Appendix: Technical Report of Property models for the automotive domain – Version 1.0

Severine Sentilles, Efi Papatheocharous, Federico Ciccozzi and Kai Petersen
Mälardalen University, Swedish Institute of Computer Science, Blekinge Institute of Technology

Draft
2016-02-18

Abstract

This technical report lists the property models that are important for the automotive domain, including property models for Development Effort, Development Time, Cost and Performance. For each of these property models, evaluation methods are described in detail.

The property models can be used to make estimates of a property of a given system (e.g., performance). Several property models can be used for the estimation of a property. Properties are described based on: property ID, name, data format and documentation (description of the property, purpose, intended use, etc.)

A property may be assessed through the use of evaluation methods. Evaluation methods are listed in this report underneath their property models. However, evaluation methods can be related with several property models. Evaluation methods are described based on: method ID, name, output, unit, applicability, parameter(s), driver(s), formula, description, available implementation.

Combinations of evaluations may also be used by another evaluation method(s), or even experts, to evaluate for example, trade-offs between alternatives in architectural decision-making. A more concrete example is to decide among several components that have various origins: they might be developed (e.g., in-house), reused, bought (e.g., COTS, sub-contracted) or obtained (e.g., open source), which is the optimal one to use, in terms of the property of interest.

Table of Contents

A. Property model for Development Effort

DevelopmentEffortPropertyModel = <"Development Effort", {"Basic COCOMO 1", "Intermediate COCOMO 1", "Advanced, Detailed COCOMO 1", "Application composition COCOMO II", "Early design COCOMO II", "Reuse COCOMO II", "Post-architecture COCOMO II", "Expert estimation", "Estimation by Analogy", "Weighted Micro Function Points"}>

Development Effort

PropID: Development Effort

Data_format: Number (unit: Person-month)

Documentation:

Development effort is "*the total number of person-months logged by the development team in all the stages of product development, starting from initial design through final product acceptance testing*" [Harter 2000]. For example, if a product requires 6 person-months for its development, this means that 1 person will need 6 months to develop product. This is equivalent to 2 persons for 3 months or 3 persons for 2 months, etc.

Reference:

- [Harter 2000] Donald E. Harter, Mayuram S. Krishnan, Sandra A. Slaughter, (2000) Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development. Management Science 46(4): 451-466. http://dx.doi.org/10.1287/mnsc.46.4.451.12056

A.1. Evaluation methods for Development Effort

A.1.1. Basic COCOMO 1

Evaluation method Name: Basic COCOMO 1
Measure: Development Effort
Unit: Person-month

Applicability: familiar projects, ambitious projects, tightly constrained/complex projects

Parameters:

- KLOC is a measure of the size of a computer program. The size is determined by measuring the number of lines of source code a program has. High-level languages such as C++, will compile into more lines of machine code than an assembly language, which is a low-level language.

Drivers:

- Development mode (organic, semi-detached, embedded)
- Application domain
- Project characteristics: Size, Innovation, Deadline, Dev. Environment

Formula: Development Effort = $a(KLOC)^b$
with *a, b* two constants which are set based on the mode of development according to the following table:

Basic COCOMO	a	b
Organic	2.4	1.05
Semi-detached	3.0	1.12
Embedded	3.6	1.20

Description:
The Basic COCOMO 1 model is a single-valued, static model that computes software development effort as a function of program size expressed in estimated kilo-line of code [Pressman 1993].

The model relies on the characteristics of the project, which allows deciding upon one of the three following development modes:

- *Organic*: for relatively small teams developing software in a highly familiar, in-house environment.
- *Semi-detached*: when the team members have some experience related to some aspects of the system under development but not others and the team is composed of experienced and inexperienced people.
- *Embedded*: if the project must operate within a strongly coupled complex of hardware, software, regulations, and operational procedures, such as real-time systems.

These modes range from the familiar to the ambitious, tightly constrained development projects.

Another way to describe the development mode is through the following project characteristics:

Development Mode	Project Characteristics			
	Size	Innovation	Deadline/constraints	Dev. Environment
Organic	Small	Little	Not tight	Stable
Semi-detached	Medium	Medium	Medium	Medium
Embedded	Large	Greater	Tight	Complex hardware/ customer interfaces

Table from [Merlo-Schett et al. 2002-03].

- **Model assumptions**
  - KLOC can be somehow estimated at the point the estimate is needed
  - There are 152 hours per person-month. According to the organization this value may differ from the standard by 10% to 20% [Merlo-Schett et al. 2002-03].

- **Advantages**
  - Transparent, one can see how it works.
  - Good for quick, early, rough order of low magnitude estimate of software cost.
  - Drivers are particularly helpful to the estimator to understand the impact of different factors that affect project costs.
  - Granularity is low which is consistent to the granularity of the information available to support the estimation.

- **Disadvantages**
  - Limited accuracy (does not take into account factors known to significantly affect the development effort).
  - Hard to estimate accurately KLOC early on in the project, when effort estimates are required, actually KLOC is a length measure and not a size measure.
  - Vulnerable to the quality of tuning of the model based on the needs of the organization and classifications of development mode.

- **Sources:**

- [Pressman 1993] Pressman, Roger S. A Manager's Guide to Software Engineering. New York: McGraw-Hill, 1993.
- [Boehm 1981] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981
- [Merlo-Schett et al. 2002-03] Merlo-Schett N., Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of Zurich, Switzerland. 2002-2003. https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf

Available Implementation: None.

A.1.2. Intermediate COCOMO 1

Evaluation Method Name: Intermediate COCOMO 1
Measure: Development Effort
Unit: Person-month

Applicability: **f**amiliar projects, ambitious projects, tightly constrained/complex projects, known requirements, systems and sub-systems

Parameters:
- KLOC
- EAF based on the Product (RELY, DATA, CPLX), Computer (TIME, STOR, VIRT, TURN), Personnel (ACAP, AEXP, PCAP, VEXP, LEXP), Project (MODP, TOOL, SCED) characteristics

Drivers:
- Development mode (organic, semi-detached, embedded) based on the Project characteristics: Size, Innovation, Deadline, Development Environment

Formula: Development Effort = $a(KLOC)^b*(EAF)$
with *a, b* two constants which are set based on the mode of development according to the following table:

Intermediate COCOMO	a	b
Organic	3.2	1.05
Semi-detached	3.0	1.12
Embedded	2.8	1.20

and *EAF* the effort adjustment factor based on 15 cost drivers.

Description:

*See Basic COCOMO 1 for the description of organic, semi-detached, embedded.

Cost Driver	Description	Rating					
		Very Low	Low	Nominal	High	Very High	Extra High
Product							
RELY	Required software reliability	0.75	0.88	1.00	1.15	1.40	-
DATA	Database size	-	0.94	1.00	1.08	1.16	-
CPLX	Product complexity	0.70	0.85	1.00	1.15	1.30	1.65
Computer							
TIME	Execution time constraint	-	-	1.00	1.11	1.30	1.66
STOR	Main storage constraint	-	-	1.00	1.06	1.21	1.56
VIRT	Virtual machine volatility	-	0.87	1.00	1.15	1.30	-
TURN	Computer turnaround time	-	0.87	1.00	1.07	1.15	-
Personnel							
ACAP	Analyst capability	1.46	1.19	1.00	0.86	0.71	-
AEXP	Applications experience	1.29	1.13	1.00	0.91	0.82	-
PCAP	Programmer capability	1.42	1.17	1.00	0.86	0.70	-
VEXP	Virtual machine experience	1.21	1.10	1.00	0.90	-	-
LEXP	Language experience	1.14	1.07	1.00	0.95	-	-
Project							
MODP	Modern programming practices	1.24	1.10	1.00	0.91	0.82	-
TOOL	Software Tools	1.24	1.10	1.00	0.91	0.83	-
SCED	Development Schedule	1.23	1.08	1.00	1.04	1.10	-

The Effort Adjustment Factor (EAF) is the product of the effort multipliers corresponding to each of the cost drivers for the project.

- **Model assumptions**
  - There are 152 hours per person-month [Merlo-Schett et al. 2002-03].
  - The ratings of the cost drivers are applicable to the project for which estimates are needed.

- **Advantages**
  - Good to use when the requirements have been specified.
  - Commonly used model.
  - Open / accessible model.
  - Granularity is higher but it needs to be consistent to the granularity of the information available to support the estimation.
  - Available implementation.

- **Disadvantages**
  - Extremely vulnerable to mis-classification of the development mode.
  - Success depends largely on tuning the model to the needs of the organization, using historical data which is not always available.
  - KLOC measures system length not size.

- **Sources:**
  - [Boehm 1981] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981
  - [Merlo-Schett et al. 2002-03] Merlo-Schett N, Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of Zurich, Switzerland. 2002-2003. https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf

Available Implementation:
COCOMO® 81 Intermediate Model Implementation
http://sunset.usc.edu/research/COCOMOII/cocomo81_pgm/cocomo81.html

A.1.3. Advanced, Detailed COCOMO 1

Evaluation Method Name: Advanced, Detailed COCOMO 1
Measure: Development Effort
Unit: person-month

Applicability: **f**amiliar projects, ambitious projects, tightly constrained/complex projects, systems and sub-systems, product design phase, detailed-design phase, code and unit test phase, integration and test phase, at the component level

Parameters:

- Program size

Drivers:

- Cost drivers weighted according to each phase of the software lifecycle

Formula: --

Description: A major shortcoming of both the Basic and Intermediate COCOMO 1 models is that they consider a software product as a single homogeneous entity. However, most large systems are made up several smaller sub-systems. These sub-systems may have widely different characteristics.

The Detailed COCOMO Model differs from the Intermediate COCOMO model in that it uses effort multipliers for each phase of the project. These phase dependent effort multipliers yield better estimates because the cost driver ratings may be different during each phase.
In Advanced COCOMO Model the cost of each subsystem is estimated separately. This approach reduces the margin of error in the final estimate.

The Advanced COCOMO Model computes effort as a function of program size and a set of cost drivers weighted according to each phase of the software lifecycle. The Advanced models applies the Intermediate model at the component level, and then a phased-based approach is used to consolidate the estimate [Fenton, 1997].

The four phases used in the Detailed COCOMO model are: requirements planning and product design (RPD), detailed design (DD), code and unit test (CUT), and integration and test (IT). Each cost driver is broken down by phases as in the example shown in the table below for the Analyst Capability (ACAP). Estimates for each module are combined into subsystems and eventually an overall project estimate. Using the detailed cost drivers, an estimate is determined for each phase of the lifecycle.

Cost Driver	Rating	RPD	DD	CUT	IT
ACAP	Very Low	1.80	1.35	1.35	1.50
	Low	0.85	0.85	0.85	1.20
	Nominal	1.00	1.00	1.00	1.00
	High	0.75	0.90	0.90	0.85
	Very High	0.55	0.75	0.75	0.70

Table from [Merlo-Schett 2002].

- **Model assumptions**
  - Program size can be estimated in good-enough accuracy.
  - The model can be also used to estimate effort of maintenance. Maintenance includes small updates and repairs during the operational life of a system. Most parameters are used, but some like SCED, RELY, MODP are not applicable.

- **Advantages**
  - Applied at the component level
  - Reduces the margin of error in the final estimate.
  - Adds separate cost drivers for each phase and allows for a more flexible phase distribution
  - Tied-up to a quality model [Boehm 1978] and [ISO/IEC 9126-1].

- **Disadvantages**
  - Requires much more detailed cost drivers and detailed information needs to be available to support the estimation.
  - Ratings of cost drivers could not be found in the original material [Boehm, 1981]
  - Formula is not found.
  - No implementation is available.

- **Sources:**
  - [Fenton and Pfleeger 1997] Fenton, N.E. and Pfleeger, S.L. (1997). Software Metrics: A Rigorous and Practical Approach International Thomson Computer Press.
  - [Boehm 1981] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981
  - [Merlo-Schett et al. 2002-03] Merlo-Schett N, Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of Zurich, Switzerland. 2002-2003. https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf
  - [ISO/IEC 9126-1] ISO/IEC 9126-1, Software engineering – product quality – Part 1: Quality Model, first ed.: 2001-06-15.

- [Boehm 1978] Boehm, B. W., Brown, J. R., Kaspar, H., Lipow, M., McLeod, G., and Merritt, M., Characteristics of Software Quality, North Holland, 1978, Boehm, Barry W.

Available Implementation: N/A

Introduction to COCOMO II

The model COCOMO II consists of a composition of the three previous models. It is tuned to estimate projects of life cycle practices of the 1990's and 2000's. The full COCOMO II model includes three stages. Stage 1 supports estimation of prototyping or applications composition efforts. Stage 2 supports estimation in the Early Design stage of a project, when less is known about the project's cost drivers. Stage 3 supports estimation in the Post-Architecture stage of a project.
COCOMO II is based on the general formula [USC 2000]:

$$PM = \prod_{i=1}^{17} (EM_i) \cdot A \cdot \left[\left(1 + \frac{REVL}{100}\right) \cdot Size\right]^{\left(0.91 + 0.01 \sum_{j=1}^{5} SF_j\right)} + \left(\frac{ASLOC \cdot \left(\frac{AT}{100}\right)}{ATPROD}\right)$$

where

$$Size = KNSLOC + \left[KASLOC \cdot \left(\frac{100 - AT}{100}\right) \cdot \frac{(AA + SU + 0.4 \cdot DM + 0.3 \cdot CM + 0.3 \cdot IM)}{100}\right]$$

$$B = 0.91 + 0.01 \sum_{j=1}^{5} SF$$

Estimate effort with:

Symbol	Description
A	Constant, currently calibrated as 2.45
AA	Assessment and assimilation
ADAPT	Percentage of components adapted (represents the effort required in understanding software)
AT	Percentage of components that are automatically translated
ATPROD	Automatic translation productivity
REVL	Breakage: Percentage of code thrown away due to requirements volatility
CM	Percentage of code modified
DM	Percentage of design modified
EM	Effort Multipliers: RELY, DATA, CPLX, RUSE, DOCU, TIME, STOR, PVOL, ACAP, PCAP, PCON, APEX, PLEX, LTEX, TOOL, SITE
IM	Percentage of integration and test modified
KASLOC	Size of the adapted component expressed in thousands of adapted source lines of code
KNSLOC	Size of component expressed in thousands of new source lines of code
PM	Person Months of estimated effort
SF	Scale Factors: PREC, FLEX, RESL, TEAM, PMAT
SU	Software understanding (zero if DM = 0 and CM = 0)

Formula and Table from [USC 2000]

A.1.4. Application composition COCOMO II

Evaluation Method Name: Application composition COCOMO II
Measure: Development Effort
Unit: Person-month

Applicability: Early, prototyping and design phase (application composition), familiar applications to be composed from interoperable components (e.g., GUI builders, database or object managers, middleware, hypermedia handlers)

Parameters:
- Number of application points
- Application point productivity

Drivers:
- Complexity of object points

Formula: PM = (NAP * (1-%reuse/100)) / PROD
where NAP is the number of application points, **%reuse i**s the percentage of screens, reports, and 3GL modules reused from previous applications, pro-rated by degree of reuse, and PROD is the application point productivity. **PROD** (application point productivity) is estimated as NOP/person-month. **NOP** refers to New Object Points (Object Point count adjusted for reuse). In NOP (number of object points) the use of the term "object" in "Object Points" defines screens, reports, and 3GL modules as objects.

Description:
The model supports estimation of prototyping or applications composition efforts. It uses Object Points Counting Procedure (as described in [USC 2000]).

- **Model assumptions**
  - Estimates can be made after the requirements have been agreed.
  - To estimate the number of application points (i.e., the number of screens, reports, 3GL components) that the application will comprise of, assumptions on their standard definition needs to be assumed that it applies in the specific context.

- **Advantages**
  - Reflects more up-to-date software practices than COCOMO 1.

  - Size measurement is not based on code length but on object points and complexity.
  - Flexible/tailorable for estimation for Application Generator.

- **Disadvantages**
  - Requires expertise in estimating application or object points.
  - Vulnerable to mis-estimations due to subjective classification of object instances into simple, medium and difficult complexity levels.
  - Assumes typical format of applications, based on clients, servers, screens, reports and 3GL components, which might be considered outdated.

- **Sources:**
  - [Sommerville 2010] Sommerville, I., Software Engineering, 9th ed., Addison Wesley, 2010
  - [USC 2000] USC COCOMO II manual, http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf
  - http://csse.usc.edu/csse/affiliate/private/COCOMOII_2000/Book_Draft/K_chapter_5_991223_v6+CR.doc

Available Implementation: COCOMO II - Constructive Cost Model
http://csse.usc.edu/tools/COCOMOII.php

A.1.5. Early design COCOMO II

Evaluation Method Name: Early design COCOMO II
Measure: Development Effort
Unit: person-month

Applicability: Early design phase, familiar projects, application generation with more granular information

Parameters:
- Number of function points (FP). FP measure a software project by quantifying the information processing functionality associated with major external data input, output, or file types.
- A is the Effort Coefficient (= 2.94 calibrated for COCOMO II 2000)
- EM are the effort multipliers (7 are used for the Early design COCOMO II)

Drivers:
- Scale factors: the following are used PREC, FLEX, RESL, TEAM, PMAT.

$$PM = A * Size^{B} * \prod_{i}^{p} EM_i$$

Formula: PM = $a(FP)^b*(EM)$ OR more general

where PM are person-months, FP number of function points and EM effort multipliers, *a, b* two constants, a is the multiplicative constant and b is the scale factors.

$$B = \beta_0 + \sum_{i=1}^{5} \beta_i SF_i$$

The five Scale Factors are:

1. PREC Precedentedness (how novel the project is for the organization)
2. FLEX Development Flexibility
3. RESL Architecture / Risk Resolution
4. TEAM Team Cohesion
5. PMAT Process Maturity

Scale Factors (W_i)		annotation
PREC	If a product is similar to several previously developed project, then the precedentedness is high	Describe much the same influences that the original Development Mode did, largely intrinsic to a project and uncontrollable
FLEX	Conformance needs with requirements / external interface specifications, ...	
RESL	Combines Design Thoroughness and Risk Elimination (two scale factors in Ada).	Identify management controllables by which projects can reduce diseconomies of scale by reducing sources of project turbulence, entropy and rework.
TEAM	accounts for the sources of project turbulence and entropy because of difficulties in synchronizing the project's stakeholders.	
PMAT	time for rating: project start. Two ways for rating: 1. by the results of an organized evaluation based on the SEI CMM, 2. 18 Key Process Areas in the SEI CMM.	

Table 4: scale factors description for COCOMO II

Scale factors from [Merlo-Schett et al. 2002-03]

Description:

Model supports estimation in the Early Design stage of a project, when less is known about the project's cost drivers. The same approach is used for the Post Architecture COCOMO II.

This model is used in the early stages of a software project when very little may be known about the size of the product to be developed, the nature of the target platform, the nature of the personnel to be involved in the project, or the detailed specifics of the process to be used. This model could be employed in either Application Generator, System Integration, or Infrastructure development sectors.

It uses Function points which measures a software project by quantifying the information processing functionality associated with major external data input, output, or file types. Five user function types should be identified as defined in the table below.

External Input (Inputs)	Count each unique user data or user control input type that (i) enters the external boundary of the software system being measured and (ii) adds or changes data in a logical internal file.
External Output (Outputs)	Count each unique user data or control output type that leaves the external boundary of the software system being measured.
Internal Logical File (Files)	Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.
External Interface Files (Interfaces)	Files passed or shared between software systems should be counted as external interface file types within each system.
External Inquiry (Queries)	Count each unique input-output combination, where an input causes and generates an immediate output, as an external inquiry type.

Table from [USC 2000]

The function point cost estimation approach is based on the amount of functionality in a software project and a set of individual project factors [Behrens 1983] [Kunkler 1985] [IFPUG 1994]. Function points are useful estimators since they are based on information that is available early in the project life cycle.

Each instance of these function types is then classified by complexity level. The complexity levels determine a set of weights, which are applied to their corresponding function counts to determine the Unadjusted Function Points quantity. This is the Function Point sizing metric used by COCOMO II. The usual Function Point procedure involves assessing the degree of influence (DI) of fourteen application characteristics on the software project determined according to a rating scale of 0.0 to 0.05 for each characteristic. The 14 ratings are added together, and added to a base level of 0.65 to produce a general characteristics adjustment factor that ranges from 0.65 to 1.35. Each of these fourteen characteristics, such as distributed functions, performance, and reusability, thus have a maximum of 5% contribution to estimated effort. This is inconsistent with COCOMO experience; thus COCOMO II uses Unadjusted Function Points for sizing, and applies its reuse factors, cost driver effort multipliers, and exponent scale factors to this sizing quantity.

Table 2. COCOMO II cost driver values

Cost 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
RELY	0.82	0.92	1.00	1.10	1.26	
DATA		0.90	1.00	1.14	1.28	
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	
TIME			1.00	1.11	1.29	1.63
STOR			1.00	1.05	1.17	1.46
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.42	1.19	1.00	0.85	0.71	
PCAP	1.34	1.15	1.00	0.88	0.76	
PCON	1.29	1.12	1.00	0.90	0.81	
AEXP	1.22	1.10	1.00	0.88	0.81	
PEXP	1.19	1.09	1.00	0.91	0.85	
LTEX	1.20	1.09	1.00	0.91	0.84	
TOOL	1.17	1.09	1.00	0.90	0.78	
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	

For Early design a reduced set of multiplicative cost drivers is used shown in the Table below. The models is used to make rough estimates of the project's cost and duration before its entire architecture is determined.

	Early Design cost drivers	Post-Architecture cost drivers (Counterpart combined)
Product reliability and complexity	RCPX	RELY, DATA, CPLX, DOCU
Required reuse	RUSE	RUSE
Platform difficulty	PDIF	TIME, STOR, PVOL
Personnel capability	PERS	ACAP, PCAP, PCON
Personnel experience	PREX	AEXP, PEXP, LTEX
Facilities	FCIL	TOOL, SITE
Required Development Schedule	SCED	SCED

Early design and Post-Architecture cost drivers from [Merlo-Schett et al. 2002-03]

- **Model assumptions**
  - Rates assumed from COCOMO II
- **Advantages**

- A reduced set of multiplicative cost drivers is used (for example instead of RELY, DATA, CPLX, DOCU the driver RCPX is used)
- Early cost drivers are obtained by combining the Post-Architecture model cost drivers
- It can be used without collecting much information
- It can be used early, before the entire architecture is determined

- **Disadvantages**
  - No available implementation.

- **Sources:**
  - [Merlo-Schett et al. 2002-03] Merlo-Schett N, Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of Zurich, Switzerland. 2002-2003. https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf
  - http://csse.usc.edu/csse/affiliate/private/COCOMOII_2000/Book_Draft/K_chapter_5_991223_v6+CR.doc
  - [Behrens 1983] Behrens, C. (1983), "Measuring the Productivity of Computer Systems Development Activities with Function Points," IEEE Transactions on Software Engineering, 1983.
  - [Kunkler 1985] Kunkler, J. (1983), "A Cooperative Industry Study on Software Development/Maintenance Productivity," Xerox Corporation, Xerox Square --- XRX2 52A, Rochester, NY 14644, Third Report, March 1985.
  - [IFPUG 1994] IFPUG (1994), IFPUG Function Point Counting Practices: Manual Release 4.0, International Function Point Users' Group, Westerville, OH.

Available Implementation: N/A

A.1.6. Reuse COCOMO II

Evaluation Method Name: Reuse COCOMO II
Measure: Development Effort
Unit: person-month

Applicability: projects based on reuse (black-box or white-box)

Parameters:
- Number of LOC reused or generated

Drivers:

Formula: **PM = (ASLOC *AT/100) / ATPROD**
for the black-box reuse; where ASLOC is the number of lines of generated code, AT is the percentage of code automatically generated and ATPROD is the productivity of engineers in integrating this code.

ESLOC = ASLOC *(1-AT/100) * AAM
for the white-box reuse; where ASLOC is the number of lines of generated code, AT is the percentage of code automatically generated and AAM is the adaptation adjustment multiplier computed from the sum of the costs of changing the reused code, the costs of understanding how to integrate the code and the costs of reuse decision making. The latter two costs range from 50 for complex unstructured code to 10 for well-written object oriented code and from 0 to 8 depending on the amount of analysis effort required.

Description:
COCOMO is not only capable of estimating the cost and schedule for a development started from "scratch", but it is also able to estimate the cost and schedule for products that are built upon already existing code. Adaptation considerations have also been incorporated into COCOMO, where an estimate for KSLOC will be calculated. This value will be substituted in place of the SLOC found in the equations already discussed. This adaptation of code utilizes an additional set of equations that are used to calculate the final count on source instructions and related cost and schedule. These equations use the following values as components:

- Adapted Source Lines of Code (ASLOC). The number of source lines of code adapted from existing software used in developing the new product.
- Percent of Design Modification (DM). The percentage of the adapted software's design that received modification to fulfill the objectives and environment of the new product.
- Percent of Code Modification (CM). The percentage of the adapted software's code that receives modification to fulfill the objectives and environment of the new product.
- Percent of Integration Required for Modified Software (IM). The percentage of effort needed for integrating and testing of the adapted software in order to combine it into the new product.
- Percentage of reuse effort due to Software Understanding (SU).
- Percentage of reuse effort due to Assessment and Assimilation (AA).
- Programmer Unfamiliarity with Software (UNFM)

The AAF is the adaptation adjustment factor. The AAF is the calculated degree to which the adapted software will affect overall development.

$$PMtotal = (SCED) \times PMnominal \times \prod_{i=1}^{18} EMi$$

- **Model assumptions**

- Estimates and percentages of reuse can be made.

- **Advantages**
  - Allows adaptations of COCOMO in the case code is reused, which is very common way of development.
- **Disadvantages**
  - No available implementation.

- **Sources:**
  - [USC 2000] USC COCOMO II manual, http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf

Available Implementation: N/A

A.1.7. Post-architecture COCOMO II

Evaluation Method Name: Post-architecture COCOMO II
Measure: Development Effort
Unit: Person-month

Applicability: projects that a software life-cycle architecture has been developed, system Integration, or infrastructure developments

Parameters:
- Number of lines of source code
- A is the Effort Coefficient = 2.94 for COCOMO II
- EM are the effort multipliers (17 are used for the Post-architecture COCOMO II)

Drivers:

Formula: $$PM = A * Size^{B} * \prod_{i}^{p} EM_i$$

Description: Model supports estimation in the Post-Architecture stage of a project. It uses COCOMO.II formulas. It uses for sizing the formulas shown below, and applies a set of 17 cost drivers grouped into 4 categories (Product factors, Platform factors, Personnel factors, Project factors). These four categories are parallel the four categories of COCOMO 1. Many of the seventeen factors of COCOMO II are similar to the fifteen factors of COCOMO 1. It

is used after project's overall architecture is developed. This stage proceeds most cost -effectively if a software life-cycle architecture has been developed, validated with respect to the systems mission, concept of operation and risk.

$$\text{Size} = \left(1 + \frac{\text{REVL}}{100}\right) \times (\text{New KSLOC} + \text{Equivalent KSLOC})$$

$$\text{Equivalent KSLOC} = \text{Adapted KSLOC} \times \left(1 - \frac{\text{AT}}{100}\right) \times \text{AAM}$$

$$\text{where AAM} = \begin{cases} \dfrac{\text{AA} + \text{AAF} \times (1 + [0.02 \times \text{SU} \times \text{UNFM}])}{100}, & \text{for AAF} \le 50 \\ \dfrac{\text{AA} + \text{AAF} + (\text{SU} \times \text{UNFM})}{100}, & \text{for AAF} > 50 \end{cases}$$

$$\text{AAF} = (0.4 \times \text{DM}) + (0.3 \times \text{CM}) + (0.3 \times \text{IM})$$

Sizing equations from [Merlo-Schett et al. 2002-03].

Table 2. COCOMO II cost driver values

Cost 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
RELY	0.82	0.92	1.00	1.10	1.26	
DATA		0.90	1.00	1.14	1.28	
CPLX	0.73	0.87	1.00	1.17	1.34	1.74
RUSE		0.95	1.00	1.07	1.15	1.24
DOCU	0.81	0.91	1.00	1.11	1.23	
TIME			1.00	1.11	1.29	1.63
STOR			1.00	1.05	1.17	1.46
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.42	1.19	1.00	0.85	0.71	
PCAP	1.34	1.15	1.00	0.88	0.76	
PCON	1.29	1.12	1.00	0.90	0.81	
AEXP	1.22	1.10	1.00	0.88	0.81	
PEXP	1.19	1.09	1.00	0.91	0.85	
LTEX	1.20	1.09	1.00	0.91	0.84	
TOOL	1.17	1.09	1.00	0.90	0.78	
SITE	1.22	1.09	1.00	0.93	0.86	0.80
SCED	1.43	1.14	1.00	1.00	1.00	

- **Model assumptions**
  - Adjusts for software reuse and reengineering where automated tools are used for translation of existing software

- **Advantages**
  - Accounts for requirements volatility in the estimates.
  - The most detailed model.

  - Applicable to multiple sizing methods.
  - Tailorable mix of the Application Composition model (for early prototyping efforts) and two increasingly detailed estimation models for subsequent portions of the life cycle, Early Design and Post-Architecture.

- **Disadvantages**
  - Requires a lot of effort to produce estimates as the level of granularity of the information needed is high.
  - No implementation available.

- **Sources:**
  - [USC 2000] USC COCOMO II manual, http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf
  - [Merlo-Schett et al. 2002-03] Merlo-Schett N, Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of Zurich, Switzerland. 2002-2003. https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf

Available Implementation: None.

A.1.8. Expert estimation

Evaluation Method Name: Expert estimation
Measure: Development Effort
Unit: person-month

Applicability: Similar projects, expertise and experience, to any case

Parameters:
- Experts value estimate
- Experts confidence, experience, skill
- Experts weights
- Expert's background
- Available checklist of information related to the project

Drivers: Expert's experience and knowledge, gut-feeling.

Formula: N/A

Description:
In expert estimation, the estimation work is conducted by a person recognized as an expert on the task, and that a significant part of the estimation process is based on non-explicit and non-recoverable reasoning process, i.e. “intuition” [Jørgensen 2004]. Expert estimation is the dominant strategy when estimating software development effort.

Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached [Sommerville 2010].

It has been demonstrated in [Jørgensen 2004] that in some cases expert estimates are more accurate than other types of estimation methods. This is the case when the methods used are not properly calibrated, or for cases in which having specific knowledge is important (e.g. knowing who works on which part of the project).

According to [Jørgensen 2004], the following 12 principles can improve the accuracy of expert estimations:

- evaluate estimation accuracy but avoid high evaluation pressure
- avoid conflicting estimation goals (e.g. “bid”, “planned”, “most likely” and “wishful thinking” vs “realism)
- ask the estimators to justify and criticize their estimates
- avoid irrelevant and unreliable estimation information
- use documented data from previous development tasks
- find estimation experts with relevant domain background and good estimation records
- estimate top-down and bottom-up independently of each other
- use estimation checklists
- combine estimates from different experts and estimation strategies
- assess the uncertainty of the estimate
- provide feedback on the estimation accuracy and development task relations
- provide estimation training opportunities

- **Model assumptions**
    - Expert is available

- **Advantages**
    - Expert estimation is the dominant strategy when estimating software development effort
    - Simple to understand
    - Does not rely on project sizing (e.g. KLOC) to provide the estimate

- **Disadvantages**
    - How the estimation has been performed remains typically undocumented.

    - Subjective to the expert's opinion and bias.

- **Sources:**
    - [Jørgensen 2004] Jørgensen, M. A review of studies on expert estimation of software development effort, Journal of Systems and Software, Volume 70, Issues 1–2, February 2004, Pages 37-60, ISSN 0164-1212, http://dx.doi.org/10.1016/S0164-1212(02)00156-5.

Available Implementation: N/A

A.1.9. Estimation by Analogy (EbA)

Evaluation Method Name: Estimation by Analogy (EbA)
Measure: Development Effort
Unit: Person-months

Applicability: Anytime, data is available

Parameters: --

Drivers: --

Formula: --

Description: EbA is applicable when other projects in the same application domain have been completed. The cost of a new project is estimated by analogy with these completed projects. EBA can be performed according to one of the following three scenarios [Jørgensen et al. 2003]:

1. Pure expert judgment (the "database" of previous projects is in the expert's head)
2. Expert estimation informally supported by a database containing information about previous projects, or
3. Estimation based on the use of automatic clustering algorithms to find similar projects from a database.

To discover similar or analogous projects typical methods are: Top-K, K-nearest neighbours, where all projects need to be compared with similarity measures like the squared Euclidean distance, Cosine similarity, Gaussian distance, RBF kernel.

An example approach is AQUA from [Li et al. 2007].

There are three basic steps for estimating effort by analogy using AQUA:

Step 1. Retrieve analogy

In order to get the Top-N similar objects from DB, s_g is compared with all the objects in R through similarity measures. Similarity measures between two objects over a set of attributes are defined in terms of local and global similarity measures (Richter, 1995). Local similarity measure $Lsim$: $M_j \times M_j \rightarrow [0, 1] \cup \{NULL\}$ is defined as measuring the similarity between two objects related to an attribute $a_j \in A$, where M_j is the type of attribute a_j.

The global similarity measure between $s_g \in S$ and $r_i \in R$, $Gsim$: $S \times R \rightarrow [0, 1] \cup \{NULL\}$, is defined as a function of local similarity measures: $Gsim(s_g, r_i) = f(Lsim(a_1(s_g,), a_1(r_i)), Lsim(a_2(s_g,),a_2(r_i)), \ldots, Lsim(a_m(s_g,),a_m(r_i)))$. The concrete function f will be defined in Section 3.6. The result of this step is a set of vectors regarding the analogy of s_g and corresponding global similarities: $Y(s_g) = \{<r_1, Gsim(s_g, r_1)>, \ldots, <r_n, Gsim(s_g, r_n)>\}$.

Detailed definitions of the types of attributes, as well as the local and global similarity measures, will be given in Sections 3.3, 3.5 and 3.6, respectively.

Step 2. Determine the Top-N similar objects

In order to predict the effort of s_g, the Top-N similar objects $R_{topN}(s_g)$ are selected from R based on $Y(s_g)$. Different from the existing analogy-based effort estimation methods, thresholds of both similarity measures and the number of analogies used for analogy adaptation will be considered in order to determine $R_{topN}(s_g)$ in AQUA. For given N and T, the set $R_{topN}(s_g)$ is defined as follows:

$R_{topN}(s_g) = \{r_1^g, r_2^g, \ldots, r_N^g\}$, with

(N_1) number of analogies for adaptation is N, i.e., $|R_{topN}(s_g)| = N$,

(N_2) the global similarity between s_g and all objects in R_{topN} must be greater than the given threshold T: $Gsim(s_g, r_i^g) \geq T$ for all $r_i^g \in R_{topN}$ and

(N_3) R_{topN} includes only the closest N objects to s_g in R in terms of global similarity: $Gsim(s_g, r_i^g) \geq Gsim(s_g, r_x)$ for all $r_i^g \in R_{topN}$ and all $r_x \in (R/R_{topN})$, where (R/R_{topN}) is the subtraction of sets, which includes the objects in R but not in R_{topN}.

Step 3. Predict effort

Given $R_{topN}(s_g)$, the effort estimate of s_g is then adapted from the values of $Effort$ of objects from $R_{topN}(s_g)$: $Effort(s_g) = g(Effort(r_1^g), Effort(r_2^g), \ldots, Effort(r_N^g))$.

Again, the adaptation strategy g will be given in Section 3.6.

- **Model assumptions**
  - Prior knowledge exists (stored or can be accessed).

- **Advantages**
  - Clear mathematical reasoning.
  - Provides opportunities for weighing and obtaining local and global measures.

- **Disadvantages**
  - Similar projects in relation to the project needing the estimate exist, are documented or accessible.
  - Requires apriori selection of the most important attributes.

- **Sources:**
  - [Jørgensen et al. 2003] Jørgensen, M., Indahl, U., Sjøberg, D. (2003) Software effort estimation by analogy and regression toward the mean. J Syst Softw 68(3):253–262.
  - Li, J., Ruhe, G., Al-Emran, A. and Richter, M.M., 2007. A flexible method for software effort estimation by analogy. Empirical Software Engineering, 12(1), pp.65-106.

Available Implementation: Earlier implementation of ANaloGy SoftwarE TooL (ANGEL) http://dec.bournemouth.ac.uk/ESERG/ANGEL/; not available any more.

A.1.10. Weighted Micro Function Points

Evaluation Method Name: Weighted Micro Function Points
Measure: Development Effort
Unit: Percentage of the whole unit

Applicability:

Parameters:

- **Flow complexity (FC):** Measures the complexity of a programs' flow control path in a similar way to the traditional cyclomatic complexity, with higher accuracy by using weights and relations calculation.
- **Object vocabulary (OV):** Measures the quantity of unique information contained by the programs' source code, similar to the traditional Halstead vocabulary with dynamic language compensation.
- **Object conjuration (OC):** Measures the quantity of usage done by information contained by the programs' source code.

- **Arithmetic intricacy (AI):** Measures the complexity of arithmetic calculations across the program
- **Data transfer (DT):** Measures the manipulation of data structures inside the program
- **Code structure (CS):** Measures the amount of effort spent on the program structure such as separating code into classes and functions
- **Inline data (ID):** Measures the amount of effort spent on the embedding hard coded data
- **Comments (CM):** Measures the amount of effort spent on writing program comments

Drivers:

- **D:** The cost drivers factor supplied by the user input

Formula: The WMFP algorithm uses a three-stage process: function analysis, APPW transform, and result translation. A dynamic algorithm balances and sums the measured elements and produces a total effort score. The basic formula is:

$$\sum(WiMi)^{\prod Dq}$$

M = the source metrics value measured by the WMFP analysis stage

W = the adjusted weight assigned to metric M by the APPW model

N = the count of metric types

i = the current metric type index (iteration)

D = the cost drivers factor supplied by the user input

q = the current cost driver index (iteration)

K = the count of cost drivers

Description:

WMFP uses a parser to understand the source code breaking it down into micro functions and derive several code complexity and volume metrics, which are then dynamically interpolated into a final effort score. A three-stage process is used: function analysis, APPW transform, and result translation. The basic formula is used.
A dynamic algorithm balances and sums the measured elements and produces a total effort score. This score is then transformed into time by applying a statistical model called average programmer profile weights (APPW) which is a proprietary successor to COCOMO II 2000 and COSYSMO. The resulting time in programmer work hours is then multiplied by a user defined cost per hour of an average programmer, to produce an average project cost, translated to the user currency.

- **Model assumptions**
  - Assumes availability of code.

- **Advantages**

    - Produces more accurate results than traditional software sizing methodologies
    - Requires less configuration and knowledge from the end user, as most of the estimation is based on automatic measurements of an existing source code
    - Compatible with waterfall and newer SDLCs, such as Six Sigma, Boehm spiral, and Agile (AUP/Lean/XP/DSDM) methodologies

- **Disadvantages**
    - The basic elements of WMFP, when compared to traditional sizing models such as COCOMO, are more complex to a degree that they cannot realistically be evaluated by hand, even on smaller projects, and require a software to analyze the source code. Thus it can only be used as an Estimation by Analogy method.

- **Sources:**
    - https://en.wikipedia.org/wiki/Weighted_Micro_Function_Points
    - [ProjectCodeMeter 2010] "ProjectCodeMeter Users Manual" pp. 33–34 (2010).
    - [Jones 2009] Jones C. "Software Engineering Best Practices": pp. 318–320 [1] (October 2009)
    - [TickIT 209] TickIT Quarterly publication (2009) "Quarter 1, 2009": page 13

Available Implementation: http://www.projectcodemeter.com/cost_estimation/index.html

B. Property models for Development Time

```
DevelopmentTimePropertyModel = <"Development Time", {"COCOMO 1",
"COCOMO II", "Early design COCOMO II", "Expert estimation"}>
```

Development Time

PropID: Development Time
Data_format: Positive integer (unit:month)
Documentation:
Development time is the estimated time to develop the software, expressed in months.

B.1. Evaluation methods for Development Time

B.1.1. COCOMO 1

Evaluation Method Name: COCOMO 1
Measure: Development Time
Unit: Month

Applicability: familiar projects, ambitious projects, tightly constrained/complex projects

Parameters:
- Effort

Drivers:
- Development mode (organic, semi-detached, embedded)
- Application domain
- Project characteristics: Size, Innovation, Deadline, Dev. Environment

Formula: 2,5 * (Effort)^c
with *c* a constant which is set based on the mode of development according to the following table:

Development mode	**c**
organic	0,38
semi-detached	0,35
embedded	0,32

Description:
The model relies on the characteristics of the project, which allows to decide upon one of the three following development modes:
- *Organic*: for relatively small teams developing software in a highly familiar, in-house environment
- *Semi-detached*: when the team members have some experience related to some aspects of the system under development but not others and the team is composed of experienced and inexperienced people.
- *Embedded*: if the project must operate within a strongly coupled complex of hardware, software, regulations, and operational procedures, such as real-time systems.

These modes range from the familiar to the ambitious, tightly constrained development projects.

Another way to describe the development mode is through the following project characteristics:

Development Mode	Project Characteristics			
	Size	Innovation	Deadline/constraints	Dev. Environment
Organic	Small	Little	Not tight	Stable
Semi-detached	Medium	Medium	Medium	Medium
Embedded	Large	Greater	Tight	Complex hardware/ customer interfaces

Table from [Merlo-Schett et al. 2002-03].

- **Model assumptions**
  - The calculation is based on the development effort calculated through one of the COCOMO evaluation methods (Basic COCOMO 1, Intermediate COCOMO 1, Advanced COCOMO 1).
  - In Basic COCOMO 1, there are 152 hours per person-month. According to the organization this value may differ from the standard by 10% to 20% [Merlo-Schett et al. 2002-03].
  - In addition to the EAF, the model parameter "a" is slightly different in Intermediate COCOMO from the Basic model. The parameter "b" remains the same in both models.The following formula is used to calculate the difference in the Effort:

$$MM_{Korr} = EAF * MM_{nominal}$$

Equation 1: intermediate COCOMO; man month correction

- **Advantages**
  - Transparent, one can see how it works.
  - Good for quick, early, rough order of magnitude estimate of software cost.
  - Drivers are particularly helpful to the estimator t o understand the impact of different factors that affect project costs.

- **Disadvantages**
  - Limited accuracy (does not take into account the factors known to significantly affect the development effort).

- **Sources:**
  - [Boehm 1981] Boehm, Barry W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, 1981
  - [Merlo-Schett et al. 2002-03] Merlo-Schett N, Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of

Zurich, Switzerland. 2002-2003.
https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf

Available Implementation: None.

B.1.2. COCOMO II

Evaluation Method Name: COCOMO II
Measure: Development time
Unit: Month

Applicability: Application Generator, System Integration, or Infrastructure developments

Parameters:

- Person-month
- Schedule

Drivers:

- PREC, FLEX, RESL, TEAM, PMAT

Formula:

$$TDEV = \left[3.67 \times (PM)^{(0.28 + 0.2 \times (B - 1.01))}\right] \cdot \frac{\text{SCED\%}}{100}$$

where

$$B = 0.91 + 0.01 \sum_{j=1}^{5} SF_j$$

Formula from [USC 2000]

Symbol	Description
PM	Person Months of estimated effort from Early Design or Post-Architecture models (excluding the effect of the SCED effort multiplier).
SF	Scale Factors: PREC, FLEX, RESL, TEAM, PMAT
TDEV	Time to develop
SCED	Schedule
SCED%	The compression / expansion percentage in the SCED effort multiplier

Table from [USC 2000]

Description:

- **Model assumptions**
  - Development effort is estimated from Early Design or Post-Architecture models.

- **Advantages**
  - Transparent, one can see how it works.
  - Good for quick, early, rough order of low magnitude estimate of development time.

- **Disadvantages**
  - Dependent on variables that are not easily estimated (e.g., SCED).

- **Sources:**
  - [USC 2000] USC COCOMO II manual, http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf

Available Implementation:

B.1.3. Early design COCOMO II

Evaluation Method Name: Early design COCOMO II
Measure: Development Time
Unit: calendar time

Applicability: Early design phase, familiar projects, application generation with more granular information

Parameters:

- PM person-months
- C is a constant = 3.67 calibrated for COCOMO II
- D is a constant = 0.28 calibrated for COCOMO II
- F is a calculated parameter

Drivers:
Formula:

$$TDEV_{NS} = Cx(PM_{NS})^{F}$$

where $F = D + 0.2x0.01x\sum_{j=1}^{5} SF_j$

where $F = D + 0.2 \times (E-B)$

Equation 3: Time to develop

and C = 3.67 and D = 0.28

$$PM_{NS} = AxSize^{E} x \prod_{i=1}^{n} EM_i$$

where A = 2.94 (for COCOMO II.2000)

Equation 2: Person month

where $$E = B + 0.01x \sum_{j=1}^{5} SF_j$$

where B = 0.91 (for COCOMO II.2000)

$$B = \beta_0 + \sum_{i=1}^{5} \beta_i SF_i$$

Description: Model supports estimation in the Early Design stage of a project, when less is known about the project's cost drivers. The same approach is used for the Post Architecture COCOMO II.

It is based on the Development effort estimation formulas, i.e., it uses the PM (person-months), FP number of function points and EM effort multipliers, *a, b* two constants, a is the multiplicative constant and b is the scale factors.

The five Scale Factors are:

1. PREC Precedentedness (how novel the project is for the organization)
2. FLEX Development Flexibility
3. RESL Architecture / Risk Resolution
4. TEAM Team Cohesion
5. PMAT Process Maturity

Scale Factors (W_i)		annotation
PREC	If a product is similar to several previously developed project, then the precedentedness is high	Describe much the same influences that the original Development Mode did, largely intrinsic to a project and uncontrollable
FLEX	Conformance needs with requirements / external interface specifications, ...	
RESL	Combines Design Thoroughness and Risk Elimination (two scale factors in Ada).	Identify management controllables by which projects can reduce diseconomies of scale by reducing sources of project turbulence, entropy and rework.
TEAM	accounts for the sources of project turbulence and entropy because of difficulties in synchronizing the project's stakeholders.	
PMAT	time for rating: project start. Two ways for rating: 1. by the results of an organized evaluation based on the SEI CMM, 2. 18 Key Process Areas in the SEI CMM.	

Table 4: scale factors description for COCOMO II

Scale factors from [Merlo-Schett et al. 2002-03]

- **Model assumptions**
  - Calibrations are not obsolete

- **Advantages**
  - It can be used without collecting much information
  - It can be used early, before the entire architecture is determined

- **Disadvantages**
  - No available implementation.

- **Sources:**
  - [Merlo-Schett et al. 2002-03] Merlo-Schett N, Glinz M, Mukhija A. *COCOMO (Constructive Cost Model) - Seminar on Software Cost Estimation*. University of Zurich, Switzerland. 2002-2003. https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf
  - [USC 2000] USC COCOMO II manual, http://csse.usc.edu/csse/research/COCOMOII/cocomo2000.0/CII_manual2000.0.pdf

Available Implementation:

B.2.4. Expert estimation

Evaluation Method Name: Expert estimation
Measure: Development Time
Unit: Month

Applicability: Similar projects, expertise and experience, to any case

Parameters:
- Experts value estimate
- Experts confidence, experience, skill
- Experts weights
- Expert's background
- Available checklist of information related to the project

Drivers: Expert's experience and knowledge, gut-feeling.

Formula: N/A

Description:
In expert estimation, the estimation work is conducted by a person recognized as an expert on the task, and that a significant part of the estimation process is based on non-explicit and non-recoverable reasoning process, i.e. “intuition”.[Jørgensen 2002]. Expert estimation is the dominant strategy when estimating software development effort.

Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached [Sommerville 2010]

It has been demonstrated in [Jørgensen 2004] that in some cases expert estimates are more accurate than other types of estimation methods. This is the case when the methods used are not properly calibrated, or for cases in which having specific knowledge is important (e.g. knowing who works on which part of the project).

According to [Jørgensen 2004], the following 12 principles can improve the accuracy of expert estimations:

- evaluate estimation accuracy but avoid high evaluation pressure
- avoid conflicting estimation goals (e.g. “bid”, “planned”, “most likely” and “wishful thinking” vs “realism)
- ask the estimators to justify and criticize their estimates
- avoid irrelevant and unreliable estimation information
- use documented data from previous development tasks
- find estimation experts with relevant domain background and good estimation records
- estimate top-down and bottom-up independently of each other
- use estimation checklists
- combine estimates from different experts and estimation strategies
- assess the uncertainty of the estimate
- provide feedback on the estimation accuracy and development task relations
- provide estimation training opportunities

- **Model assumptions**
    - Expert is available

- **Advantages**
    - Expert estimation is the dominant strategy when estimating software development effort
    - Simple to use
    - Does not rely on project sizing (e.g. KLOC) to provide the estimate

- **Disadvantages**
    - How the estimation has been performed remains typically undocumented

- **Sources:**
    - [Jørgensen 2004] Jørgensen, M. A review of studies on expert estimation of software development effort, Journal of Systems and Software, Volume 70, Issues 1–2, February 2004, Pages 37-60, ISSN 0164-1212, http://dx.doi.org/10.1016/S0164-1212(02)00156-5.

Available Implementation: N/A

C. Property models for Cost

```
CostPropertyModel = <Cost, {“Cost Estimation”, “Expert estimation”}>
```

Cost

PropID: Cost
Data_format: Number (unit:)
Documentation:
Cost is the total cost of a development project (according to [Sommerville 2010]) and involves three parameters: (1) hardware and software costs, including maintenance, (2) travel and training costs, and (3) effort costs (the costs of paying software engineers). Depending on the type of project and organization, any of these costs can be the dominant cost.

Organizations compute effort costs in term of overhead costs where they take the total cost of running the organization and divide this by the number of productive staff. Therefore, the following costs are all part of the total effort cost:

- costs of providing, heating and lighting office space
- costs of support staff such as accountants, secretaries, cleaners, etc.
- costs of networking and communications
- costs of central facilities, such as libraries, recreational facilities
- costs of pensions, health insurance, etc.

- **Sources:** [Sommerville 2010] Sommerville, I., Software Engineering, 9th ed., Addison Wesley, 2010

C.1. Evaluation methods for Cost

C.1.1. Cost estimation

Evaluation Method Name: Cost estimation
Measure: Cost
Unit: monetary equivalent (USD?)

Applicability: any domain, anytime

Parameters:

- Development Effort
- Average Monthly Salary for developers

Drivers: Economy

Formula: Cost **=** (Development Effort)* Salary
Description:

- **Model assumptions**
  - Development effort can be estimated

- **Advantages**
  - Simple, generic and applicable

- **Disadvantages**
  - Too generic

- **Sources:**
  - [Abbas et al. 2012] Abbas, Syed Ali, et al. "Cost Estimation: A Survey of Well-known Historic Cost Estimation Techniques." Journal of Emerging Trends in Computing and Information Sciences 3.4 (2012): 612-636.

Available Implementation: http://www.payscale.com/

C.2.2. Expert estimation

Evaluation Method Name: Expert estimation
Measure: Development Time
Unit: Month

Applicability: Similar projects, expertise and experience, to any case

Parameters:
- Experts value estimate
- Experts confidence, experience, skill
- Experts weights
- Expert's background
- Available checklist of information related to the project

Drivers: Expert's experience and knowledge, gut-feeling.

Formula: N/A

Description:
In expert estimation, the estimation work is conducted by a person recognized as an expert on the task, and that a significant part of the estimation process is based on non-explicit and non-recoverable reasoning process, i.e. "intuition" [Jørgensen 2004]. Expert estimation is the dominant strategy when estimating software development effort.

Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached [Sommerville 2010].

It has been demonstrated in [Jørgensen 2004] that in some cases expert estimates are more accurate than other types of estimation methods. This is the case when the methods used are not properly calibrated, or for cases in which having specific knowledge is important (e.g. knowing who works on which part of the project).

According to [Jørgensen 2004], the following 12 principles can improve the accuracy of expert estimations:
- evaluate estimation accuracy but avoid high evaluation pressure
- avoid conflicting estimation goals (e.g. "bid", "planned", "most likely" and "wishful thinking" vs "realism)
- ask the estimators to justify and criticize their estimates

- avoid irrelevant and unreliable estimation information
- use documented data from previous development tasks
- find estimation experts with relevant domain background and good estimation records
- estimate top-down and bottom-up independently of each other
- use estimation checklists
- combine estimates from different experts and estimation strategies
- assess the uncertainty of the estimate
- provide feedback on the estimation accuracy and development task relations
- provide estimation training opportunities

- **Model assumptions**
    - Expert is available

- **Advantages**
    - Expert estimation is the dominant strategy when estimating software development effort
    - Simple to use
    - Does not rely on project sizing (e.g. KLOC) to provide the estimate

- **Disadvantages**
    - How the estimation has been performed remains typically undocumented

- **Sources:**
    - [Jørgensen 2004] Jørgensen, M. A review of studies on expert estimation of software development effort, Journal of Systems and Software, Volume 70, Issues 1–2, February 2004, Pages 37-60, ISSN 0164-1212, http://dx.doi.org/10.1016/S0164-1212(02)00156-5.

Available Implementation: N/A

D. Property models for Performance

```
PerformancePropertyModel1 = <CAN_sched, {"Controlled Area Network
(CAN) schedulability analysis"}

PerformancePropertyModel2 = <WCET, {“Worst-case execution time (WCET)
analysis based on Implicit Path Enumeration Technique (IPET)”,
“Expert estimation”, “Measurement-based WCET analysis”,
“Probabilistic Hybrid WCET Analysis”}>

PerformancePropertyModel3 = <E2E_respTime, {“End-to-end response time
(or delay) analysis”, ...}>

PerformancePropertyModel4 = <Stoch_CAN, {“stochastic analysis of
CAN-based Real-Time Automotive Systems”}>
```

CAN schedulability

PropID: CAN_sched
Data_format: boolean (unit: N/A)
Documentation:
CAN schedulability uses worst-case response time of CAN messages to check and eventually guarantee that messages between CAN-connected nodes do not exceed their deadlines. A CAN_sched value of true means that it is schedulable, while false means not schedulable.

D.1 Evaluation methods for CAN schedulability analysis

D.1.1. Controller Area Network (CAN) schedulability analysis

Evaluation method Name: Controlled Area Network (CAN) schedulability analysis
Measure: CAN_sched
Unit: schedulable / not schedulable (boolean)

Applicability: real-time embedded systems with CAN-based communication (e.g., embedded distributed units (ECUs) communicating via CAN in automotive)

Parameters: data frames, number of nodes, static set of hard real-time msgs, msg IDs, max data bytes per msg, max transmission time per msg, msgs period (Tm), msg deadline (Dm), queuing jitter (Jm), queuing delay (wm), msg transition time (Cm)

Drivers: --

Formula: $\forall m = [1...n], R_m <= D_m$

(where $R_m = WCET$ of message *m*, and $D_m =$ deadline of message *m*)

Description: CAN is used extensively in automotive applications for communication across ECUs via messages that have specific deadlines which are not supposed to be missed. CAN schedulability analysis employs worst-case response time of CAN messages to check and eventually guarantee that messages do not exceed their deadlines.

- **Model assumptions**
  - fixed priority preemptive
  - Highest priority msg enters in arbitration when arbitration starts
  - Depending on the number of buffers of on-chip CAN controller, Dm <= Tm can be needed
  - One time domain wrt nodes clock

- **Advantages**
  - Crucial timing analysis in the automotive domain and in general hard real-time embedded systems with CAN-based communication.

- **Disadvantages**
  - It is limited to the model assumptions
  - Variations in the model affect the analysis and shall be thoroughly assessed

- **Sources:**
    - [Davis et al. 2007] Davis, R.I., Burns, A., Bril, R.J., Lukkien, J.J. Controller Area Network (CAN) schedulability analysis: Refuted, revisited and revised.Real-Time Systems. 2007 Apr 1;35(3):239-72.

Available Implementation: Volcano Network Architect (VNA) by Mentor Graphics

Worst-Case Execution Time

PropID: WCET
Data_format: number (unit:milliseconds)
Documentation:

- Worst-Case Execution Time (WCET) analysis is used to determine the WCET of a program (or a function) based on mathematical models representing software and hardware.

D.2. Evaluation methods for Worst-Case Execution Time

D.2.1. Worst-case execution time (WCET) analysis based on Implicit Path Enumeration Technique (IPET)

Evaluation method Name: Worst-case execution time (WCET) analysis based on Implicit Path Enumeration Technique (IPET)
Measure: WCET
Unit: milliseconds

Applicability: real-time systems, embedded, distributed on, e.g. CAN and LIN, tasks on different microcontrollers

Parameters: executable binary, annotations for additional external info: possible program flow, hardware configuration can be required, clock rate, routine entry, memory access, known register values, address mapping, recursion depth, loop bound, etc..

Drivers: ----

Formula: $WCET = max(\Sigma x_i \cdot t_i), i = [e_1 ... e_n]$
(x_i = execution frequency of edge e_i in the control flow graph, t_i = execution time of edge e_i, edges defined in the optimization problem as constraints)

Description:
To give system timing guarantees, WCET of tasks running on different microcontrollers is crucial. WCET analysis can be done both dynamically and statically. Static WCET analysis is used to determine the WCET of a program (or a function) based exclusively on mathematical models representing software and hardware.

- **Model assumptions**
  - no event-triggered are transmitted (possible)
  - no sporadic frames are transmitted (possible)
  - no sleep requests occur (possible)
  - no errors occur (possible)

- **Advantages**
  - description of complex flow facts is possible
  - generation of constraints is simple
  - constraints can be solved by existing tools

- **Disadvantages**
  - solving ILP is in general NP hard
  - flow facts that describe execution order are difficult to integrate
  - generic control flow
  - complex hardware
  - manual workload needed for annotation

- **Sources:**
  - [Byhlin et al 2005] Byhlin, S., Ermedahl, A., Gustafsson, J. and Lisper, B., 2005, July. Applying static WCET analysis to automotive communication software. In *Real-Time Systems, 2005.(ECRTS 2005). Proceedings. 17th Euromicro Conference on*(pp. 249-258). IEEE.

Available Implementation: AbsInt (http://www.absint.com/ait/, commercial or evaluation license), SWEET (http://www.mrtc.mdh.se/projects/wcet/sweet/index.html, open-source)

D.2.2. Expert estimation

Evaluation Method Name: Expert estimation
Measure: Development Time
Unit: Month

Applicability: Similar projects, expertise and experience, to any case

Parameters:
- Experts value estimate
- Experts confidence, experience, skill
- Experts weights
- Expert's background
- Available checklist of information related to the project

Drivers: Expert's experience and knowledge, gut-feeling.

Formula: N/A

Description:
In expert estimation, the estimation work is conducted by a person recognized as an expert on the task, and that a significant part of the estimation process is based on non-explicit and non-recoverable reasoning process, i.e. "intuition" [Jørgensen 2004]. Expert estimation is the

dominant strategy when estimating software development effort .

Several experts on the proposed software development techniques and the application domain are consulted. They each estimate the project cost. These estimates are compared and discussed. The estimation process iterates until an agreed estimate is reached [Sommerville 2010].

It has been demonstrated in [Jørgensen 2004] that in some cases expert estimates are more accurate than other types of estimation methods. This is the case when the methods used are not properly calibrated, or for cases in which having specific knowledge is important (e.g. knowing who works on which part of the project).

According to [Jørgensen 2004], the following 12 principles can improve the accuracy of expert estimations:

- evaluate estimation accuracy but avoid high evaluation pressure
- avoid conflicting estimation goals (e.g. “bid”, “planned”, “most likely” and “wishful thinking” vs “realism)
- ask the estimators to justify and criticize their estimates
- avoid irrelevant and unreliable estimation information
- use documented data from previous development tasks
- find estimation experts with relevant domain background and good estimation records
- estimate top-down and bottom-up independently of each other
- use estimation checklists
- combine estimates from different experts and estimation strategies
- assess the uncertainty of the estimate
- provide feedback on the estimation accuracy and development task relations
- provide estimation training opportunities

- **Model assumptions:** availability of experts.

- **Advantages**
    - Expert estimation is the dominant strategy when estimating software development effort
    - Simple to use
    - Does not rely on project sizing (e.g. KLOC) to provide the estimate

- **Disadvantages**
    - How the estimation has been performed remains typically undocumented

- **Sources:**
    - [Jørgensen 2004] Jørgensen, M. A review of studies on expert estimation of software development effort, Journal of Systems and Software, Volume 70,

Issues 1–2, February 2004, Pages 37-60, ISSN 0164-1212,
http://dx.doi.org/10.1016/S0164-1212(02)00156-5.

Available Implementation: N/A

D.2.3. Measurement-based WCET analysis

Evaluation method Name: measurement-based worst-case execution time (WCET) analysis
Measure: WCET
Unit: milliseconds

Applicability: real-time systems, embedded, distributed

Parameters:sample block of C/C++ code, user event markers in code

Drivers: ----

Formula: $WCET = \mu - \beta\, log(-log((1-p_e)^b))$
(μ = Gumbel distribution location parameter, β = Gumbel distribution scale parameter , p_e exceedance probability, b = block size of the sample block)

Description:
This measurement-based approach produces both a WCET estimate, and a prediction of the probability that a future execution time will exceed our estimate. The approach is statistical-based and uses extreme value theory to build a model of the tail behavior of the measured execution time value.

- **Model assumptions**
  - Uses a Gumbel distribution
  - Uses Chi-Squared test
  - Execution time samples are independent and thus the blocking method chosen will not affect the statistical properties of the blocks
  - No data dependent loops in tasks
  - Execution time distribution has a non-heavy tail for most tasks

- **Advantages**
  - Ability to predict the exceedance probability

- Exceedance probability more predictable and controllable than maximum Observed execution time
- Based on real execution traces
- Constraints can be solved by existing tools

- **Disadvantages**

- **Sources:**
  - [Hansen et al 2009] Hansen, Jeffery, Scott A. Hissam, and Gabriel A. Moreno. "Statistical-based wcet estimation and validation." *Proceedings of the 9th Intl. Workshop on Worst-Case Execution Time (WCET) Analysis*. 2009.

Available Implementation: $Lambda_{WBA}$ (http://www.sei.cmu.edu/predictability/tools/lambda/)

D.2.4. Probabilistic Hybrid WCET Analysis

Evaluation method Name: Hybrid worst-case execution time (WCET) analysis
Measure: WCET
Unit: milliseconds

Applicability: real-time systems, embedded, distributed

Parameters: basic code blocks, info about dependency among blocks

Drivers: ----

Formula: N/A

Description:
This approach combines (probabilistically) the worst case effects seen in individual blocks to build the execution time model of the worst case path of the program. Execution times are measured.

- **Model assumptions**
  - When blocks are not independent, involved random variables are assumed to be comonotonic
  - Irreducible structures belong to one block

    - Timing information of the execution time of each basic block in each run is available (calculated by the tool)

- **Advantages**
    - Portability: minimal dependence on processor architecture
    - Fully flexible timing program generation
    - Genericity: source of data for tracing analysis can be provided in many different ways
    - Automatic loop analysis: maximum number of iterations of loops are deduced Automatically from trace analysis

- **Disadvantages**
    - Can be slower than static analysis
    - Commercial tool

- **Sources:**
    - [Bernat et al 2002] Bernat, Guillem, Anotione Colin, and Stefan M. Petters. "WCET analysis of probabilistic hard real-time systems." *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*. IEEE, 2002.

Available Implementation: Rapitime (https://www.rapitasystems.com/products/rapitime, commercial)

End-to-end response time (or delay) analysis

PropID: E2E_respTime
Data_format: number (unit:milliseconds)
Documentation:
End-to-end response time analysis is a particular variant of schedulability analysis which calculates upper bounds on the response time (and delays) of task chains distributed over several nodes or in the system.

D.3. Evaluation methods for End-to-end response time (or delay) analysis

D.3.1. End-to-end response time (or delay) analysis

Evaluation method Name: End-to-end response time (or delay) analysis
Measure: E2E_respTime
Unit: milliseconds

Applicability: networked AUTOSAR-compliant systems, real-time systems, embedded, heterogeneous networks, e.g. CAN, LIN and FlexRay

Parameters: task period, task response time, task worst case response time, generated messages, static transmission time, transmission delay, worst case message transmission time, queueing protocol delay, ...

Drivers: ----

Formula:

$$\Delta_p^G \leq \sum_{i=1}^{n-1}(R_{p_i} + Q_{m_{p_i}} + C_{m_{p_i}} + T_{p_{i+1}}) + R_{p_n}$$

where R_{pi} denotes the worst-case response time of the task generating message m_{pi} , and $C_{m_{pi}}$ the worst-case transmission time of m_{pi}, $Q_{m_{pi}}$ represents the queueing delay for the protocol used to schedule m_{pi}, T_{pi+1} represents the period of the receiver task

Description: A generalized end-to-end timing analysis to handle hierarchical heterogeneous bus structures, by exploiting common underlying structures for different communication bus interfaces. Analytical infrastructure can therefore be leveraged to perform end-to-end timing analysis of networked AUTOSAR-compliant automotive systems.

- **Model assumptions**
  - AUTOSAR-compliant system
  - Communication based on a combination of LIN, CAN, FlexRay

- **Advantages**
  - Common structure between different communication bus interfaces
  - Generic end-to-end response-time bound for task pipelines spanning different communication networks
  - Practical constraints imposed by AUTOSAR result in high schedulable utilization values

- **Disadvantages**

- **Sources:**
  - [Lakshmanan et al 2010] Lakshmanan, K., Bhatia, G. and Rajkumar, R., 2010, March. Integrated end-to-end timing analysis of networked autosar-compliant systems. In*Proceedings of the Conference on Design, Automation and Test in Europe*(pp. 331-334). European Design and Automation Association.

Available Implementation: SysWeaver (http://users.ece.cmu.edu/~raj/publications.html#SysWeaver)

Stochastic Analysis of CAN-Based Real-Time Automotive Systems

PropID: Stoch_CAN
Data_format: number (unit: milliseconds)
Documentation:

- Stochastic analysis of CAN-based systems analyzes the timing performance of distributed automotive architecture with priority-based scheduling.

D.4. Evaluation methods for Stochastic Analysis of CAN-Based Real-Time Automotive Systems

D.4.1. Stochastic Analysis of CAN-Based Real-Time Automotive Systems

Evaluation method Name: Stochastic analysis of CAN-based RT systems
Measure: Stoch_CAN
Unit: milliseconds

Applicability: real-time systems, embedded, based on CAN and OSEK

Parameters: task period, task initial phase, task execution time, task priority, job arrival time, job release time, resource hyperperiod, average utilization, ...

Drivers: ----

Formula:

$$\mathbb{P}(\mathcal{R}_{i,j} = t - Q_{i,j}) = \mathbb{P}(\mathcal{F}_{i,j} = t), \forall t.$$

where $R_{i,j} = response\ time$, $Q_{i,j} = queuing\ time$, $F_{i,j} = finish\ time$, $at\ any\ time\ t$

Description:
Stochastic analysis of the timing performance of distributed automotive architecture with priority-based scheduling, i.e., OSEK compliant operating systems and the CAN bus protocol.

- **Model assumptions**
  - OSEK-compliant OS.
  - Communication on CAN.
  - Communication based on the periodic activation model.
  - Priority-based scheduling.
  - Communication based on the preservation of the latest written value and the overwriting of old ones (shared variable buffer).
  - Axecution times of all the jobs of are independent and identically distributed.
  - No clock synchronization among the nodes connected to the CAN bus.
  - All nodes boot up at arbitrary times.

- **Advantages**
  - Adds probability to analysis of the latency in the end-to-end propagation of information among periodically activated tasks and messages.

    - Proofs that technique provides a good approximation of the latency distribution.

- **Disadvantages**
    - Constrained due to the assumptions (described above).

- **Sources:**
    - [Zeng 2009] Zeng, H., Di Natale, M., Giusto, P. and Sangiovanni-Vincentelli, A., 2009. Stochastic analysis of CAN-based real-time automotive systems. *Industrial Informatics, IEEE Transactions on*, *5*(4), pp.388-401.

Available Implementation: ad-hoc (not available)