impact on the way the software is developed (and/or if Part E is not answered).

9. Indicate the following:

Management Overhead

Ratio of developers to managers

Percentage time developers spend on management activities

Documentation Requirements

DOD-STD-2167A (Y/N)

MIL-STD-498 (Y/N)

Users Manuals (Number of styles)

Training Documents (Number of styles)

Maintenance Guides (Number of styles)

Other

Management Approval Status (Mark one only)

Pushing the project

Top priority

Fully supported

Currently supported

Phased support

Interim Arrangement

Experimental

Other (Explain)

Project/Application Security Measures (Mark those which apply)

Restricted access to the development site(s)

Restricted knowledge of components and purpose (Need to know)

Security Personnel required for the development site

Security alarms required for the development site

Staffing Restrictions for developers, or all staff (specify)

Other (Explain)

Attempted Reuse Level (% Reused code)

Coding Standards (Internal, External or Not used)

Other Standards Used

POSIX (Y/N)

X

Language Standards (List)

Other

9. Management Overhead

Managers are considered to be full-time over the life of the project. Where this is not appropriate, please indicate full-time and part-time numbers, as well as the minimum and maximum number of managers used.

Management overheads include effort to supervise other staff and mangagement of the project at a high level and does not include activities such as configuration management or quality control.

Meetings with the clients for requirements elicitation do not count as management, whereas meetings with the clients for contract negotiations etc do count.

Documentation Requirements

This should include requirements to follow standards such as 2167A as well as the requirements for manuals (different user's guides, training manuals, etc).

Management Approval Status

This should indicate whether the management is:

a) PUSHING the project (application) to the extent that it may be difficult to control their expectations of the project, particularly if problems arise
b) the project (application) is their top priority, but have realistic expectations
c) the project (application) is FULLY SUPPORTED
d) the project (application) is CURRENTLY SUPPORTED, but may be withdrawn
e) the first phase of the project (application) is supported, but later phases are subject to review (PHASED SUPPORT)
f) the project (application) is considered an INTERIM arrangement until something more suitable can be arranged
g) the management consider the project (application) as EXPERIMENTAL and may think that it is likely to fail
h) OTHER

Project/Application Security Measures

The project (application) itself, rather than its end use, requires security measures to be in place. This may be because it is commercially sensitive or requires knowledge of classified material.

Attempted Reuse Level

While the actual amount of reused code cannot be known prior to the development, historical information and expected improvements, are often used to predict the amount of reused code, as a percentage of the size (normally in Lines of Code) of the final source code

Other Standards Used

Indicate any other standards used, such as testing standards, user-interface standards etc. Include both development and coding standards.

10. Mark the constraints which apply to the project's (applications) software. Indicate if the constraints are general or have acceptance criteria and if the will be checked - internally, by the client or user, by an external organisation or not at all.

    High Useability
    Real-Time
    High Portability
    High Throughput
    Multiple (Specified) Hosts
    Limited CPU time
    High Maintainability
    Interactive
    High Reliability
    Limited Memory
    Security
    Limited I/O Capacity
    Safety
    Limited Storage
    Other (List)

11. Mark the resources available to the project (application development).

    Reuse Librarian
    Library of Reusable Components
    Support Staff
    Mature Hardware
    Understanding Client
    Easy Schedule
    Stable team
    Stable Support Tools
    Appropriate team breakdown
    Flexible Budget
    Local team / customer
    Timely Resources
    Other (List)

10. Mark the constraints which apply to the project's (application's) software and where possible explain how it will be determined if the project (application) has met the constraints.

    This should indicate measures which will be used to determine if the constraints were met. For example:

    - The Useability might be determined by specific tests to be applied by the User or require that certain style guides are followed.
    - Real-time requirements might by tested experimentally, by simulators, or by a theoretical analysis of the problem.
    - Reliability may be determined as a measure of the code, based on tests, or based on performance in the field - such as the average number of days between failures.

12. Indicate the areas in which, when the project commenced, your staff have :
    (1) training,
    (2) experience (more than 6 months), or
    (3) training and experience.

    Language Used
    Process Used
    Hardware Used
    Development Environment
    Type of Application
    Support Tools
    Client
    Users
    Other (List)

13. Indicate how novel the project (application) is

    Nothing similar has ever been attempted
    Nothing similar has been attempted by the organisation
    Nothing similar has been attempted by the development team
    Variation on a standard product
    Other

14. Indicate other areas which may (have) effect(ed) staff productivity, and give details.

    Negative Effect

    Positive Effect

12. Indicate the areas in which your staff have (1) training, (2) experience (more than 6 months), (3) training and experience

The PROCESS is the method by which software is developed and tested and also includes house-keeping activities.

The DEVELOPMENT ENVIRONMENT is a combination of the Process and Resources used to develop a project (application) and the working conditions under which the project (application) is developed.

The TYPE OF APPLICATION is a combination of factors such as: Real-Time, Embedded, Scientific, Information Processing and Control.

SUPPORT TOOLS include Communications Tools, Project Planning Tools and Software Development Environments.

CLIENT and USER indicate if the client and user are available for answering questions about the requirements.

Part E. Progressive Refinement

(Group 2 Information as described in Section 3.1)

1. Organisation Number
2. Project Number
3. Complete the tables using the following guidelines:

   a) If overtime was recorded separately in the Effort record, record the overtime worked in the table below under the O/T columns.

   b) If overtime was included in the Effort recorded for the project, record the Effort in the table below under the E columns. Leave the O/T columns blank.

   c) If the overtime was not recorded and was different for each phase build, identify the approximate percentage under the O/T columns in the table below and write % at the top of the appropriate columns.

   d) If the overtime was not recorded and was approximately constant over the different phases of the project, please indicate the approximate overtime (as a %) for each staff category:

1. Development Overtime %	
2. Support Overtime %	
3. Sub-contractors Overtime %	

Appli-cation Number	Phase / Build	Effort ( )						Size (Capacity) Builds Only	Constraints Met Builds Only
		Develop-ment		Support		Sub-Contractors			
		E	O/T	E	O/T	E	O/T		

Motivation

A build of a system may not add any new Capacity (i.e. additional functionality), but may address other requirements such as time or memory constraints. Therefore, both the Capacity (as defined in Part C) and the number of constraints (see Part D) met by the build need to be recorded.

Example phases/builds are:

Requirements analysis
System design
Build 1 - Detailed design
Build 1 - Coding
Build 1 - Unit Testing
Build 1 - Integration Testing
Build 2 - (as for build 1)
Build 3 - Constraints test design
Build 3 - Constraints testing - Iteration 1 - n
Build 3 - Code adjustment - Iteration 1 - n-1

where n is determined by the test results but must be less than a specified value. Software Development Activities can also be included as separate phase/builds.

For each build indicate the application(s) effected.

Effort

In the brackets () indicate the units in which Effort (time spent by all developers) in developing the project) was recorded. It is preferable that the units used are staff-hours, so if alternative units are used, also indicate the conversion rate to obtain staff-hours. Effort should be given as it was recorded and not allow for unrecorded Overtime.
The Effort (in the units stated) of the organisation completing the questionnaire's project staff should be included in the Development Effort column. The Support Effort should indicate the Effort of support and managerial staff whose time is not booked to a particular project or activity. The final column should be used to indicate the Effort from sub-contractors.

If this questionnaire is being completed by the client or user, indicate the Effort spent by your organisation separately, and record information about the Contractors Development Effort in the appropriate location in the table.

Part F. Alternative Size Measures

(Group 2 Information as described in Section 3.1)

1. Organisation Number
2. Project Number
3. Application Number (if Applicable)

Size Measure (& Description)	Size (Total App/Proj)
LOC (Attach a description of how it was calculated).	
FP - Inputs	L: A: H:
FP - Outputs	L: A: H:
FP - Enquiries	L: A: H:
FP - Logical Internal Files	L: A: H:
FP - External Interface Files	L: A: H:
FP - Data Communications	
FP - Distributed Data Processing	
FP - Performance	
FP - Configuration Usage	
FP -Transaction Rates	
FP - On-line Data Entry	
FP - End-User Efficiency	
FP - On-line Update	
FP - Complex Processing	
FP - Re-useability	
FP - Installation Ease	
FP - Operational Ease	
FP - Multiple Sites	
FP - Facilitate Change	
FP - Unadjusted (If above info not available)	
FP - Adjusted (If above info not available)	
Sub-systems	
Ada Library Units (or equivalent - give language)	
Ada Packages (or equivalent - give language)	

This part may be completed at the application or the project level. It is preferable that it is complete at both levels wherever possible.

These measures are provided to allow estimates of software size, which can be determined after the concept evaluation stage of the development, to be incorporated in the progressive refinement model. It will also be used to compare the models from the literature with the Slicing/Progressive Refinement model.

Any measures currently being calculated for the project should be included. In addition, measures which can be determined from the code (such as Lines of Code), and those which are easy to determine (such as the Function Point Technology Factors) should be recorded.

Where possible copies of the definitions used should be attached. Standard definitions of Function Point components are given below. If alternative definitions are used, please attach them.

Function Point Function Types

There are five function types: External Input, External Output and External Enquiry are the three Transactional function types and Internal Logical File and External Interface File are the Data Business function types. If functionality is duplicated, it should only be counted once.

Internal Logical files are logical collections of data from the user's perspective which are maintained by transactions belonging to the application of interest. External Interface files are logical collections of data from the user's perspective which are referenced by the application of interest, but maintained by other applications. Temporary, backup, help, report and implementation dependent files should not be included under either category. The complexity of an Internal Logical File or and External Interface file is given by the table below.

		Data Element Types		
		1-19	29-50	>50
Normalised Tables (Record Types)	1	L	L	A
	2-5	L	A	H
	>5	A	H	H

External Inputs add, change or delete information from one or more Internal Logical files. External Enquiries extract information from the system. External Outputs are similar to External Enquiries except that the information is processed before being extracted.

The three Transactional function types are also classified into one of three complexity levels - (L)ow, (A)verage or (H)igh. The table to the right can be used to determine these classifications. The number of File Types Referenced is used to determine the appropriate column and the number of Data Element Types is used to determine the appropriate row. For Outputs, use the numbers in the shaded area and use the numbers in the clear area for Inputs. The appropriate level for Enquiries is determined by considering the input and the output side of the Enquiry separately and then choosing the higher complexity level.

			File Types Referenced		
	Output		<2	2-3	>3
		Input	<2	2	>2
Data Element Types	1-5	1-4	L	L	A
	6-19	5-15	L	A	H
	>19	>15	A	H	H

Function Point Technology Factors

The following table describes the factors to be considered when determining the appropriate value for each Function Point Technology Factor. The following notes apply to the table.

* User Efficiency features are:

Navigational aids (function keys, dynamic menus)

Hard copy user documentation of on-line transactions

Menu System	Automated cursor movement
Mouse interface	Pre-assigned function keys
Scrolling	Pop-up windows
Cursor selection of screen data	Heavy formatting (eg reverse video, colour)
Minimal number of screens	On-line help/documentation
Remote printing (on-line)	On-line batch submission
Bilingual (counts as 4)	Multi-lingual (counts as 6)

** Complex Processing features are:

Security or Sensitive Control (eg audit) processing

Extensive logical processing

Extensive mathematical processing

Extensive exception processing resulting in incomplete transactions

Multiple types of I/O processing (eg multi-media)

*** Operational Ease factors are:

Operator intervention required for start-up, back-up and recovery

Operator intervention not required for start-up, back-up and recovery (counts as 2)

Minimal need for tape mounts

Minimal need for paper handling

**** Facilitate change factors are:

Flexible query/report facility for requests which require access to one control file

Flexible query/report facility for requests which require access to multiple control files (counts as 2)

Flexible query/report facility for complex requests which require access to multiple control files (counts as 3)

Control data keep in user-maintained tables. Changes take place next business day.

Control data keep in user-maintained tables. Changes take place immediately. (counts as 2)

	0	1	2	3	4	5
Data Communications	Pure batch or isolated	Remote data entry or printing	Remote data entry and printing	On-line data collection or teleprocessing	One TP protocol supported	Multiple TP protocols supported
Distributed Data Processing	No aid to data transfer or processing	Prepares data for use on another processor	Preparation for, transfer to & processing on another CPU	Uni-directional distributed processing & data transfer	Distributed processing & data transfer in both directions	Processing functions are dynamically allocated
Performance	None	No special action required	Critical at peak times. Processing by next day.	Critical in bus. hrs. Deadlines constrained by interfaces	Performance analysis tasks required in design	Performance analysis tools required
Heavily Used Configuration	No restrictions	Less restrictive than typical	Security and timing considerations	Specific Processor requirements	Requires dedicated CPU or constraints on its use	Constraints on distributed components of the system
Transaction Rate	No peak periods	> Monthly peak periods	Weekly peak periods	Performance analysis used	Performance analysis tools	Tools for whole of life-cycle
On-line Data Entry	Batch mode	< 8% interactive	8-15% interactive	14-24% interactive	24-39% interactive	>30% interactive
Design for End User Efficiency *	0 features	1-3 features	4-5 features	6+ features	Tools required to check efficiency	Efficiency demonstrated to the user
On-line Update	No on-line updates	1-3 Control files	4+ Control files. Small volume.	Update of major control files.	Protection against data loss	Highly automated recovery
Complex Processing **	None	1 factor	2 factors	3 factors	4 factors	5 factors
Re-useability	None	Within Application	<10% application considered >1 user's needs	More than 1 user's needs considered	Source level customisation planned and documented	Customised by user parameters
Installation Ease	None	Special set-up required	Conversion and installation guides required.	(2) and Impact of conversion important	(2) and automated conversion and installation tool	(3) and automated conversion and installation tool
Operational Ease ***	None	1 factor	2 factors	3 factors	4 factors	Unattended operation
Multiple Sites	One site	Identical hardware and software	Similar hardware and software	Different hardware and / or software	(1) or (2) and documentation & support plans	(3) and documentation & support plans
Facilitate Change ****	None	1 factor	2 factors	3 factors	4 factors	5 factors

Part G. Detailed Size Measures

(Group 2 Information as described in Section 3.1)

1. Organisation Number
2. Project Number

Note: More detailed breakdown of Function Points etc can be given if available.

Module	Size (Capacity)	Size(LOC)	Size (FP)	Size (Other - Specify)

See Part F for a definition of Function Points and Part C for a table from which Capacity can be determined.

Part H. Detailed Cost Breakdowns

(Group 2 Information as described in Section 3.1)

1. Organisation Number
2. Project Number

Phase/Build	Module	Effort (Staff-Hours)	
		Development	Sub-Contractor

Effort

In the brackets () indicate the units in which Effort (time spent by all developers) in developing the project) was recorded. It is preferable that the units used are staff-hours, so if alternative units are used, also indicate the conversion rate to obtain staff-hours. Effort should be given as it was recorded and not allow for unrecorded Overtime.

The Effort (in the units stated) of the organisation completing the questionnaire's project staff should be included in the Development Effort column. The final column should be used to indicate the Effort from sub-contractors.

Module

For these purposes a module is considered to be any part of the system which can be separately compiled. If Effort was not tracked to this level of detail, but was associated with collections of modules, associate the Effort with the relevant group of modules. You may identify "Super-Modules" and use their names in the Module column.

Phase/Build

These should be the same as those given in Part E. Where Phase/Build information is not given, it will be assumed that the Phase/Build is the same as that in the previous row.

Questionnaire

Part I. Alternative Cost Information

(Group 2 Information as described in Section 3.1)

1. Organisation Number
2. Project Number
3. Application Number (if applicable)

4. COCOMO

1. Is the project:

   Organic

   Semi-Detached

   Embedded

2. Tick the appropriate box for the following attributes:

Driver	Very Low	Low	Nominal	High	Very High	Extra High
Reliability						
Data Base Size						
Product Complexity						
Execution Time Constraint						
Main Storage Constraint						
Virtual Machine Volatility						
Computer Turn Around Time						
Analyst Capability						
Applications Experience						
Programmer Capability						
Virtual Machine Experience						
Prog. Language Experience						
Modern Programming Practices						
Use of Software Tools						
Required Development Schedule						

5. Other

As used by your organisation.

This part may be completed at the application or the project level. It is preferable that it is complete at both levels wherever possible.

COCOMO

The COCOMO model is described in [Boehm, 1984]. Tables describing Boehm's Cost Drivers are given on the next two pages. The terms Organic, Semi-Detached and Embedded refer to the project's "Development Mode". The guidelines given by Boehm are:

Organic Developments are familiar, with stable requirements. They are relatively unconstrained and forgiving. [Heemstra, 1992] says that the system being developed is also relatively small.

Embedded Developments are unfamiliar, ambitious, unforgiving and tightly constrained. [Heemstra, 1992] says they also have volatile requirements.

Semi-detached Developments fall between Organic and Embedded Developments.

Other

This should include the names of the approaches used in your organisation. For each approach, the names and values of all the factors used should be given. A description of how the factors, and how they are determined, should also be given, although references to publicly available documents are sufficient. Where there is insufficient room, additional sheets should be attached.

The following table which describe the attributes used in the COCOMO model is derived from that given in [Boehm, 1984].

Driver	Very Low	Low	Nominal	High	Very High	Extra High
Reliability	Slight inconvenience	Low, easily recoverable losses	Moderate, recoverable losses	High financial loss	Risk to human life	
Data Base Size *	D/P > D/P ≤	0 10	10 100	100 1000	1000	
Product Complexity	See Separate Table					
Execution Time			≤ 50% of available	≤ 70%	≤ 85%	≤ 95%
Main Storage Constraint			≤ 50% of available	≤ 70%	≤ 85%	≤ 95%
Virtual Machine Volatility	Major: Minor: (Change)	12 months 1 month	6 months 2 weeks	2 months 1 week	2 weeks 2 days	
Computer Turn Around Time		Interactive	Average turnaround <4 hours	4-12 hours	>12 hours	
Analyst Capability **	15th	35th	55th	75th	90th	
Applications Exp.	≤ 4 months	1 year	3 years	6 years	12 years	
Programmer Capability **	15th	35th	55th	75th	90th	
Virtual Machine Exp.	≤ 1 month	4 months	1 year	3 years		
Prog. Lang. Exp.	≤ 1 month	4 months	1 year	3 years		
Modern Prog. Practices	No use	Beginning Use	Some Use	General Use	Routine Use	
Use of Software Tools	Basic micro-processor tools	Basic mini tools	Basic mid/maxi tools	Strong, maxi prog., test tools	Activity-based tools	
Development Schedule	75% of nominal	85%	100%	130%	160%	

* D stands for the size of the database in bytes and P stands for the program's delivered source instructions.

** The numbers given reflect the percentiles with respect to analysis or programming ability, efficiency, communication, cooperation.

The following table describes the product complexity using in the COCOMO model as given in [Boehm, 1984]. Note that SP stands for structured programming.

	Control Operations	**Computational Operations**	**Device-Dependent Operations**	**Data Management Operations**
Very Low	Straight-line code with non-nested SP operators. Simple predicates	Evaluation of simple expressions. A = B + C * (D - E)	Simple read, write statements with simple formats	Simple arrays in main memory
Low	Straightforward nesting of SP operators. Mostly simple predicates.	Evaluation of moderate-level expressions. D = SQRT(B**2 - 4 * A * C)	No cognisance needed of overlap or the particular processor or I/O device. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files.
Nominal	Mostly simple nesting. Some inter-module control. Decision tables.	Use of standard maths and statistical routines. Basic matrix or vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits.
High	Highly nested SP operators with many compound predicates. Queue and stack control. Considerable inter-module control.	Basic numerical analysis: (NA) multi-variate interpolation, ODEs. Basic truncation, round-off concerns.	Operations at physical I/O level (storage address translations, seeks, reads, etc). Optimised I/O overlap.	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level.
Very High	Re-entrant and recursive coding. Fixed-priority interrupt handling.	Structured NA: near-singular matrix equations, partial differential equations (PDEs).	Routines for interrupt diagnosis, servicing, masking. Communication line handling.	A generalised, parameter-driven file structuring routine. Search optimisation, command processing, file building.
Extra High	Multiple resource scheduling and dynamic priorities. Micro-code-level control.	Difficult and unstructured NA highly accurate analysis of noisy, stochastic data.	Device timing-dependent coding, micro-programmed operations.	Highly coupled, dynamic relational structures. Natural language data management.

Part J. Other

(Group 3 Information as described in Section 3.1)

1. Organisation Number
2. Project Number
3. Application Number (if applicable)

Please give details of any other factors which you feel effect the cost of developing software systems. For example, indicate tools which effected productivity, hardware and staffing problems, and requirements changes.

Appendix B: Guidelines for Facilitators

FACILITATORS

Facilitators are the people who have agreed to assist in obtaining the data from the organisations participating in these data collection activities. Their roles are to:

1. Identify and contact organisations which may be able to supply data
2. Supply these organisations with the information they require before entering a data collection agreement
3. Ensure that these organisation understand that:
   - the data will be treated as Commercial-In-Confidence
   - the data will not be used to evaluate the organisations
   - they will have early access to the results of all studies on the data
   - supplementary information may be requested of them at a later data (if these requests are not met, then the final models may be less applicable to their organisation)
4. Prepare copies of the questionnaire for each organisation which agrees to supply data, contacting the iMAPS team for organisation numbers when required
5. Interview the organisations or supply them with copies of the questionnaire
6. Assist the organisations to identify potential electronic sources of the requested information
7. Collate cross-referencing information between the developers, sub-contractors, clients and users for each project
8. Deliver the collected information to the iMAPS team
9. Identify (and contact) other potential facilitators
10. Ensure that other facilitators understand the importance of the conditions listed in 3 and the nature of the data to be collected.

PROJECT SELECTION

Information should be collected for as many projects as possible and should not be limited to successful projects. Data can be collected on past projects as well as from on-going projects. While it may not be possible to complete the entire questionnaire for past projects, any available information should be supplied. Where the only information available is likely to be incomplete or inaccurate then no information should be collected for that project. (For example, if there is no record of the actual Effort worked by software development stage (build or phase) and no-one is able to answer the Group 1 questions about the intended system.)

Where the number of projects identified exceeds the number from which data could reasonably be collected, then priorities must be assigned to the projects and/or more facilitators identified. Priority should be given to the most recent projects, and on-going projects. Where further restrictions are required, the facilitators should concentrate on collecting information from projects developed by a small number of organisations and for a small number of application domains. Additional projects can then be considered, as time permits.

QUESTIONNAIRE

Once potential projects have been identified, the client's project leaders, the end users and all contractors (see Contacts) should be contacted to determine if they are willing to supply data.

All those who agree to supply data should "complete" the questionnaire, either through an interview with the facilitator or directly.

The Questionnaire was designed to facilitate the collection of three Groups of information. The first two groups address the two phases of the conjectured approach. While it is preferred that the questions from both groups are answered, the answers to only one of these groups are necessary. It is likely that Group 1 and Group 3 questions will need to be answered manually. However, it is anticipated that most of the information requested for the Group 2 questions will be available electronically.

Provision of electronic forms of the information, where available, is preferable to manual completion of the questionnaire.

The copies of the questionnaire should include an appropriate Organisation, Project, and possibly Application Number as described below. Cross reference information relating the questionnaire to the project should be retained.

ORGANISATION NUMBERS

These are used to distinguish different organisations. They should be supplied by the iMAPS team. However, facilitators may be given a range of numbers which they can allocate to the organisations they contact.

PROJECT AND APPLICATION NUMBERS

These may be determined by the facilitator or the organisation. However, unique project numbers should be used for each project in which a given organisation is involved and unique application numbers should be used for each application in a project.

CONTACTS

Development Teams:	Questions should be answered by team leader.
Sub-contractors:	Questions should be answered by sub-contractor's team leader for any applications they developed.
Clients:	Questions should be answered by project leaders to the best of their knowledge. They should be encouraged to answer the Group 1 questions (Part C, Q 1-7 and Part D, Q 1-6) - in particular.
Users:	Where possible, the Group 1 questions (Part C, Q 1-7 and Part D, Q 1-6) should be answered by a user which a deep knowledge of the product being developed.

Cross referencing information between these organisations and the projects should be maintained.

RETURNING RESULTS

The collected information should be returned to the iMAPS Team by email or post C/o one of the authors.

Gina Kingston email:Gina.Kingston@DSTO.defence.gov.au
Software Engineering (SE) Group ph : +618 259 6611
Information Technology Division (ITD) fax: +618 259 5589
Defence Science and Technology Organisation (DSTO)
PO Box 1500
Salisbury, SA 5108
AUSTRALIA

B.1 Providing Processing Information

The questions for the Processing component of the Capacity concept are those most likely to be misinterpreted. For this reason this section describes how to determine the Processing for a collection of related example projects. The generic example considers a Parser for either a Programming Language, or a Natural Language, which is specialised into four more concrete examples as the result of considering the information requested.

B.1.1 Providing the Information for Part C

For Q7, Part C we need to determine the number of Operations, Objects and Areas the processing spans.

Areas of Processing:

The following table lists the areas of processing and each area was considered in turn.

Areas of Processing	
×	Calculation
√	Manipulation
√	Obtain/Retrieve
√	Store
	Transfer
	Present
×	Monitor
×	Act

Calculation: While certain parsing algorithms may require limited calculations to keep track of the number of entities etc, this is not an essential part of the nature of parsing and no calculation is recorded.

Manipulation: The main function of parsing is to determine the structure of a sentence by identifying and classifying its components. Therefore manipulation is one of the main areas of processing.

Obtain/Retrieve: This is required to read data into the system.

Store, Transfer and Present: The system should do something with the results it produces. They will need to be stored for later use, transferred to another (parts of the) system, or presented to the user.

Monitor: If we assume that the amount of text to be parsed is fixed there is no need to monitor for additional input etc.

Act: A parser does not control the movement of any physical (or pseudo-physical) activities.

Therefore the number of Areas falls into the 3-6 category.

Operations:

As discussed above, processing is the main area, so the operations in this area should be considered. It was previously stated that the function of a parser was to:

determine the structure of a sentence by

- identifying and
- classifying its components.

We need to determine if these are Simple or Complex operations.

Consider the following situations:

1. The parser is for a simple assembly language with no ambiguities.
2. The parser is for a programming language, such as Ada, with ambiguities.
3. The parser is for the English language, with no ability to handle incorrect grammar, partially completed sentences, or recognise special phrases, or the meaning of verbs.
4. The parser is for the English language and must handle all of the above.

Obviously the operations in the first case are Simple and in the last case are Complex. The other two require more consideration, but should be considered Simple operations. (Note that the work to define these operations may be considerable, but the problem of defining the operations is a consideration for Difficulty and not Capacity.)

Objects:
The objects need to be classified into one of three categories, Few, Similar or Diverse.

Consider again the four examples described above.

In the first example, operands, operators and possibly labels will need to be identified. Therefore the classification should be Few.

In the second example, a variety of components will need to be identified: variables, sub-programs, numbers, tasks etc. This means that the classification should be at least Similar. In addition, anomalies will need to be identified. This example is a border-line case. However, as there is only a small exception to the Similar rule and the types of objects are not very Diverse, the system should be classified as Similar.

In the third example, a variety of components will need to be identified: verbs, nouns, articles, pronouns, adjectives, adverbs, noun phrases etc. Therefore the classification should be Similar.

In the fourth example, the objects are similar to the third. Additional objects would be required for slang, dialects, and to associate words or phases to their meanings. Again, this is a border-line example. And while this system may seem even harder to classify it should again be classified as Diverse. The main reason for this is the necessity to associate words and phrases to their meanings. Without greater understanding of the purpose of this system or its implementation, it should be considered as Diverse.

Summary:
Using Table B-2 the processing levels shown in Table B-1 are obtained for the four examples of parsing systems.

Table B-1: Calculated Processing Levels

Example	Operations	Objects	Areas	Level
1	Simple	Few	3 (2-6)	C
2	Simple	Similar	3 (2-6)	F
3	Simple	Similar	3 (2-6)	F
4	Complex	Diverse	3 (2-6)	R

Table B-2: Processing Levels

Level	Usual Characteristics		
	Operations	Objects	Areas
A	No processing		
B	Basic	Few	1
C	Basic	Few	2-6
D	Basic	Few	7-8
E	Basic	Similar	1
F	Basic	Similar	2-6
G	Basic	Similar	7-8
H	Basic	Diverse	1
I	Basic	Diverse	2-6
J	Basic	Diverse	7-8
K	Complex	Few	1
L	Complex	Few	2-6
M	Complex	Few	7-8
N	Complex	Similar	1
O	Complex	Similar	2-6
P	Complex	Similar	7-8
Q	Complex	Diverse	1
R	Complex	Diverse	2-6
S	Complex	Diverse	7-8

B.1.2 Providing the Operations Information for Part D

For Q6, Part D we need to complete the Operations and Characteristics tables. The four examples considered for Part C will also be considered for Part D.

As discussed in Part C, there is no Calculation, Monitoring or Acting so these rows may be easily completed. For the purposes of this example, we will assume that the results will be stored and that there will be no Transfer or Present components to the overall Processing. Therefore the Calculation, Transfer, Present, Monitor and Act rows of the Table B-5 contain all zeros (0).

Operation Level:
The generic description of Operation Levels is given in Table B-3. More detailed information on the Levels for different Operations for operations from different processing areas is given in Table B-4.

Table B-3: Generic Classification of Operation Levels

Operation Level	Types of Operations
0	Not used
1	Single, isolated, operations
2	Simple combinations of operations performed on simple types
3	Operations performed on simple objects
4	Operations performed on complex objects
5	Operations performed on collections of very similar objects
6	Operations performed on collections of objects

The Manipulation, Store and Obtain operations are all performed on data structures. In parsers it is generally assumed that the input is a sequence of tokens which are processed one at a time, in order. As at the time the tokens are obtained, they are considered identical by the system, the obtain operational level is 1. The final result is generally a parse-tree, and it is unlikely that more complex structures are used while parsing so the Store and Manipulate operations are of the same Level. As the results are tree-structured, this must be at least level 5 and as the objects stored in the tree are Similar (except for example 4, where they are considered Diverse) this is the appropriate level. (Example 4 should be at level 6).

Table B-4: Operation Levels by Processing Area

<table>
<tr><th></th><th>1 & 2</th><th>3</th><th>4</th><th>5</th><th>6</th></tr>
<tr><td>Calculation</td><td>Integers, Reals</td><td>Vectors</td><td>Matrices etc (Fixed Size & Dimension)</td><td>Matrices etc (General, Regular)</td><td>Other</td></tr>
<tr><td>Manipulation</td><td rowspan="5">Single Values</td><td rowspan="5">Linear Objects</td><td rowspan="5">Non-linear Objects (Common Components)</td><td rowspan="5">Non-linear Objects (General Components)</td><td rowspan="5">Non-linear Objects (Varied Components)</td></tr>
<tr><td>Obtain/Retrieve</td></tr>
<tr><td>Store</td></tr>
<tr><td>Transfer</td></tr>
<tr><td>Present</td></tr>
<tr><td>Monitor</td><td>Switches</td><td>Fixed moves</td><td>"Linear"</td><td>2-D</td><td>3-D</td></tr>
<tr><td>Act ("Movement")</td></tr>
</table>

Operation Rates:

The Operation, Object and Application Domain Rates are given in Table B-6. Using this information the Operation Rates can be obtained for the Processing areas, Manipulation, Obtain and Store. For the purposes of this exercise we will assume that one Obtain and one Store operation are required. In practice this is likely to vary from system to system and may only be guessed at the early stages in the project. Thus the Obtain and Store areas have an Operation Rate of 1.

The Operation Rate for the Manipulation area is different for the four examples given.

For examples 1 and 3, there will be one main operation to distinguish different types of objects, so their Operation Rate will be 1.

For example 2, there will be a few additional operations to process anomalies, so its Operation Rate will be 2.

For example 4, a more detailed description is required to refine the number of operations. However one or more operations will be required to handle each of:

- grammatically correct English
- incomplete sentences
- incorrect grammar
- slang
- dialects
- identifying phrases with particular meanings
- identifying the meanings of words.

Because of the nature of some of these requirements, it is likely that more than operation will be required. Therefore the minimum number of operations required is 7 and the maximum is not known. The Operation rate could be assumed to be in the 6-10 category, but as the number of operations is somewhat vague, and it is likely that there will be more than one operation per requirement rating 4, the 11-15 category is recommended.

Table B-5: Operations Details

	(Maximum) Operation Level				(Maximum) Operation Rate				Object Rate				Application Rate			
Example =>	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Calculation	0				0				0				0			
Manipulation	5	5	5	6	1	2	1	4	2	6	4	7	1	1	1	2
Obtain/Retrieve	1				1				1				1			
Store	5	5	5	6	1				1				1			
Transfer	0				0				0				0			
Present	0				0				0				0			
Monitor	0				0				0				0			
Act	0				0				0				0			

Object Rate:

The Obtain and the Store will operate on single, although different, object types. While this is almost certainly true for the first three examples, the fourth example is not detailed enough to determine if different types of results will be stored. For the purposes of this example we will consider that only one type of result is necessary. Therefore all four systems have an Object Rate of 1 for the Obtain and Store areas.

Table B-6: Operation, Object and Application Rates

Operation/Object/Application Rate	Operations (Objects, Application Domains)
0	None
1	Single operation/object/domain
2	Few operations/objects/domains (2-5)
3	Low number of operations/objects/domains (6-10)

4	Medium number of operations/objects/domains (10-15)
5	High number of operations/objects/domains (16-20)
6	Large number of operations/objects/domains (21-30)
7	Complete library of operations/objects/domains (specific)
8	Complete library of operations/objects/domains (generic)

The Object Rate for the Manipulation area needs to be considered separately for each example. This is a more complex rating than that used in Part C, which also captures the domains of the objects . The ratings given on this scale for the four examples were:

1. Few
2. Similar
3. Similar (just)
4. Diverse (diverse)

Table B-7 shows the correlation between ratings used in Part C and the Object and Application ratings used here for Part D. The values we obtain for the Object and Application Rates for Part D can be checked against this table.

Table B-7: The Correlation between Part C and Part D ratings.

Part C Rating	Object Rate	Application Rate
Few	1 - 3	1
Similar	4 - 7	1
Diverse	8 1 - 8	1 2 or 3 - 8

As discussed previously, there are two or three types of objects for Example 1, so its object level is 2.

For Example 2, an incomplete list of four or five objects was given. The actual structure is much more complex, and while it will not be calculated at this stage it is likely that there are over twenty types of objects, including anomalies. Therefore, the Object Rating for this example should be 6.

For Example 3, an incomplete list of objects was again given. However, it is likely that less objects would be needed that for the previous example. (As this is only a simple English structure parser it is not likely that verbs of different tenses etc would be distinguished). Again the actual number will not be determined at this stage, but it is likely to be slightly more than 10. The 11-15 category, or Rating 4 was chosen.

The final example attempts to give a complete classification for the English language, so is Rating 7. Because the parser is meant to recognise different dialects, it might be thought that it should be given a rating of 8. However, a rating of 8 should be used to indicate very different objects. Even parsing of different Latin based languages would have similar objects which would result in the Parser having an Object rate of 7.

Application Rate

The Application Rate for the first three examples is 1 in all three areas, Obtain, Manipulate and Store. The Application Rate is also 1 for the areas Obtain and Store in the fourth example. The Application Rate for the Manipulation area of the fourth example is harder to determine.

In a banking system, loans, tax, and savings are all considered different areas. However, no such fine lines exist for domains in language definition. (Although, the inclusion languages which were not based on Latin would obviously involve multiple domains.) Considering the nature of the different activities considered under the "Operation Rate" heading there appears to be multiple domains. For example, the identification of the structure of the sentences and the association of phrases with particular meanings appear to belong to different domains. (It is not clear if the association of meanings to individual words belongs in one of these domains or is a separate domain.) Thus the number of domains appears to lie in the 2-5 region, resulting in a rating of 2.

Operations Table

The completed Operations table is given in Table B-5). The next step is to complete the manipulations table.

B.1.3 Providing the Characteristics Information for Part D

This information is to be supplied in Table B-8. As stated previously, there is no Calculating, Transferring, Presenting, Monitoring, or "Act"ing performed by the four examples considered. These areas are shaded in grey in the table.

Additional information is required to complete the remainder of the table for any of the examples. For the first three examples, it is likely that only a few of the options, apart from those associated with error checking, would be marked. However, very different systems could result for the fourth example depending on the options checked.

Table B-8: System characteristics

C	M	O	S	T	Pr	Mo	A	**Handles**
								Boundary conditions
								Over/underflow
	na	na	na	na		na	na	Round-off Errors
na	na					na	na	Device Errors
								High or varying accuracy
		na	na	na				High or varying precision (granularity)
								Partial Solutions
			na					Uncertain information
								Incomplete information
		na			na			Back-tracking
								Special cases (eg near singular matrices)
								Other Errors - Correction
								Other Errors - Fail safe

iMAPS: Collecting Data for Software Costing

Gina Kingston, Martin Burke and Peter Fisher

(DSTO-GD-0090)

DISTRIBUTION LIST

Number of Copies

AUSTRALIA

DEFENCE ORGANISATION

S&T Program

Chief Defence Scientist)	
FAS Science Policy)	1 shared copy
AS Science Industry External Relations)	
AS Science Corporate Management)	
Counsellor, Defence Science, London	Doc Control sheet
Counsellor, Defence Science, Washington	1
Senior Defence Scientific Adviser)	1 shared copy
Scientific Adviser - Policy and Command)	
Assistant Secretary Scientific and Technical Analysis	1
Navy Scientific Adviser	3 copies of Doc Control sheet and 1 distribution list
Scientific Adviser - Army	Doc Control sheet and 1 distribution list
Air Force Scientific Adviser	1
Director Trials	1
Director, Science Policy - Force Development and Industry	1
Science Policy - Defence Industry	2
Director, Aeronautical & Maritime Research Laboratory	1

Electronics and Surveillance Research Laboratory

Chief Information Technology Division	1
Research Leader Command & Control and Intelligence Systems	1
Research Leader Military Computing Systems	1
Research Leader Command, Control and Communications	1
Executive Officer, Information Technology Division	Doc Control sheet
Head, Information Architectures Group	Doc Control sheet
Head, C3I Systems Engineering Group	1
Head, Information Warfare Studies Group	Doc Control sheet
Head, Software Engineering Group	1
Head, Trusted Computer Systems Group	Doc Control sheet
Head, Advanced Computer Capabilities Group	Doc Control sheet
Head, Computer Systems Architecture Group	Doc Control sheet
Head, Systems Simulation and Assessment Group	Doc Control sheet
Head, Intelligence Systems Group	Doc Control sheet
Head Command Support Systems Group	1

Head, Exercise Analysis Group	Doc Control sheet
Head Information Management and Fusion Group	Doc Control sheet
Head Human Systems Integration Group	Doc Control sheet
Publications and Publicity Officer, ITD	1
Gina Kingston	2
Martin Burke	2
Peter Fisher	2
DSTO Library	
Library Fishermens Bend	1
Library Maribyrnong	1
Library DSTOS	2
Library, MOD, Pyrmont	Doc Control sheet
Forces Executive	
Director General Force Development (Sea),	Doc Control sheet
Director General Force Development (Land),	Doc Control sheet
Director General Force Development (Air),	Doc Control sheet
Army	
ABCA Office, G-1-34, Russell Offices, Canberra	4
S&I Program	
Defence Intelligence Organisation	1
Library, Defence Signals Directorate	Doc Control sheet
B&M Program (libraries)	
OIC TRS, Defence Central Library	1
Officer in Charge, Document Exchange Centre (DEC),	1
US Defence Technical Information Center,	2
UK Defence Research Information Centre,	2
Canada Defence Scientific Information Service,	1
NZ Defence Information Centre,	1
National Library of Australia,	1
Universities and Colleges	
Australian Defence Force Academy	1
Library	1
Head of Aerospace and Mechanical Engineering	1
Senior Librarian, Hargrave Library, Monash University	1
Librarian, Flinders University	1
Professor Ross Jeffery, University of New South Wales	1
Professor Ray Offen, JRCASE, Macquarie University	1
Professor Richard Jarret, University of Adelaide	1
Simon Timcke, University of Adelaide	
Other Organisations	
NASA (Canberra)	1
AGPS	1
State Library of South Australia	1
Parliamentary Library, South Australia	1

OUTSIDE AUSTRALIA

TTCP	
XTP-2 National Leaders	4
Abstracting and Information Organisations	
INSPEC: Acquisitions Section Institution of Electrical Engineers	1
Library, Chemical Abstracts Reference Service	1
Engineering Societies Library, US	1
American Society for Metals	1
Documents Librarian, The Center for Research Libraries, US	1
Information Exchange Agreement Partners	
Acquisitions Unit, Science Reference and Information Service, UK	1
Library - Exchange Desk, National Institute of Standards and Technology, US	1
SPARES	10
Total number of copies:	**82**

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION
DOCUMENT CONTROL DATA

1. PRIVACY MARKING/CAVEAT (OF DOCUMENT)
N/A

2. TITLE
iMAPS: Collecting Data for Software Costing

3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION)
Document (U)
Title (U)
Abstract (U)

4. AUTHOR(S)
Gina Kingston, Martin Burke and Peter Fisher

5. CORPORATE AUTHOR
Electronics and Surveillance Research Laboratory
PO Box 1500
Salisbury SA 5108

6a. DSTO NUMBER	6b. AR NUMBER	6c. TYPE OF REPORT	7. DOCUMENT DATE
DSTO-GD-0090	AR-009-686	General Document	April 1996

8. FILE NUMBER	9. TASK NUMBER	10. TASK SPONSOR	11. NO. OF PAGES	12. NO. OF REFERENCES
N9505/10/80	iMAPS	DST	98	19

13. DOWNGRADING/DELIMITING INSTRUCTIONS
N/A

14. RELEASE AUTHORITY
Chief, Information Technology Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT
APPROVED FOR PUBLIC RELEASE

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE CENTRE, DIS NETWORK OFFICE, DEPT OF DEFENCE, CAMPBELL PARK OFFICES, CANBERRA ACT 2600

16. DELIBERATE ANNOUNCEMENT
No limitation

17. CASUAL ANNOUNCEMENT Yes

18. DEFTEST DESCRIPTORS
Computer programs
Cost analysis
Department of Defence (Australia)
iMAPS

19. ABSTRACT
This paper discusses the iMAPS Software Costing conjectures, and documents the data required to calibrate and validate the models. It discusses issues related to the collection of the data, including the benefits to participants, and the significance of this research to the Australian Defence Organisation.