

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

GUIDELINES
for independent work on the course
«Software Economics»
for students of specialty 121 «Software engineering»

МЕТОДИЧНІ ВКАЗІВКИ
для самостійної роботи з курсу
«Економіка програмного забезпечення»
для студентів
спеціальності 121 «Інженерія програмного забезпечення»

Харків 2017

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
«ХАРКІВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

GUIDELINES
for independent work on the course
«Software Economics»
for students of specialty 121 «Software engineering»

МЕТОДИЧНІ ВКАЗІВКИ
для самостійної роботи з курсу
«Економіка програмного забезпечення»
для студентів
спеціальності 121 «Інженерія програмного забезпечення»

Затверджено
редакційно-видавничою
радою університету,
протокол № 3 від 22.12.2016 р.

Харків
НТУ «ХПІ»
2017

Методичні вказівки для самостійної роботи з курсу «Економіка програмного забезпечення» для студентів спеціальності 121 «Інженерія програмного забезпечення» / Уклад.: Вовк М.А., Чередніченко О.Ю., Орловський Д.Л., Копп А.М. – Харків: НТУ «ХПІ», 2017. – 36 с. – Англ. мовою.

Укладачі: М.А. Вовк, О.Ю. Чередніченко, Д.Л. Орловський, А.М. Копп

Рецензент проф. Шаронова Н.В.

Кафедра програмної інженерії та інформаційних технологій управління

CONTENTS

Introduction.....	4
1 Software Cost Estimation Methods	5
1.1 Non-algorithmic Methods	5
1.2 Algorithmic Methods.....	7
2 COCOMO Model	9
2.1 Basic COCOMO Model	10
2.2 Intermediate COCOMO Model.....	10
2.3 Complete COCOMO Model.....	12
3 Function Point Analysis.....	13
3.1 Five standard “functions”	16
3.2 Data Functions – Internal Logical Files (ILFs).....	17
3.3 Data Functions – External Interface Files (EIFs).....	19
3.4 Transaction Functions – External Inputs (EI’s)	20
3.5 Transaction Functions – External Outputs (EO’s)	21
3.6 Transaction Functions – External Inquiries (EQ’s).....	23
4 PERT Model	25
5 Project Schedule Development.....	28
5.1 Schedule Inputs.....	29
5.2 Scheduling Tools	29
5.3 Use of Project Stages.....	30
5.4 Project Review.....	31
5.5 Work Breakdown Structure (WBS)	32
6 Check questions	35
References.....	36

INTRODUCTION

Software economics is the study of how scarce project resources are allocated for software projects. Software economics helps software managers allocate those resources in the most efficient manner. The process of counting function points, gathering data, analyzing data is commonly referred to as Software Metrics, but in reality is a branch of economics which should be called Software Economics. This course provides an overview of business aspects surrounding software related solutions and investments, mastering the necessary knowledge of theoretical foundations of software economics, software issues that provide economic justification and planning of economic efficiency, complexity, costs related to the organization and conduct of the project, as well as providing practical skills to assess the economic characteristics of software projects.

These guidelines were designed for independent work on the course of “Software Economics”. It is especially useful for students, who is studying in English and foreign students. The objectives are to provide an empirical view of where the effort and money goes when we build large software systems and to suggest ways of reducing and controlling software costs.

1 SOFTWARE COST ESTIMATION METHODS

There are two major types of cost estimation methods: algorithmic and non-algorithmic. Algorithmic models vary widely in mathematical sophistication. Some are based on simple arithmetic formulas using such summary statistics as means and standard deviations. Others are based on regression models and differential equations. To improve the accuracy of algorithmic models, there is a need to adjust or calibrate the model to local circumstances. These models cannot be used off-the-shelf. Even with calibration the accuracy can be quite mixed.

We first give an overview of non-algorithmic methods.

1.1 Non-algorithmic Methods

Analogy costing. This method requires one or more completed projects that are similar to the new project and derives the estimation through reasoning by analogy using the actual costs of previous projects. Estimation by analogy can be done either at the total project level or at subsystem level. The total project level has the advantage that all cost components of the system will be considered while the subsystem level has the advantage of providing a more detailed assessment of the similarities and differences between the new project and the completed projects. The strength of this method is that the estimate is based on actual project experience. However, it is not clear to what extent the previous project is actually representative of the constraints, environment and functions to be performed by the new system.

Expert judgment. This method involves consulting one or more experts. The experts provide estimates using their own methods and experience. Expert-consensus mechanisms such as Delphi technique or PERT will be used to resolve the inconsistencies in the estimates. The Delphi technique works as follows:

- 1 The coordinator presents each expert with a specification and a form to record estimates.

2 Each expert fills in the form individually (without discussing with others) and is allowed to ask the coordinator questions.

3 The coordinator prepares a summary of all estimates from the experts (including mean or median) on a form requesting another iteration of the experts' estimates and the rationale for the estimates.

4 Repeat steps 2-3 as many rounds as appropriate.

A modification of the Delphi technique proposed by Boehm and Fahquhar seems to be more effective: Before the estimation, a group meeting involving the coordinator and experts is arranged to discuss the estimation issues. In step 3), the experts do not need to give any rationale for the estimates. Instead, after each round of estimation, the coordinator calls a meeting to have experts discussing those points where their estimates varied widely.

Parkinson. Using Parkinson's principle "work expands to fill the available volume", the cost is determined (not estimated) by the available resources rather than based on an objective assessment. If the software has to be delivered in 12 months and 5 people are available, the effort is estimated to be 60 person-months. Although it sometimes gives good estimation, this method is not recommended as it may provide very unrealistic estimates. Also, this method does not promote good software engineering practice.

Price-to-win. The software cost is estimated to be the best price to win the project. The estimation is based on the customer's budget instead of the software functionality. For example, if a reasonable estimation for a project costs 100 person-months but the customer can only afford 60 person-months, it is common that the estimator is asked to modify the estimation to fit 60 person-months' effort in order to win the project. This is again not a good practice since it is very likely to cause a bad delay of delivery or force the development team to work overtime.

Bottom-up. In this approach, each component of the software system is separately estimated and the results aggregated to produce an estimate for the overall system. The requirement for this approach is that an initial design must

be in place that indicates how the system is decomposed into different components.

Top-down. This approach is the opposite of the bottom-up method. An overall cost estimate for the system is derived from global properties, using either algorithmic or non-algorithmic methods. The total cost can then be split up among the various components. This approach is more suitable for cost estimation at the early stage.

1.2 Algorithmic methods

The algorithmic methods are based on mathematical models that produce cost estimate as a function of a number of variables, which are considered to be the major cost factors. Any algorithmic model has the form:

$$Effort = f(x_1, x_2, \dots, x_n),$$

where $\{x_1, x_2, \dots, x_n\}$ denote the cost factors. The existing algorithmic methods differ in two aspects: the selection of cost factors, and the form of the function f . We will first discuss the cost factors used in these models, and then characterize the models according to the form of the functions and whether the models are analytical or empirical.

Cost factors. Besides the software size, there are many other cost factors. The most comprehensive set of cost factors are proposed and used by Boehm in the COCOMO II model.

Linear models. Linear models have the form:

$$Effort = a_0 + \sum_{i=1}^n a_i x_i,$$

where the coefficients a_1, \dots, a_n are chosen to best fit the completed project data.

Multiplicative models. Multiplicative models have the form:

$$Effort = a_0 \prod_{i=1}^n a_i^{x_i}.$$

Again the coefficients a_1, \dots, a_n are chosen to best fit the completed project data.

Power function models. Power function models have the general form:

$$Effort = a \cdot S^b,$$

where S is the code-size, and a, b are (usually simple) functions of other cost factors. This class contains two of the most popular algorithmic models which are described at topic 2.

Model calibration using linear regression. A direct application of the above models does not take local circumstances into consideration.

However, one can adjust the cost factors using the local data and linear regression method. We illustrate this model calibration using the general power function model:

$$Effort = a \cdot S^b.$$

Take logarithm of both sides and let $Y = \log(Effort)$, $A = \log(a)$ and $X = \log S$. The formula is transformed into a linear equation:

$$Y = a + b \cdot X.$$

Applying the standard least square method to a set of previous project data $\{Y_b, X_i, i = 1, \dots, k\}$, we obtain the required parameters b and A (and thus a) for the power function.

2 COCOMO MODEL

COCOMO (Constructive Cost Estimation Model) was proposed by Boehm. According to him, any software development project can be classified into one of the following three categories based on the development complexity: organic, semidetached, and embedded. The classification is done considering the characteristics of the product as well as those of the development team and development environment. Usually these three product classes correspond to application, utility and system programs, respectively. Data processing programs are normally considered to be application programs. Compilers, linkers, etc., are utility programs. Operating systems and real-time system programs, etc. are system programs.

The definition of organic, semidetached, and embedded systems are elaborated below.

Organic. A development project can be considered of organic type, if the project deals with developing a well understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar types of projects.

Semidetached. A development project can be considered of semidetached type, if the development consists of a mixture of experienced and inexperienced staff. Team members may have limited experience on related systems but may be unfamiliar with some aspects of the system being developed.

Embedded. A development project is considered to be of embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational procedures exist.

According to Boehm, software cost estimation should be done through three stages: Basic COCOMO, Intermediate COCOMO, and Complete COCOMO.

2.1 Basic COCOMO Model

The basic COCOMO model gives an approximate estimate of the project parameters. The basic COCOMO estimation model is given by the following expressions:

$$Effort = a * (KLOC)^b PM,$$

$$Tdev = 2.5 * (Effort)^c Month,$$

where

– *KLOC* is the estimated size of the software product expressed in Kilo Lines of Code;

– *a, b, c* are constants for each category of software products;

– *Tdev* is the estimated time to develop the software, expressed in months;

– *Effort* is the total effort required to develop the software product, expressed in person months (PMs).

The effort estimation is expressed in units of person-months (PM).

The value of the constants *a, b, c* are given below (table 1):

Table 1 – The value of the constants *a, b, c*

Software project	a	b	c
Organic	2.4	1.05	0.38
Semi-detached	3.0	1.12	0.35
Embedded	3.6	1.20	0.32

2.2 Intermediate COCOMO Model

The basic COCOMO model assumes that effort and development time are functions of the product size alone. However, many other project parameters apart from the product size affect the development effort and time required for

the product. Therefore, in order to obtain an accurate estimation of the effort and project duration, the effect of all relevant parameters must be taken into account.

The intermediate COCOMO model recognizes this fact and refines the initial estimate obtained using the basic COCOMO expressions by using a set of 15 cost drivers (multipliers) based on various attributes of software development. For example, if modern programming practices are used, the initial estimates are scaled downward by multiplication with a cost driver having a value less than 1.

Each of the 15 attributes receives a rating on a six-point scale that ranges from “very low” to “extra high” (in importance or value) as shown below (table 2). An effort multiplier from the table below [*i*] applies to the rating. The product of all effort multipliers results in an Effort Adjustment Factor (EAF).

Table 2 – Cost drivers based on various attributes of software development

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
Required software reliability	0.75	0.88	1.00	1.15	1.40	
Size of application database		0.94	1.00	1.08	1.16	
Complexity of the product	0.70	0.85	1.00	1.15	1.30	1.65
Hardware attributes						
Run-time performance constraints			1.00	1.11	1.30	1.66
Memory constraints			1.00	1.06	1.21	1.56
Volatility of the virtual machine environment		0.87	1.00	1.15	1.30	
Required turnabout time		0.87	1.00	1.07	1.15	
Personnel attributes						

End of table 2

Cost Drivers	Ratings					
	Very Low	Low	Nominal	High	Very High	Extra High
Analyst capability	1.46	1.19	1.00	0.86	0.71	
Applications experience	1.29	1.13	1.00	0.91	0.82	
Software engineer capability	1.42	1.17	1.00	0.86	0.70	
Virtual machine experience	1.21	1.10	1.00	0.90		
Programming language experience	1.14	1.07	1.00	0.95		
Project attributes						
Application of software engineering methods	1.24	1.10	1.00	0.91	0.82	
Use of software tools	1.24	1.10	1.00	0.91	0.83	
Required development schedule	1.23	1.08	1.00	1.04	1.10	

EAF is used to refine the estimates obtained by basic COCOMO as follows:

$$Effort = Effort * EAF \quad Tdev = 2.5 * (Effort).$$

2.3 Complete COCOMO Model

Both the basic and intermediate COCOMO models consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems, each of them in turn could be of organic type, some semidetached, or embedded. The complete COCOMO model takes into account these differences in characteristics of the subsystems and estimates the effort and development time as the sum of the estimates for the individual subsystems. This approach reduces the percentage of error in the final estimate.

The following development project can be considered as an example application of the complete COCOMO model. A distributed Management Information System (MIS) product for an organization having offices at several places across the country can have the following sub-components:

- database part;
- graphical User Interface (GUI) part;
- communication part.

Of these, the communication part can be considered as embedded software. The database part could be semi-detached software, and the GUI part organic software. The costs for these three components can be estimated separately, and summed up to give the overall cost of the system.

3 FUNCTION POINT ANALYSIS

Function Points (FPs) were introduced by Alan Albrecht of IBM.

Objectives of Function Point Analysis.

There are three main objectives within the process of Function Point Analysis (FPA):

1 Measure software by quantifying the functionality requested by and provided to the customer.

2 Measure software development and maintenance independently of technology used for implementation.

3 Measure software development and maintenance consistently across all projects and organizations.

In working towards objectives 2 and 3 above, several organizations have created large repositories of FP counts that cross projects, technologies, and organizations. These repositories can be an invaluable tool for your first estimation efforts, because it lets you compare your project to similar projects that have been developed by other organizations around the world.

What is a “Function Point”? Function points are a standard unit of measure that represent the functional size of a software application. In the same

way that a house is measured by the square feet it provides, the size of an application can be measured by the number of function points it delivers to the users of the application.

The size of the application being measured is based on the user's view of the system. It is based on what the user asked for, not what is delivered. It's based on the way the user interacts with the system, including the screens that the user uses to enter input, and the reports the users receive as output. Finally, it's also based on their understanding of the data that needs to be stored and processed by the system.

FPA is technology-neutral. As a Certified Function Point Specialist (CFPS) it does not matter what technology you are using to implement your application. It doesn't matter if it's a Web application written in Java, PHP, ColdFusion, or .Net; or a client-server app written in Java, Delphi, VB; or even an AS/400 RPG application. Just show me your screens and your data tables and I'll derive "number of function points" from there.

Adding FPA to software development portfolio is also very easy. Historically, adding the process of counting FPs to development process results in a cost increase of only 1%.

Studies have shown that multiple function point counters can independently count the same application to within 10% accuracy of each other. Repeatability is very important, because without it we could not begin to trust the data from the hundreds of applications that are stored in repositories around the world.

This process works extremely well with use cases, and can even work with the concept of "stories" in Extreme Programming.

The "Unadjusted FP Count" of IFPUG v4.1 is now an ISO standard. In addition to being an ISO standard, FPs are used as the de facto standard for cost estimating applications like Cocomo II, Construx Estimate, and other estimating packages.

Certified Function Point Specialist, or CFPS. A CFPS is a person who has passed the official IFPUG certification test. The CFPS designation must be renewed every three years.

Because many companies have been using FP information for quite some time, there are several large repositories of project data, where companies have combined FP counts along with other information, such as tools used, man hours, and overall project cost. With accurate counts and other accurate data, you don't have to feel so along when making those all-important project estimates.

The following table shows a brief history of function points, beginning with the introduction of the concept by Alan Albrecht in 1979 (table 3).

Table 3 – A brief history of Function Point Analysis

1979	FPS introduced by Alan Albrecht
1984	First FP guidelines
1986	First IFPUG Board of Directors
1994	CPM Release 4.0
2003	ISO standard

The benefits of Function Point Analysis:

1 The ability to accurately estimate:

- project cost;
- project duration;
- project staffing size.

2 An understanding of other important metrics, such as:

- project defect rate;
- cost per FP;
- FP's per hour (what I refer to as "velocity");
- the productivity benefits of using new or different tools.

A simple five step counting process.

To start at a high level, there are five steps in the process of counting FPs. They are:

- 1 Determine the type of count.
- 2 Identify the scope and boundary of the count.
- 3 Determine the unadjusted FP count.
- 4 Determine the Value Adjustment Factor.
- 5 Calculate the Adjusted FP Count.

Introduction of steps 1, 2, 4, and 5 are given in the sample count, because they are most easily introduced by using an example. At this point it is important to get into the heart of step 3, because this is where the actual FP counting takes place. At this point FP practitioners look at a software application in terms of five standard functions.

3.1 Five standard “functions”

In counting FPs there are five standards “functions” that should be count. The first two of these are called Data Functions, and last three are called Transaction Functions. The names of these functions are listed below:

- 1 Data Functions:
 - internal logical files;
 - external interface files.
- 2 Transactional Functions:
 - external Inputs;
 - external Outputs;
 - external Inquiries.

Here is provided more detailed information and definitions on the five Data and Transactional Functions. Before getting into the details of the five functions there are several terms that you need to understand, because they’ll be used in each of the subsequent definitions. These are taken directly from the CPM.

Important terms and definitions used in describing the five functions.

User identifiable. This term refers to defined requirements for processes and/or groups of data that are agreed upon, and understood by, both the users and software developers.

Control information. This is data that influences and elementary process of the application being counted. It specifies what, when, or how data is to be processed.

Elementary process. An elementary process is the smallest unit of activity that is meaningful to the user. An elementary process must be self-contained and leave the business of the application being counted in a consistent state.

Data Element Type, or DET. A data element type is a unique, user recognizable, non-repeated field. This definition applies to both analyses of data functions and transactional functions.

Record Element Type, or RET. A record element type is a user recognizable subgroup of data elements within an Internal Logical File or External Interface File.

3.2 Data Functions – Internal Logical Files (ILFs)

ILF stands for “Internal Logical File”. ILFs represent data that is stored and maintained within the boundary of the application you are counting. When counting ILFs you are basically counting the data functions that your application is being built to maintain.

The more precise IFPUG definition of an ILF is: “An ILF is a user-identifiable group of logically related data or control information maintained within the boundary of the application. The primary intent of an ILF is to hold data maintained through one or more elementary processes of the application being counted.”

Furthermore, for data or control information to be counted as an ILF, both of the following IFPUG counting rules must also apply:

- 1 The group of data or control information is logical and user identifiable.
- 2 The group of data is maintained through an elementary process within the application boundary being counted.

Examples of ILFs. Samples of things that can be ILFs include:

1 Tables in a relational database.

2 Flat files.

3 Application control information, perhaps things like user preferences that are stored by the application.

4 LDAP data stores.

This isn't to say that all these things are ILFs, just that they can be. Function point counts resulting from ILFs.

When you're counting ILFs you will constantly be referring to the two tables that follow. The purpose of the first table is to help you determine whether the ILF you're currently looking at has a complexity level of Low (L), Average (A), or High (H). You do this by going into the table knowing the number of DETs and number of RETs you have counted in the ILF, and then finding the resulting Low, Average, or High value (table 4).

For instance, suppose I counted 5 DETs and 1 RET; that would be a Low complexity table. Conversely, if I had a table with 21 DETs and 2 RETs, that would be an Average complexity table.

Table 4 – The ILF complexity matrix

RETS	Data Element Types (DETs)		
	1-19		1-19
1	L	1	L
2 to 5	L	2 to 5	L
6 or more	A	6 or more	A

ILF under consideration has a complexity of Low, Average, or High, you'll come to this next table and determine the number of FPs that should be counted for this ILF (table 5). A Low complexity ILF is worth 7 points, an Average ILF is worth 10 points, and a High is worth 15.

Table 5 – ILF weights

Complexity	Points
Low	7
Average	10
High	15

3.3 Data Functions – External Interface Files (EIFs)

EIF stands for “External Interface File”. In my words, EIFs represent the data that your application will use/reference, but data that is not maintained by your application.

The official IFPUG definition of an EIF is: “An external interface file (EIF) is a user identifiable group of logically related data or control information referenced by the application, but maintained within the boundary of another application. The primary intent of an EIF is to hold data referenced through one or more elementary processes within the boundary of the application counted. This means an EIF counted for an application must be in an ILF in another application.”

Again, think of this as data that your application needs and uses, but does not maintain.

Examples of things that can be EIFs are identical to the list for ILFs, but again, the big difference here is that EIFs are not maintained by the application under consideration, but ILFs are. Function point counts resulting from EIFs.

Assigning an FP value to an EIF is the same as assigning one to an ILF. First, determine the number of DETs and RETs in the EIF, then do a lookup in the following table to determine whether the EIF has a complexity of Low, Average, or High (table 6).

Table 6 – EIF complexity matrix

	Data Element Types (DET)s		
RETS	1-19	RETS	1-19
1	L	1	L

End of table 6

	Data Element Types (DETs)		
RETS	1-19	RETS	1-19
2 to 5	L	2 to 5	L
6 or more	A	6 or more	A

Then, once you know whether the EIF is considered Low, Average, or High, look in the following table for the number of FPs to count for this particular EIF (table 7).

Table 7 – EIF weights

Value	No. or Function Points
Low	5
Average	7
High	10

The first two lookup tables are identical, but that more FPs are assigned to ILFs than EIFs.

3.4 Transaction Functions – External Inputs (EI’s)

EI stands for “External Input”. Here the official IFPUG definition of an EI is as follows: An external input (EI) is an elementary process that processes data or control information that comes from outside the application boundary. The primary intent of an EI is to maintain one or more ILFs and/or to alter the behavior of the system.

Examples of EIs include:

- 1 Data entry by users.
- 2 Data or file feeds by external applications.

Function point counts resulting from EI’s.

Allocating FPs to EIs is similar to the process we covered for ILFs and EIFs. However, in this case, instead of doing a lookup based on DET’s and

RET's to determine a Low/Average/High complexity, the lookup is performed using DET and FTR values. As you'll recall from the earlier definition, an FTR is a "file type referenced", so it can be either an ILF or an EIF.

As an example, suppose that you have a process that has 5 DET's, and during the processing it references an EIF named Users and an ILF named Process. You would go into the following table looking for the complexity of an EI that has 5 DET's and 2 FTR's. This EI is considered an "Average" complexity EI (table 8).

Table 8 – EI complexity matrix

FTR's	Data Element Types (DET's)		
	1-4		1-4
0-1	L	0-1	L
2	L	2	L
3 or more	A	3 or more	A

To carry our example forward, as you can see from the following table, an Average complexity EI is worth 4 FPs (table 9).

Table 9 – EI weights

Complexity	Points/Weight
Low	3
Average	4
High	6

3.5 Transaction Functions – External Outputs (EO's)

External Outputs are referred to as EO's. The IFPUG definition of an EO is as follows:

An external output (EO) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external output is to present information to a user through processing logic other

than, or in addition to, the retrieval of data or control information. The processing logic must contain at least one mathematical formula or calculation, create derived data maintain one or more ILFs or alter the behavior of the system.

EO examples include:

1 Reports created by the application being counted, where the reports include derived information.

2 Function point counts resulting from EO's.

Allocating FP's to EO's is very similar to the process for EI's. Again, you perform your lookup using DET's and FTR's, with a resulting Low/Average/High complexity.

As an example, suppose that you have a process that you've identified as being an EO, and it has 10 DET's and references two FTR's. You would go into the following table looking for the complexity of an EI that has 10 DET's and 2 FTR's. As you'll see from the table below, this EO is considered an "Average" complexity EO (table 10).

Table 10 – EO complexity matrix Weights

FTR	Data Element Types (DET)		
	1-5		1-5
0-1	L	0-1	L
2-3	L	2-3	L
4 or more	A	4 or more	A

To carry our example forward, using the table that follows, you'll see that an Average complexity EO has a value of 5 FPs (table 11).

Table 11 – EO weights

Complexity	Points/Weight
Low	4
Average	5

End of table 11

Complexity	Points/Weight
High	7

3.6 Transaction Functions – External Inquiries (EQ’s)

The last transactional function is referred to as an EQ, or External Inquiry. The IFPUG definition of an EQ is as follows:

An external inquiry (EQ) is an elementary process that sends data or control information outside the application boundary. The primary intent of an external inquiry is to present information to a user through the retrieval of data or control information from an ILF or EIF. The processing logic contains no mathematical formulas or calculations, and creates no derived data. No ILF is maintained during the processing, nor is the behavior of the system altered.

Examples of EQs include:

1 Reports created by the application being counted, where the report does not include any derived data.

2 Other things known as “implied inquiries”, which unfortunately, are a little out of scope for this paper.

Function point counts resulting from EQ’s

Allocating an FP count to EQs is very similar to the process for EIs and EOs. Again, you perform your lookup using DETs and FTRs, with a resulting Low/Average/High complexity.

As an example, suppose that you have a process that you’ve identified as being an EQ, and it has 20 DETs and references 4 FTRs. You would go into the following table looking for the complexity of an EI that has 20 DETs and 4 FTRs (table 12). As you’ll see from the table below, this EQ is considered a “High” complexity EQ.

Table 12 – EQ complexity matrix Weights

FTRs	Data Element Types (DET)	
	1-5	1-5

End of table 12

FTRs	Data Element Types (DETs)		
0-1	L	0-1	L
2-3	L	2-3	L
4 or more	A	4 or more	A

Carrying our EQ example forward, you'll find from the table below that a High complexity EQ is worth 6 FPs (table 13).

Table 13 – EQ weights

Complexity	Points/Weight
Low	3
Average	4
High	6

Taken together, these two data functions (ILFs and EIFs) and three transactional functions (EIs, EOs, EQs) represent the five functions that are counted in a FP count.

4 PERT MODEL

Program (Project) Evaluation and Review Technique (PERT)(G) is a project management tool used to schedule, organize, and coordinate tasks within a project. It is basically a method to analyze the tasks involved in completing a given project, especially the time needed to complete each task, and to identify the minimum time needed to complete the total project. PERT planning involves the following steps:

- identify the specific activities and milestones;
- determine the proper sequence of the activities;
- construct a network diagram. v Estimate the time required for each activity;
- determine the critical path;
- update the PERT chart as the project progresses.

The main objective of PERT is to facilitate decision making and to reduce both the time and cost required to complete a project.

PERT is intended for very large-scale, one-time, non-routine, complex projects with a high degree of intertask dependency, projects which require a series of activities, some of which must be performed sequentially and others that can be performed in parallel with other activities.

PERT planning involves the following steps that are described below.

1 Identify the specific activities and milestones. The activities are the tasks required to complete a project. The milestones are the events marking the beginning and the end of one or more activities. It is helpful to list the tasks in a table that in later steps can be expanded to include information on sequence and duration.

2 Determine the proper sequence of the activities. This step may be combined with the activity identification step since the activity sequence is evident for some tasks. Other tasks may require more analysis to determine the exact order in which they must be performed.

3 Construct a network diagram. Using the activity sequence information, a network diagram can be drawn showing the sequence of the serial and parallel activities. Each activity represents a node in the network, and the arrows represent the relation between activities. Software packages simplify this step by automatically converting tabular activity information into a network diagram.

4 Estimate the time required for each activity. Weeks are a commonly used unit of time for activity completion, but any consistent unit of time can be used. A distinguishing feature of PERT is its ability to deal with uncertainty in activity completion time. For each activity, the model usually includes three time estimates:

- optimistic time – generally the shortest time in which the activity can be completed. It is common practice to specify optimistic time to be three standard deviations from the mean so that there is a approximately a 1% chance that the activity will be completed within the optimistic time;

- most likely time – the completion time having the highest probability. Note that this time is different from the expected time.

- pessimistic time – the longest time that an activity might require.

Three standard deviations from the mean is commonly used for the pessimistic time. PERT assumes a beta probability distribution for the time estimates. For a beta distribution, the expected time for each activity can be approximated using the following weighted average:

- expected time = $(\text{Optimistic} + 4 \times \text{Most likely} + \text{Pessimistic})/6$;

- this expected time may be displayed on the network diagram;

- to calculate the variance for each activity completion time, if three standard deviation times were selected for the optimistic and pessimistic times, then there are six standard deviations between them, so the variance is given by: $[(\text{Pessimistic} - \text{Optimistic}) / 6]^2$.

5 Determine the critical path. The critical path is determined by adding the times for the activities in each sequence and determining the longest path in the project. The critical path determines the total calendar time required for the project. If activities outside the critical path speed up or slow down (within

limits), the total project time does not change. The amount of time that a non – critical path activity can be delayed without the project is referred to as a slack time. If the critical path is not immediately obvious, it may be helpful to determine the following four quantities for each activity: ES – Earliest Start time EF - Earliest Finish time LS – Latest Start time LF - Latest Finish time These times are calculated using the expected time for the relevant activities. The earliest start and finish times of each activity are determined by working forward through the network and determining the earliest time at which an activity can start and finish considering its predecessors activities. The latest start and finish times are the latest times that an activity can start and finish without delaying the project. LS and LF are found by working backward through the network. The difference in the latest and earliest finish of each activity is that activity's slack. The critical path then is the path through the network in which none of the activities have slack. The variance in the project completion time can be calculated by summing the variances in the completion times of the activities in the critical path. Given this variance, one can calculate the probability that the project will be completed by the certain date assuming a normal probability distribution for the critical path. The normal distribution assumption holds if the number of activities in the path is large enough for the central limit theorem to be applied. Since the critical path determines the completion date of the project, the project can be accelerated by adding the resources required to decrease the time for the activities in the critical path. Such a shortening of the project sometimes is referred to as project crashing. Update the PERT chart as the project progresses. Make adjustments in the PERT chart as the project progresses. As the project unfolds, the estimated times can be replaced with actual times. In cases where there are delays, additional resources may be needed to stay on schedule and the PERT chart may be modified to reflect the new situation.

Benefits. PERT is useful because it provides the following information:

- expected project completion time;
- probability of completion before a specified date;

- the critical path activities that directly impact the completion time;
- the activities that have slack time and that can be lend resources to critical path activities;
- activity start and end date.

5 PROJECT SCHEDULE DEVELOPMENT

Essentially, driving without any idea of how you're going to get there is the same as working on a project without a schedule. No matter the size or scope of your project, the schedule is a key part of project management. The schedule tells you when each activity should be done, what has already been completed, and the sequence in which things need to be finished.

Luckily, drivers have fairly accurate tools they can use. Scheduling, on the other hand, is not an exact process. It's part estimation, part prediction, and part 'educated guessing.

Because of the uncertainty involved, the schedule is reviewed regularly, and it is often revised while the project is in progress. It continues to develop as the project moves forward, changes arise, risks come and go, and new risks are identified. The schedule essentially transforms the project from a vision to a time-based plan.

Schedules also help you do the following:

- they provide a basis for you to monitor and control project activities;
- they help you determine how best to allocate resources so you can achieve the project goal;
- they help you assess how time delays will impact the project;
- you can figure out where excess resources are available to allocate to other projects;
- they provide a basis to help you track project progress.

With that in mind, what's the best way of building an accurate and effective schedule for your next project?

Project managers have a variety of tools to develop a project schedule – from the relatively simple process of action planning for small projects, to use of Gantt Charts and Network Analysis for large projects. Here, we outline the key tools you will need for schedule development.

5.1 Schedule Inputs

You need several types of inputs to create a project schedule:

- personal and project calendars – understanding working days, shifts, and resource availability is critical to completing a project schedule;

- description of project scope – from this, you can determine key start and end dates, major assumptions behind the plan, and key constraints and restrictions. You can also include stakeholder expectations, which will often determine project milestones;

- project risks – you need to understand these to make sure there's enough extra time to deal with identified risks – and with unidentified risks (risks are identified with thorough Risk Analysis);

- lists of activities and resource requirements – again, it's important to determine if there are other constraints to consider when developing the schedule. Understanding the resource capabilities and experience you have available – as well as company holidays and staff vacations – will affect the schedule.

A project manager should be aware of deadlines and resource availability issues that may make the schedule less flexible.

5.2 Scheduling Tools

Here are some tools and techniques for combining these inputs to develop the schedule:

- 1 Schedule Network Analysis – this is a graphic representation of the project's activities, the time it takes to complete them, and the sequence in which they must be done. Project management software is typically used to create these analyses – Gantt charts and PERT Charts are common formats;

2 Critical Path Analysis – this is the process of looking at all of the activities that must be completed, and calculating the “best line” – or critical path – to take so that you’ll complete the project in the minimum amount of time. The method calculates the earliest and latest possible start and finish times for project activities, and it estimates the dependencies among them to create a schedule of critical activities and dates. Learn more about Critical Path Analysis.

3 Schedule Compression – this tool helps shorten the total duration of a project by decreasing the time allotted for certain activities. It’s done so that you can meet time constraints, and still keep the original scope of the project.

You can use two methods here:

1 Crashing – this is where you assign more resources to an activity, thus decreasing the time it takes to complete it. This is based on the assumption that the time you save will offset the added resource costs.

2 Fast-Tracking – this involves rearranging activities to allow more parallel work. This means that things you would normally do one after another are now done at the same time. However, do bear in mind that this approach increases the risk that you’ll miss things, or fail to address changes.

5.3 Use of Project Stages

One of the biggest reasons that projects over-run is that the “final” polishing and error-correction takes very much longer than anticipated. In this way, projects can seem to be “80% complete” for 80% of the time! What’s worse, these projects can seem to be on schedule until, all of a sudden, they over-run radically.

A good way of avoiding this is to schedule projects in distinct stages, where final quality, finished components are delivered at the end of each stage. This way, quality problems can be identified early on, and rectified before they seriously threaten the project schedule.

5.4 Project Review

Once you have outlined the basic schedule, you need to review it to make sure that the timing for each activity is aligned with the necessary resources. Here are tools commonly used to do this:

- what if scenario analysis – this method compares and measures the effects of different scenarios on a project. You use simulations to determine the effects of various adverse, or harmful, assumptions – such as resources not being available on time, or delays in other areas of the project. You can then measure and plan for the risks posed in these scenarios;

- resource leveling – here, you rearrange the sequence of activities to address the possibility of unavailable resources, and to make sure that excessive demand is not put on resources at any point in time. If resources are available only in limited quantities, then you change the timing of activities so that the most critical activities have enough resources;

- critical chain method – this also addresses resource availability. You plan activities using their latest possible start and finish dates. This adds extra time between activities, which you can then use to manage work disruptions;

- risk multipliers – risk is inevitable, so you need to prepare for its impact. Adding extra time to high-risk activities is one strategy. Another is to add a time multiplier to certain tasks or certain resources to offset overly optimistic time estimation.

After the initial schedule has been reviewed, and adjustments made, it's a good idea to have other members of the team review it as well. Include people who will be doing the work – their insights and assumptions are likely to be particularly accurate and relevant.

Key Points. Scheduling aims to predict the future, and it has to consider many uncertainties and assumptions. As a result, many people believe it's more of an art than a science.

But whether you're planning a team retreat, or leading a multimillion-dollar IT project, the schedule is a critical part of your efforts. It identifies and

organizes project tasks into a sequence of events that create the project management plan.

A variety of inputs and tools are used in the scheduling process, all of which are designed to help you understand your resources, your constraints, and your risks. The end result is a plan that links events in the best way to complete the project efficiently.

5.5 Work Breakdown Structure (WBS)

The building blocks of a schedule start with a Work Breakdown Structure (WBS). The WBS is a hierarchical reflection of all the work in the project in terms of deliverables. In order to produce these deliverables, work must be performed.

A typical approach in developing a WBS is to start at the highest level, with the product of the project. For example, you are assigned as the project manager of a New Product Development project. The new product you are developing is a new toy for children age's five through nine. The objective of this product development project is to increase the revenue of the organization by ten percent.

Example of WBS. Underneath is an example of a WBS for this new toy (figure 1). Each level of the WBS is a level of detail created by decomposition. Decomposition is the process of breaking down the work into smaller, more manageable components. The elements at the lowest level of the WBS are called tasks. In the example above, brochures, advertising and commercials are all work packages or tasks.

Marketing collateral is on a summary level called a control account in project management parlance. In Project Insight, project management software, control accounts are called "summary tasks". Summary tasks are roll ups of the tasks underneath them.

The decomposition of a schedule will continue at varying rates. "Brochures" is a task identified at the fourth level of decomposition, while the "marketing plan" is also a task, but defined at the third level of decomposition.

As a project manager, the level of decomposition will be dependent on the extent to which you will need to manage. Project Insight supports as many levels of hierarchy as are needed. The expectation is that each task will have a single owner and the owner is expected to manage and report on the work necessary to deliver the task. In Project Insight, this is called the “task owner”. If you cannot assign a single owner, or you need to have additional visibility into the progress of that task, additional decomposition is recommended.

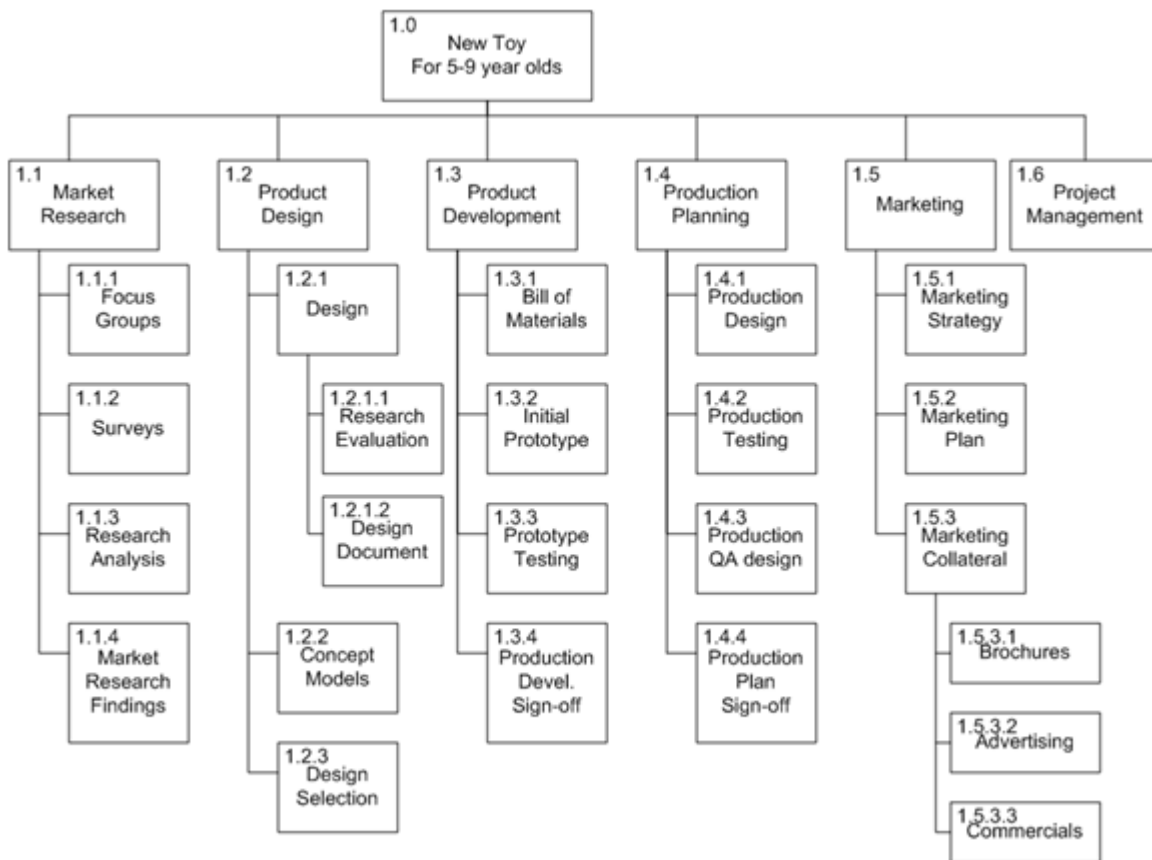


Figure 1 – Example of WBS

Once all the deliverables of the project have been identified, tasks will be performed in order to create the deliverables. In some cases, these activities are the physical deliverables, but in other cases they are the actions that need to be performed. A physical deliverable, for example, might be an image (an actual file) that is needed for the brochure. Listing out each of the tasks to be performed will result in an activity list as demonstrated below (figure 2).

The work package “focus group” actually consists of three (3) separate tasks – “identify focus group targets”, “prepare focus group objectives” and “perform focus group”. The work package “surveys”, on the other hand, is not broken down into tasks. In our example, it may have been determined that the task owner that is performing the surveys does not need to report on any of the details of that task. As stated earlier, decomposition will continue to the level that is necessary to effectively manage the project.

Work Package	WBSID	Activity	Predecessor	Duration in Weeks	Resource Type
Focus Group	1.1.1.1	Identify Focus Group Targets			
Focus Group	1.1.1.2	Prepare Focus Group Objectives			
Focus Group	1.1.1.3	Perform Focus Group			
Surveys	1.1.2	Perform Survey			
Research Analysis	1.1.3	Perform Analysis			
Market Research Findings	1.1.4	Create Market Research Findings			
Research Evaluation	1.2.1.1.1	Review Market Research Findings			
Research Evaluation	1.2.1.1.2	Develop Design Options			
Research Evaluation	1.2.1.1.3	Present Design Options			
Design Document	1.2.1.2.1	Draft Design Document			
Design Document	1.2.1.2.2	Design Document Review			
Design Document	1.2.1.2.3	Final Design Document			
Concept Models	1.2.2	Develop Concept Model			
Design Selection	1.2.3	Review Concepts			
Bill of Materials	1.3.1	Create Initial Bill of Materials			
Initial Prototype	1.3.2.1	Develop Initial Prototype			
Initial Prototype	1.3.2.2	Revise Initial Prototype			
Prototype Testing	1.3.3	Test Prototype			
Production Design	1.4.1	Design Production Process			
Production Testing	1.4.2	Design Production Testing Process			
Production QA design	1.4.3	Design Quality Assurance Tests			
Marketing Strategy	1.5.1	Develop Marketing Strategy			
Marketing Plan	1.5.2.1	Develop Initial Marketing Plan			
Marketing Plan	1.5.2.2	Final Marketing Plan			
Brochures	1.5.3.1	Create Brochures			
Advertising	1.5.3.2	Create Ads			
Commercials	1.5.3.3	Create Commercials			
Production Plan Sign-off	1.4.4	Production Plan Sign-off			
Production Devel. Sign-off	1.3.4	Production Devel. Sign-off			
Project Management	1.6	Project Management Activities		LOE	

Figure 2 – Product Development Activity List

6 CHECK QUESTIONS

- 1 What is the evaluation of a software project?
- 2 What are the adjustment factors in the metric of the number of function pointers?
- 3 Describe the steps of project evaluation based on LOC- and FP-metrics. What is the difference between the most accurate approach and the most precise?
- 4 Explain the PERT model.
- 5 What is the problem of assigning an model of the early stage of software design?
- 6 What is a metric?
- 7 What is a constructive cost model? What is it used for?
- 8 What is a functional pointer?
- 9 How to calculate the number of functional pointers?
- 10 What are the complexity adjustment coefficients in the metric of functional indicators?
- 11 Describe the basic equation of the early software design model.
- 12 Describe the scale factors of the model COCOMO II. How to assess the scale factors?
- 13 How is the development time determined in the COCOMO II model?
- 14 What are post-architecture model cost factors and how are they calculated?
- 15 Describe the recommended rule for allocating project costs.

REFERENCES

1 IFPUG Counting Practices Committee et al. Function Point Counting Practices Manual. – 2004.

2 Boehm, B. Software cost estimation with COCOMO II / B. Boehm, R. Madachy, B. Steece. – NJ. : Prentice-Hall, 2000.

3 Guide to the Software Engineering Body of Knowledge (SWEBOOK). – IEEE, 2004.

4 Вендров А. М. Практикум по проектированию программного обеспечения экономических информационных систем : учебное пособие / А. М. Вендров. – М. : Финансы и статистика, 2002.

5 Горевая М. И. Экономическая эффективность проектных решений : учебное пособие / М. И. Горевая, Г. А. Клочков, Г. И. Курчеева. – Новосибирск : НГТУ, 2008.

6 Архипенков С. Лекции по управлению программными проектами / С. Архипенков. – М., 2009.

7 Sachan R. K. Optimizing Basic COCOMO Model Using Simplified Genetic Algorithm / R. K. Sachan, A. Nigam, A. Singh, S. Singh, D. S. Kushwaha // Procedia Computer Science. – 2016. – Vol. 89. – P. 492–498.

8 Fairley R. E. D. The influence of COCOMO on software engineering education and training / R. E. D. Fairley // Journal of Systems and Software. – 2007. – Vol. 80. – N. 8. – P. 1201-1208.

Навчальне видання

**Методичні вказівки для самостійної роботи з курсу
«Економіка програмного забезпечення»
для студентів спеціальності
121 «Інженерія програмного забезпечення»**

Англійською мовою

Укладачі:

**Вовк Марина Анатоліївна
Чередніченко Ольга Юріївна
Орловський Дмитро Леонідович
Копп Андрій Михайлович**

Відповідальний за випуск проф. Годлевський М.Д.

Роботу до друку рекомендував проф. Горілий О.В.

В авторській редакції

План 2017 р., поз. 6

Підписано до друку 06.10.2017. Формат 60 × 84/16. Папір офсетн. № 2.

Друк – ризографія. Гарнітура New Roman Times. Ум. друк. арк. 2,1.

Наклад 50 прим. Зам. № . Ціна договірна.

Видавничий центр НТУ «ХП», 61002, , Харків, вул. Фрунзе, 21
Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.

Друкарня НТУ «ХП»
61002, Харків, вул.. Фрунзе, 21