Software Estimation Techniques

Contents

Introduction

- A successful project is one that is delivered

  -On time

  -Within budget

  -With the required functionality
- It means that targets are estimated and set for all these aspects, and project manager then tries to meet those targets.
- Real estimates are crucial, because of incorrect estimates, a project cannot meet its deadline.

Difficulties in Estimation

- **Nature of software**,
  - Complexity and invisibility of software.
- **Subjective nature of estimating** , For example, some research shows that people tend to underestimate the difficulty of small tasks and over-estimate that of large ones.
- **Political implications,**
  - Different objectives of people in an organization.
  - Managers may wish to reduce estimated costs in order to win support for acceptance of a project proposal.
- **Changing technologies,**
  - Technology is rapidly changing , making the experience of previous project estimates difficult to use in new ones.
- **Lack of homogeneity of project experience,**
  - Experience on one project may not be applicable to another.

Where are Estimates Done?

Estimates are carried out at different stages of a software project for a variety of reasons .

- **Feasibility study**
    - Estimates here conforms that the benefits of the potential system will justify the costs.
- **Strategic planning**
    - Project portfolio management is involved here. Benefits and costs of new projects are estimated to allocate priorities.
- **System specification**
    - Design of a system shows that how user requirements will be fulfilled .
    - Different design approaches can be considered for a single requirement specification.
    - The effort needed to implement different design proposals is estimated here.
    - Estimates at the design stage will also confirm that the feasibility study is still valid.

- **Evaluation of suppliers proposals**
  - A manager could consider putting development out to tender.
  - Potential contractors would examine the system specifications and produce estimates (their bid).
  - The manager can still produce his own estimates why ?

    -To question a bid that seems too low which could be an indication of a bad understanding of the system specifications . Or to compare the bids to in-house development.
- **Project planning**
  - As the planning and implementation of the project becomes more detailed, more estimates of smaller work components will be made. These will confirm earlier broad estimates.

Problems with over and under estimates

- **An over-estimate** is likely to cause project to take longer than it would otherwise. This can be explained by the application of two laws:
  - **Parkinson's Law:** 'Work expands to fill the time available'

    -Over estimating the duration required to complete a target will cause staff to work less hard in order to fill the available time.
  - **Brook's Law :** 'Putting more people on a late job makes it later'

    -If there is an overestimate of the effort required, this could lead to more staff being allocated than needed and managerial overheads being increased. And as the project team grows in size, more effort will be required for management, coordination and communication.
- **Under-estimating a project**
  - Can cause the project to not be delivered on time or cost
  - But still could be delivered faster than a more generous estimate

- On the other side the danger of underestimating a project is the effect on the quality.
    - Staff, particularly those with less experience, could response to pressing deadlines by producing substandard work.
    - Substandard work might only become visible at the later (testing) phases of a project, where extensive re-work can easily delay project completion.
    - Motivation is also decreased when targets are always unachievable.

Basis for software Effort Estimation

- The need for historical data
- Parameters to be estimated
- Measure of work
- Measure of effort

The need for historical data

- Most estimating methods need information about past projects.
- Care has to be considered when applying past performance to new projects because of possible differences in factors such as :

  -Different programming languages

  -Different experience of staff
- There are international Data Base containing data about thousands of projects that can be used as reference.

Parameters to be estimated

- Two project parameters are to be estimated for carrying out project planning.

  -Duration: It is usually measured in months, Work-month (wm) is a popular unit for effort measurement.

  -Effort: Popular unit for effort measurement is person-month (pm), One pm is the effort an individual can typically put in a month.

Measure of Work

- Measure of work involved in completing a project is also called the size of the project. Work itself can be characterized by cost in accomplishing the project and the time over which it is to be completed.
- Direct calculation of time and cost to implement software is difficult in early stages as they both depend on :

  -The developer’s capability and experience

  -The technology that will be used
- It is therefore a standard practice to first estimate the project size, and by using it, the effort and time taken to develop the software can be computed.
- At present, two metrics are being popularly used to measure size

  -Source lines of code (SLOC)

  -Function point (FP)

Measure of effort

- Person-month (pm) is a popular unit for effort estimation.
- It quantifies the effort that can be put in by one person over one month.
- Effort that has been put in by a team can be measured in units of ***pm*** based on the number of persons deployed and the number of months that they have worked.

Software effort estimation techniques

- **Algorithmic models:** which use ‘effort drivers’ representing characteristics of the target system and the implementation environment to predict effort;
- **Parkinson:** Staff effort available to do the project becomes the estimate.
- **Price to win:** Here the estimate is a figure that seems sufficiently low to win a contract.
- **Expert judgment:** Based on the advice of knowledgeable staff.
- **Analogy:** A similar completed project is identified and its actual effort is used as the basis of the estimate.
- **Bottom-up:** Component tasks are identified and estimated and these individual estimates are aggregated.
- **Top-down:** An over all estimate for the whole project is broken down into the effort required for component tasks.

Expert Judgment

- It involves asking for an estimate of task effort from someone who is knowledgeable either about the application or the development environment.
- This method is often used when estimating the effort needed to change an existing piece of software.
- The estimator examine the existing code in order to judge the proportion of code affected and from that derive an estimate.
- Some one familiar with the software would be in the best position to do that.

Estimation by Analogy

- This is also called **case-based reasoning.**
- The estimator identifies completed projects (**source cases**) with similar characteristics to the new project (the **target case**).
- The new project is referred to as the target project or target case.
- The completed projects are referred to as the source projects or source case.
- The effort recorded for the matching source case is used as the base estimate for the target project.
- The estimator calculates an estimate for the new project by adjusting the (base estimate) based on the differences that exist between the two projects.
- There are software tools that automate this process by selecting the nearest project cases to the new project.
- Some software tools perform that by measuring the Euclidean distance between projects.

- The Euclidean distance is calculated as follows :

Distance = square-root of (($\textbf{target\_parameter}_1$-$\textbf{source\_parameter}_1)^2$ +….+ ($\textbf{target\_parameter}_n$ - $\textbf{source\_parameter}_n$ $)^2$)

- Assume that cases are matched on the basis of two parameters , the number of inputs and the number of outputs .

  -The new project (target case) requires 7 inputs and 15 output

- You are looking into two past cases or source cases to find a better analogy with the target project :

  - Project A : has 8 inputs and 17 outputs .

  - Project B : has 5 inputs and 10 outputs .

- Which is a more closer match for the new project A or project B ?

- Distance between new project and project A :

  Square-root of ((7-8)^2 + (15-17)^2)= 2.24

- Distance between new project and project B:

  Square-root of ((7-5)^2 + (15-10)^2)= 5.39

- Project A is a better match because it has less distance than project B to the new project.

There is a new project, that is known to require 7 inputs and 15 outputs. There are two past cases: project A has 8 inputs and 17 outputs and project B has 5 inputs and 10 outputs. Which of the source projects have better analogy with the target new project?

The Top-down Approach and Parametric Models

- The top-down approach is normally associated with **parametric (or algorithmic) models**. These may be explained using the analogy of estimating the cost of rebuilding a house.
- This is of practical concern to houseowners who need insurance cover to rebuild their property if destroyed.
- Insurance companies, however, produce convenient tables where the house-owner can find estimates of rebuilding costs based on such **parameters** as the number of stores and the floor space of a house. This is a simple parametric model.
- A parametric model will normally have one or more formulae in the form:

  effort = (system size) * (productivity rate)
- For example, system size might be in the form ‘thousands of lines of code’ (KLOC) and have the specific value of 3 KLOC while the productivity rate was 40 days per KLOC. These values will often be matters of judgement.

Bottom-up Estimation

- With the bottom-up approach the estimator breaks the project into its component tasks. With a large project, the process of breaking it down into tasks is iterative: each task is decomposed into its component subtasks and these in turn could be further analyzed.
- Stop when you get to what one person can do in one/two weeks.
- Estimate costs for the lowest level activities.
- At each higher level calculate estimate by adding estimates for lower levels.
- When a project is completely novel or there is no historical data available, bottom-up approach would be the perfect choice.
- Following are the steps how a bottom up approach can be used for estimating effort
  - **Think about the number and types of software modules in the final system,** Most information systems, for example, are built from a small set of system operations, e.g. Insert, Amend, Update, Display, Delete, Print. The same principle should equally apply to embedded systems, albeit with a different set of primitive functions.

- **Estimate the SLOC of each identified module,** One way to judge the number of instructions likely to be in a program is to draw up a program structure diagram and to visualize how many instructions would be needed to implement each identified procedure.
- **Estimate the work content, taking into account complexity,** This factor will depend largely on the subjective judgement of the estimator. For example, the requirement to meet particular highly constrained performance targets can greatly increase programming effort.
- **Calculate the work days effort,** Historical data can be used to provide ratios to convert weighted SLOC to effort.

- Note that the steps above can be used to derive an estimate of lines of code that can be used as an input to one of the COCOMO models.
- Most commonly used top down estimation approaches are **Function Points Mark II** and **COCOMO II**.

Function Points Mark II

- The function point is a "unit of measurement" to express the amount of business functionality an information system provides to a user.
- It is a process which defines the required functions and their complexity in a piece of software in order to estimate the software's size upon completion.
- The Mark II Method was defined by Charles Symons in 1991.
- FP Mark II measures the size in FPs.
- The size is initially measured in unadjusted function points (UFPs) to which a technical complexity adjustment can then be applied (TCA).
- The assumption is that an information system comprises transactions which have the basic structure shown in figure,

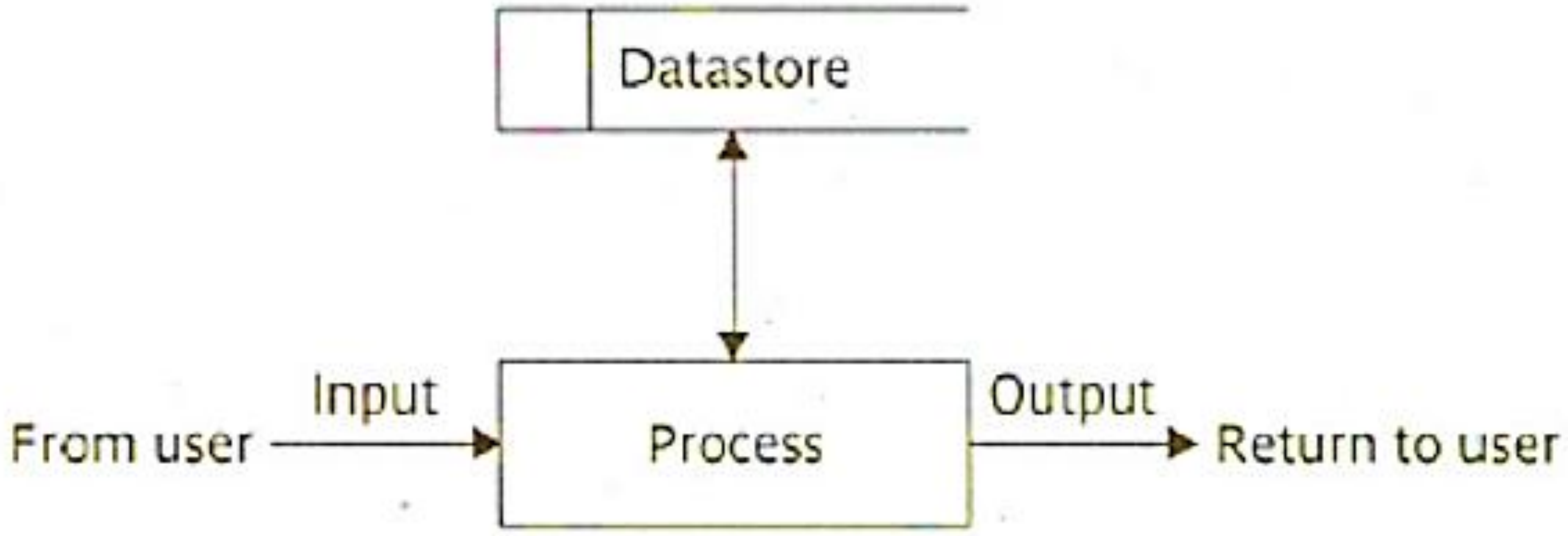


Fig: Model of a transaction

- For each transaction FPs are calculated as

FP = W_i *(number of input data element types) + W_e * (number of entity types referenced) + W_o * (number of output data element types)

- W_i, W_e, W_o are weightings derived by asking developers the proportions of effort spent in previous projects developing the code dealing with Inputs, accessing and modifying stored data and Processing outputs.
- The process for calculating weightings is time consuming and most FP counters use industry averages which are currently **0.58 for W_i, 1.66 for W_e and 0.26 for W_o.**
- Tables have been calculated to convert the FPs to lines of code for various languages. For example it is suggested that 53 lines of Java are needed on average to implement one FP, while for Visual C++ the figure is 34.

For Example:

A cash receipt transaction in the IOE maintenance accounts subsystem accesses two entity types – INVOICE and CASH-RECEIPT.

The data inputs are:

Invoice number

Date received

Cash received

If an INVOICE record is not found for the invoice number then an error message is issued. If the invoice number is found then a CASH-RECEIPT record is created. The error message is the only output of the transaction. The unadjusted function points, using the industry average weightings, for this transaction would therefore be: (0.58* 3) + (1.66 *2) + (0.26 *1) = 5.32

COSMIC Full Function Points (COSMIC FFP)

- The COSMIC (Common Software Measurement International Consortium) method is an international standard (ISO 19761) for sizing the functional requirements of any software.
- COSMIC function points are a unit of measure of software functional size. The size is a consistent measurement (or estimate) which is very useful for planning and managing software and related activities.
- The process of measuring software size is called functional size measurement (FSM).
- Function points are used to compute a functional size measurement (FSM) of software. The cost (in dollars or hours) of a single unit is calculated from past projects.
- In the context of the COSMIC FFP measurement method, which is aimed at measuring the functional size of software, only those functional user requirements allocated to software are considered.

- The COSMIC Function Point sizing method of measuring software requirements is based on two main principles:
  - **The ‘Software Context Model’**
  - **The ‘Generic Software Model’**

The ‘Software Context Model’

- Software is bounded by hardware and typically structured into **layers**.
- The **scope** of any piece of software to be measured shall depend on the **purpose** of the measurement and shall be confined wholly within a single layer.
- The **functional users** of a piece of software to be measured shall be identified from its Functional User Requirements (FUR) as the senders and/or intended recipients of data to/from the software respectively**.**
- A precise COSMIC size measurement of a piece of software requires that its FUR are known at a **level of granularity** at which its **functional processes** and sub-processes may be identified.

The ‘Generic Software Model’

- A piece of software interacts with its functional users across a **boundary**, and with **persistent storage** within the boundary.
- The FUR of a piece of software can be mapped into unique **functional processes**.
- Each functional process is started by its **triggering Entry** data movement. The data group moved by the triggering Entry is generated by a functional user in response to a **triggering event**.
- A functional process shall include at least one Entry data movement and either a Write or an Exit data movement. There is no upper limit to the number of data movements in a functional process**.**

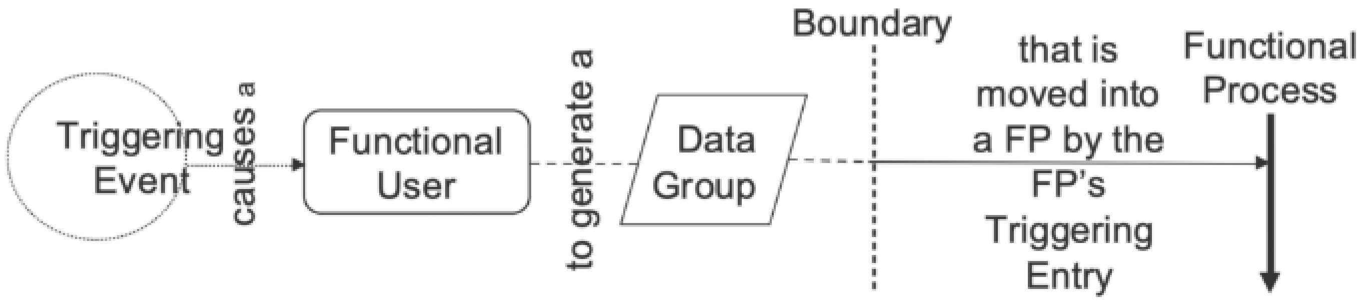


Fig: The relationship between triggering events, functional users and functional processes.

- Each functional process consists of sub-processes, **data movements (DMs)** and **data manipulations**.
- As an approximation for measurement purposes, the COSMIC method assumes that the functionality of any data manipulation is accounted for by the data movement with which it is associated.
- There are four data movement types, **Entry, Exit, Write and Read**.
- A data movement moves a single **data group**, which consists of a unique set of **data attributes** that describe a single **object of interest**.

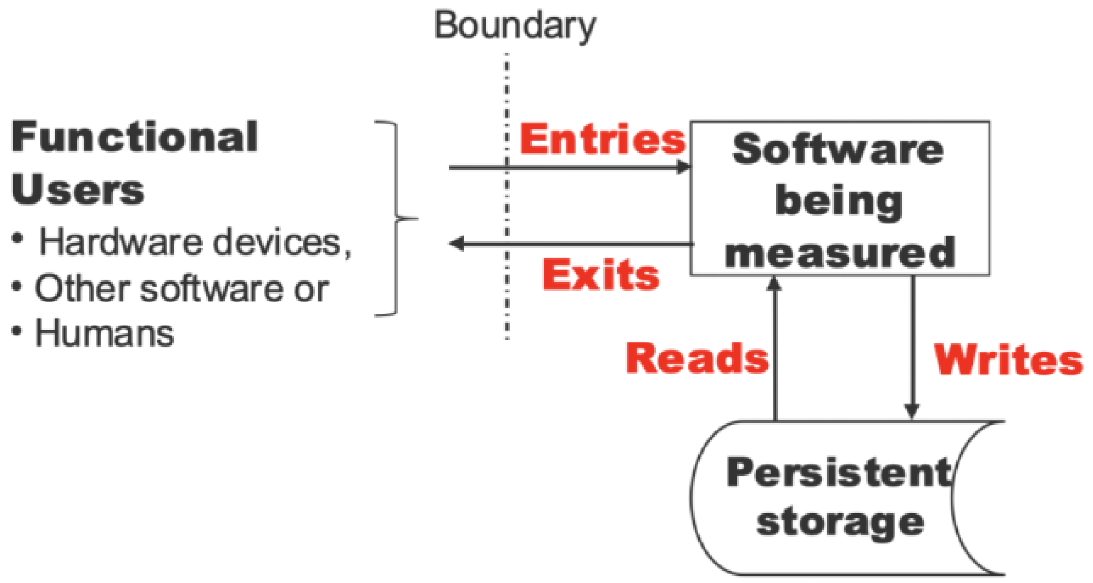


Fig: A generic model of a software and of its four types of data movements

Any software may have:

A) Three types of functional users of a software:

1. Hardware devices
2. Other software
3. Humans

B) Four types of data movements:

- **entries (E),** which are effected by subprocesses that move the data group into the software component in question from a ‘user’ outside its boundary – this could be from another layer or another separate software component in the same layer via peer-to-peer communication;
- **exits (X),** which are effected by subprocesses that move the data group from the software component to a ‘user’ outside its boundary;
- **reads (R),** which are data movements that move data groups from persistent storage (such as a database) into the software component;
- **writes (W),** which are data movements that transfer data groups from the software component into persistent storage.

C) Persistent storage:

- Any storage accessible by a software if it needs to store data or to retrieve stored data

- The COSMIC measurement principle is:

The functional size of a piece of software is equal to the number of its data movements A functional size is measured in units of ‘COSMIC Function Points’, abbreviated as ‘CFP’ where:

1 CFP is defined, by convention, as the size of a single data movement of a single data group

Then:

- The size of a functional process is equal to the number of its data movements types in CFP.
- The size of a piece of software is equal to the sum of the CFP sizes of its functional processes.

Example 1: When there is a single data movement of the 4 types in Fig. 1, the functional size is: 1 Entry + 1 Exit + 1 Read + 1 Write = 1 CFP + 1 CFP + 1 CFP + 1 CFP= 4 CFP.

Example 2: When there are two data movements of the 4 types in Fig. 1, the functional size is: 2 Entries + 2 Exits + 2 Reads + 2 Writes = 2 CFP + 2 CFP + 2 CFP + 2 CFP = 8 CFP

Cosmic measurement examples

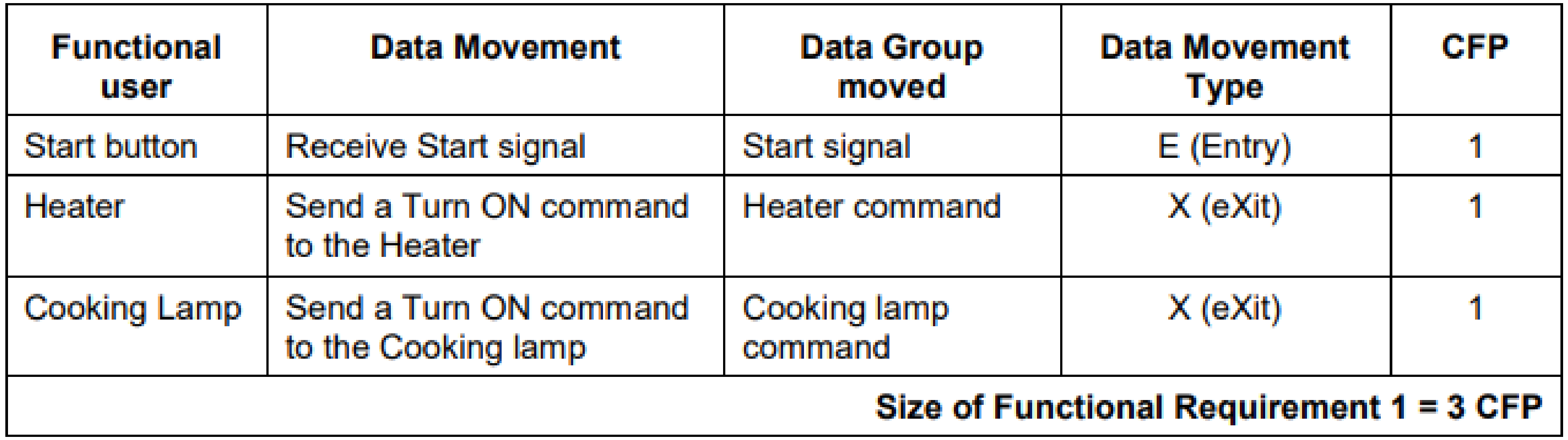
Example from the Rice Cooker case study Functional Requirement 1 – On receipt of the Start signal:

The software:

a) sends a ‘Turn ON’ signal to the Cooking Lamp

b) sends a ‘Turn ON’ signal to the heater.

This functional requirements is measured as follows with COSMIC:



Functional user	Data Movement	Data Group moved	Data Movement Type	CFP
Start button	Receive Start signal	Start signal	E (Entry)	1
Heater	Send a Turn ON command to the Heater	Heater command	X (eXit)	1
Cooking Lamp	Send a Turn ON command to the Cooking lamp	Cooking lamp command	X (eXit)	1
			Size of Functional Requirement 1 = 3 CFP	

Functional Requirement 2 - On receipt of a 30-second signal:

Starting at t = 0, and at each following 30-second interval, the software:

a) receives the 30-second signal;

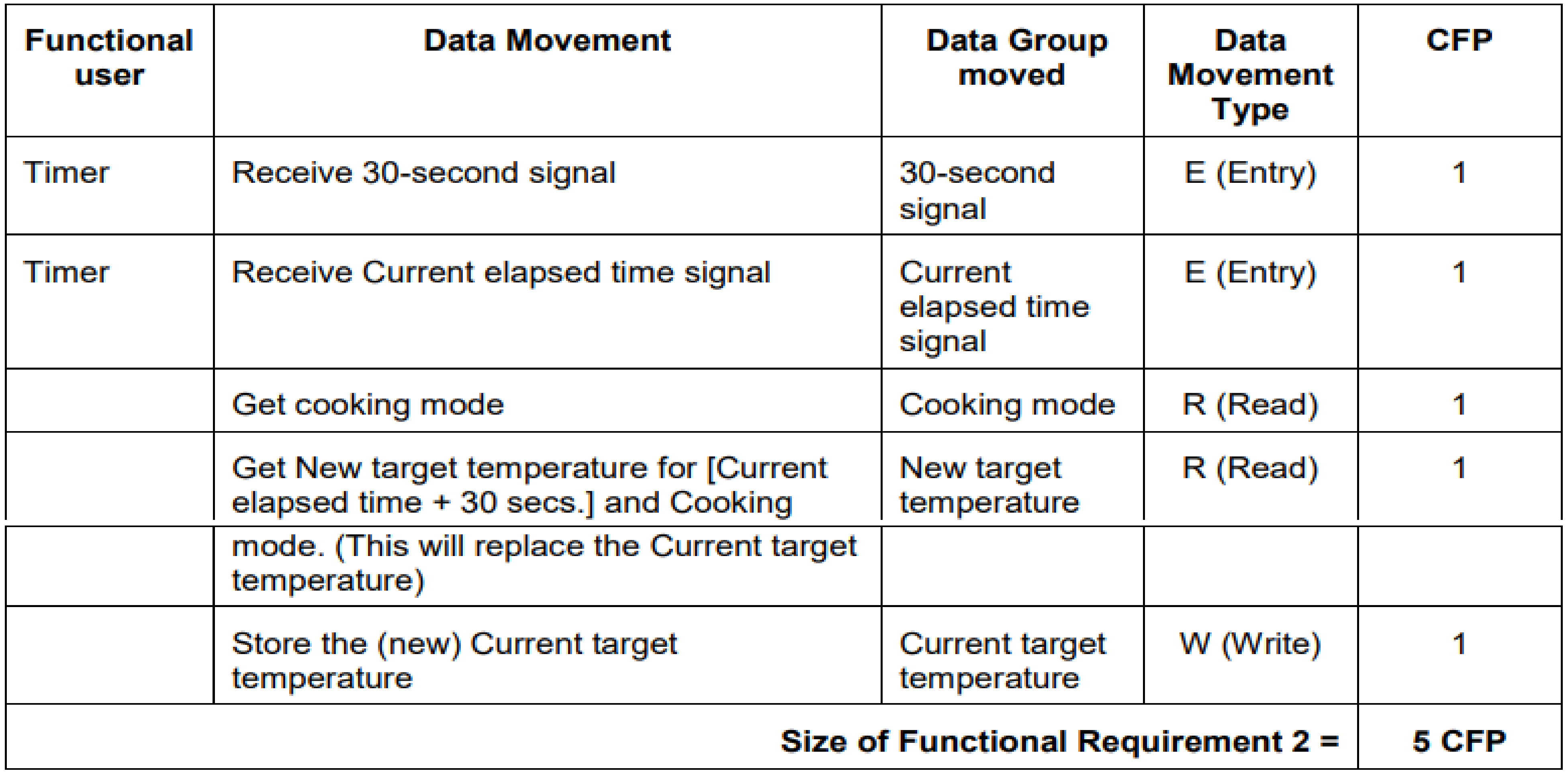
b) receives the elapsed time signal;

c) gets the cooking mode from persistent storage;

d) selects a new target temperature, for the cooking mode by reading it from the relationship in persistent storage at time = [current elapsed time + 30 seconds]. See in Figure ;

e) puts this new target temperature into persistent storage, which becomes the current target temperature until the next 30 sec. signal.

This functional requirement is measured as follows with COSMIC:

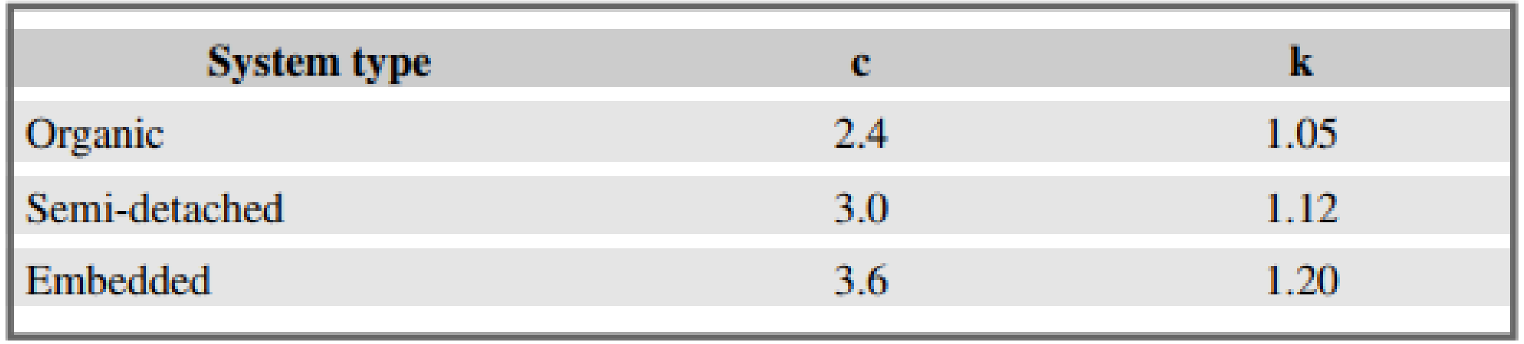


Functional user	Data Movement	Data Group moved	Data Movement Type	CFP
Timer	Receive 30-second signal	30-second signal	E (Entry)	1
Timer	Receive Current elapsed time signal	Current elapsed time signal	E (Entry)	1
	Get cooking mode	Cooking mode	R (Read)	1
	Get New target temperature for [Current elapsed time + 30 secs.] and Cooking mode. (This will replace the Current target temperature)	New target temperature	R (Read)	1
	Store the (new) Current target temperature	Current target temperature	W (Write)	1
			Size of Functional Requirement 2 =	**5 CFP**

COCOMO II: A Parametric Productivity Model

- COCOMO II is a parametric productivity model.
- Barry W. Boehm’s COCOMO (Constructive Cost Model) actually refers to a group of models.
- Boehm presented his first model (COCOMO81) in late 1970s based on a study of 63 projects.

Table. COCOMO81 constants



System type	c	k
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

- This basic model was built around the equation

$$\textbf{Effort} = \mathbf{c(size)^{k}}$$

- Effort was measured in *pm*, size was measured in *kdsi*, and *c* and *k* were constants.
- The first step was to derive an estimate of the system size in terms of *kdsi*. The constants *c* and *k* depend on whether the system could be classified as organic, semi-detached or embedded.

- **Organic mode** This would typically be the case when relatively small teams developed software in a highly familiar in-house environment and when the system being developed was small and the interface requirements were flexible. (small team, understood things, previous experience and knowledge)
- **Embedded mode** This meant that the product being developed had to operate within very tight constraints and changes to the system were very costly. (complex project, skill challenging , training is required, big team)
- **Semi-detached mode** This combined elements of the organic and the embedded modes or had characteristics that came between the two. (big team, little bit knowledge is required)

- Over years, Barry Boehm and his co-workers have refined a family of cost estimation models of which the latest one is COCOMO II (developed in 1995, published in 2000).
- It uses various multipliers and exponents, values of which have been set initially by experts.
- COCOMO II has been designed to accommodate this by having models for three different stages.
- **Application composition,** Here the external features of the system that the users will experience are designed. Prototyping will typically be employed to do this. With small applications that can be built using high-productivity application-building tools, development can stop at this point.
- **Early design** Here the fundamental software structures are designed. With larger, more demanding systems, where, for example, there will be large volumes of transactions and performance is important, careful attention will need to be paid to the architecture to be adopted.
- **Post architecture,** Here the software structures undergo final construction, modification and tuning to create a system that will perform as required.

- A database containing the performance details of executed projects has been built up and is periodically analyzed so that the expert judgements can be progressively replaced by the values derived from actual projects.
- Effort is calculated as

$$\mathbf{Pm = A\,(size)^{(sf)} * (em_1) * (em_2) * \ldots\ldots * (em_n)}$$

- A is a constant (2.94), size is measured in kdsi and sf is the exponent scale factor calculated as

$$\mathbf{sf = B + 0.01 * \sum (exponent\ driver\ ratings)}$$

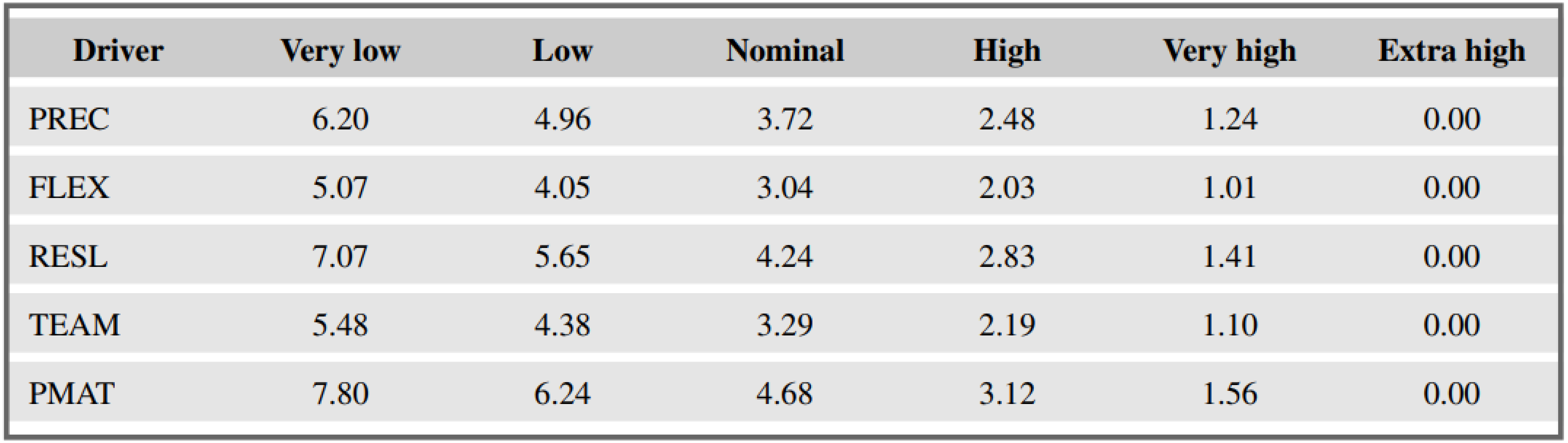
- B is a constant currently set at 0.91.

- The qualities that govern the exponent drivers are
- **Precedentedness (PREC):**This quality is the degree to which there are precedents or similar past cases for the current project. The greater the novelty of the new system, the more uncertainty there is and higher the value given to the exponent driver.
- **Development Flexibility (FLEX):** This reflects the number of different ways there are of meeting the requirements. The less flexibility there is, the higher the value of exponent driver.
- **Risk Resolution (RESL):** This reflects the degree of uncertainty about the requirements. If they are liable to change then a high value would be given to this exponent driver.
- **Team Cohesion (TEAM):** This reflects the degree to which there is a large dispersed team as opposed to there being a small tightly knit team.
- **Process Maturity (PMAT):** The more structured and organized the way the software is produced, the lower the uncertainty and lower the rating for this exponent driver.
- Each of the scale factors for a project is rated according to a range of judgements: very low, low, nominal, high, very high and extra high. There is a number related to each rating of the individual scale factors.

Example: A new project has ‘average’ novelty for the software supplier that is going to execute it and is thus given a nominal rating on this account for precedentedness. Development flexibility is high, but requirements may change radically and so the risk resolution exponent is rated very low. The development team are all located in the same office and this leads to team cohesion being rated as very high, but the software house as a whole tends to be very informal in its standards and procedures and the process maturity driver has therefore been given a rating of ‘low’.

1) What would be the scale factor (*sf*) in this case?
2) What would the estimate of effort if the size of the application was estimated as in the region of 2000 lines of code?

Table. COCOMO II Scale factor values



Driver	Very low	Low	Nominal	High	Very high	Extra high
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00
PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Solution:

The specific ratings for the drivers, based on your description, are:

i) The overall scale factor would be: sf=B+0.01*∑(exponent driver ratings),

sf= 0.91+0.01*(3.72+2.03+7.07 +1.10+6.24)=1.1116

ii) The estimate effort would be: Effort= A(size)^{sf} = 2.94*(2)$^{1.1116}$ =6.35 staff months(pm)

Driver	**Rating**	**Scale Factor Value**
Precedentedness	Nominal	3.72
Development Flexibility	High	2.03
Risk Resolution	Very Low	7.07
Team Cohesion	Very High	1.10
Process Maturity	Low	6.24

Cost Estimation

- Project cost can be obtained by multiplying the estimated effort (in man-month, from the effort estimate) with the manpower cost per month. Implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone.
- However, in addition to manpower cost, a project would incur several other types of costs which we shall refer to as the overhead costs. The overhead costs would include the costs of hardware and software required for the project and the company overheads for administration, office space , etc.

Example: Assume that the size of an organic type software product is estimated to be 32,000 lines of source code. Assume that the average salary of a software developer is \$2,000 per month. Determine the amount of staff cost to develop the product.

Solution:

From the basic COCOMO estimation formula

For organic software:

Effort = c (size)^k

Where, E = effort in person-months, Size(KLOC)= thousand lines of code (so for 32,000 lines, KLOC = 32), c and k are constants specific to the organic project type, typically c=2.4 and k=1.05 for an organic project.

Effort = c (size)^k =2.4*(32)$^{1.05}$ = 91.33 pm

Nominal development time=2.5*(91)$^{0.38}$ =14 months

Cost required=14 * \$2000=\$28,000

For Semi-detached:

Effort = c (size)^k

c=3.0 and k=1.12

Effort = c (size)^k =3.0*(32)$^{1.12}$ = 145.50 pm

Nominal development time= 2.5*(145.50)$^{0.35}$ =14 months

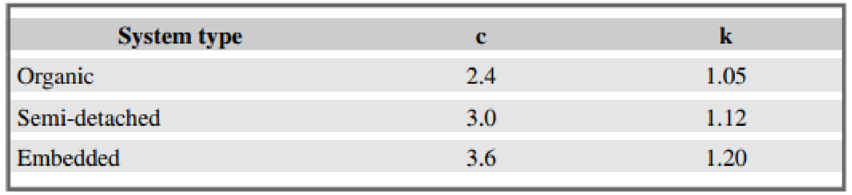
For Embedded:

Effort = c (size)^k

c=3.6 and k=1.20

Effort = c (size)^k =3.6*(32)$^{1.20}$ = 230.4 pm

Nominal development time= 2.5*(230.4)$^{0.32}$ =14 months



System type	c	k
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

Formulas;

Estimation of Development Effort:

Organic: Effort = 2.4 × (KLOC)$^{1.05}$ PM

Semi-detached: Effort = 3.0 × (KLOC)$^{1.12}$ PM

Embedded: Effort = 3.6 × (KLOC)$^{1.20}$ PM

Estimation of Development Time:

Organic: Tdev = 2.5 × (Effort)$^{0.38}$ Months

Semi-detached: Tdev = 2.5 × (Effort)$^{0.35}$ Months

Embedded: Tdev = 2.5 × (Effort)$^{0.32}$ Months