

COCOMO¹ II Data Collection Questionnaire

1. Introduction

The Center for Software Engineering at the University of Southern California is conducting research to update the software development cost estimation model called COCOMO. The project name is COCOMO II and is led by Dr. Barry W. Boehm.

A fundamental requirement for such research is real-world software development project data. This data will be used to test hypotheses and verify the model's postulations. In return the model will be open and made available to the public. The contribution of your data will ensure the final model is useful.

The data that is contributed is important to us. We will safeguard your contribution so as not to compromise company proprietary information. Some Affiliates have an active collection program and the data from past projects is available for the COCOMO II data collection effort. This questionnaire can be used to extract relevant COCOMO II data. A rosetta-stone that converts COCOMO '81 data to COCOMO II data is also available. Please contact us if you would like to get a copy.

This questionnaire attempts to address two different levels of data granularity: project level and component level. The project level of granularity is data that is applicable for the whole project. This includes things like application type and development activity being reported. Component level data are things like size, cost, and component cost drivers. If the data being submitted is on a project that has multiple components then fill out the project data once, and the component data for each of the identifiable component. If the data being submitted is for the whole project fill out the form once. The data collection activity for the COCOMO II research effort started in November 1994. The first calibration was published in 1997 based on 83 datapoints collected. It became popular as COCOMO II.1997 and produced estimates within 30% of the actuals 52% of the time for effort. The second calibration was published in 1998 based on 161 datapoints. It is known as COCOMO II.1998 and produces estimates within 30% of the actuals 71% of the time for effort. The aim of the COCOMO II research team is to continually update the existing COCOMO II database and to publish annual calibrations of the COCOMO II model. Hence by submitting your data to us, you play a significant role in the model calibration.

COCOMO II Points of Contact

For questions on the COCOMO II Model and its extensions, data definitions, or project data collection and management, contact:

A. Winsor Brown (Research Scientist)
Barry Boehm (Project Leader)
Internet Electronic-Mail

Voice: (213) 740-6599, Fax: (213) 740-4927
Voice: (213) 740-8163, Fax: (213) 740-4927
cocomo-info@sunset.usc.edu

COCOMO II Data Submission Address:

COCOMO II Data Submission
Center for Software Engineering
Department of Computer Science
Henry Salvatori Room 330
University of Southern California
941 W. 37th Place
Los Angeles, CA 90089-0781
U.S.A.

¹ COConstructive Cost Modeling (COCOMO) is defined in Software Engineering Economics by Barry W. Boehm, Prentice Hall, 1981

This page has been left blank intentionally.

2. Project Level Information

As described in the Introduction section of this questionnaire, project level information is applicable for the whole project. This includes things like application type and development activity being reported.

2.A General Information

2.1 Affiliate Identification Number. Each separate software project contributing data will have a separate file identification number of the form XXX. XXX will be one of a random set of three-digit organization identification numbers, provided by USC Center for Software Engineering to the Affiliate.

2.2 Project Identification Number. The project identification is a three-digit number assigned by the organization. Only the Affiliate knows the correspondence between YYY and the actual project. The same project identification must be used with each data submission.

2.3 Date prepared. This is the date the data elements were collected for submission.

2.4 Application Type. This field captures a broad description of the type of activity this software application is attempting to perform.

<u>Circle One:</u>	Command and Control,	MIS,	Simulation,
	Communication,	Operating Systems,	Software Dev. Tools,
	Diagnostics,	Process Control,	Testing,
	Engineering and Science	Signal processing,	Utilities

Other: _____

2.5 Development Type.

Is the development a new software product or an upgrade of an existing product?

<u>Circle One:</u>	New Product	Upgrade
--------------------	-------------	---------

2.6 Development Process. This is a description of the software process used to control the software development, e.g. waterfall, spiral, etc.

2.7 Step in Process. This field captures information about the project's position in its development process. The answers depend on the process model being followed.

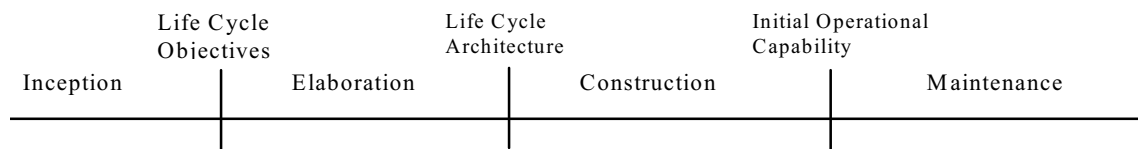
2.7.1 Waterfall Phase: Activity. This field captures the waterfall phase of development that the project is in. For one-time reporting the activity is 'completed'. It is assumed that data for completed projects includes data from software requirements through integration/test. Please report the correct phasing if this is not the case.

Circle One: Requirements, Design, Code,
 Unit Test, Integration/Test, Maintenance,
 Completed

Other: _____

2.7.2 MBASE Phase. Phase refers to the aggregate of activities between the life cycle anchor points². The four phases, based on the Rational Unified Process (RUP) [see Rational's Objectory Process³], and the anchor points are shown on the timeline below. Please see section 2 of the "CORADMO⁴ Extensions of COCOMO II Schedule Estimation Questionnaire" for more details.

Please circle the most advance anchor point (milestone) the project has achieved.



See the Appendix A for definitions of the LCO, LCA, and IOC milestones. The COCOMO II model covers the effort required from the completion of the LCO to IOC. If you are using a waterfall model, the corresponding milestones are the Software Requirements Review, Preliminary Design Review, and Software Acceptance Test.

2.8 Development Process Iteration.

2.8a. If the product development process is iterative, e.g. spiral or represents a released version, which iteration is this?

2.8b. If the development process phases are is iterative, e.g. spiral, how many iterations have there been?

Inception: _____ Elaboration: _____ Construction: _____

² Barry W. Boehm, "Anchoring the Software Process," *IEEE Software*, 13, 4, July 1996, pp. 73-82. An unabridged version, dated November 1995, is in Appendix A.

³ Rational Corp., "Rational Objectory Process 4.1 – Your UML Process", available at <http://www.rational.com/support/techpapers/toratojprcs/>.

⁴ COConstructive RAD schedule and effort MOdel

2.9 COCOMO Model. This specifies which COCOMO II model is being used in this data submission. If this is a "historical" data submission, select the Post-Architecture model or the Applications Composition model.

- Application Composition: This model involves prototyping efforts to resolve potential high-risk issues such as user interfaces, software/system interaction, performance, or technology maturity.
- Early Design: This model involves exploration of alternative software/system architectures and concepts of operations. At this stage of development, not enough is known to support fine-grain cost estimation.
- Post-Architecture: This model involves the actual development and maintenance of a software product. This stage of development proceeds most cost-effectively if a software life-cycle architecture has been developed; validated with respect to the system's mission, concept of operation, and risk; and established as the framework for the product.

Circle One: Application Composition, Early Design, Post-Architecture

2.10 Success Rating for Project. This specifies the degree of success for the project.

- Very successful; did almost everything right
- Successful; did the big things right
- OK; stayed out of trouble
- Some Problems; took some effort to keep viable
- Major Problems; would not do this project again

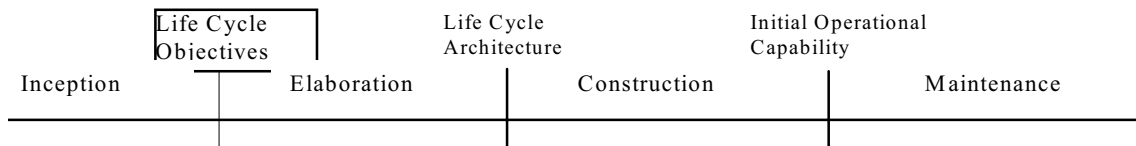
Circle One: Very Successful Successful OK Some Problems Major Problems

Schedule

2.11 Year of development. For reporting of historical data, please provide the year in which the software development was completed. For periodic reporting put the year of this submission or leave blank.

2.12 Schedule Months. For reporting of historical data, provide the number of calendar months from the time the development began through the time it completed. For periodic reporting, provide the number of months in this development activity.

Circle the life-cycle phases that the schedule covers:



Schedule in months: _____

Project Exponential Cost Drivers

Scale Factors (Wi)	Very Low	Low	Nominal	High	Very High	Extra High
Precedentedness	thoroughly unprecedented	largely unprecedented	somewhat unprecedented	generally familiar	largely familiar	Thoroughly familiar
Development Flexibility	rigorous	occasional relaxation	some relaxation	general conformity	some conformity	General goals
Architecture / risk resolution ^a	little (20%)	some (40%)	often (60%)	generally (75%)	mostly (90%)	full (100%)
Team cohesion very difficult interactions	some difficult interactions	basically cooperative interactions	largely cooperative	highly cooperative	seamless interactions	

^a % significant module interfaces specified, % significant risks eliminated.

Enter the rating level for the first four cost drivers.

2.13 Precedentedness (PREC). If the product is similar to several that have been developed before then the precedednedness is high. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Extra High	Don't Know

2.14 Development Flexibility (FLEX). This cost driver captures the amount of constraints the product has to meet. The more flexible the requirements, schedules, interfaces, etc., the higher the rating. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Extra High	Don't Know

2.15 Architecture / Risk Resolution (RESL). This cost driver captures the thoroughness of definition and freedom from risk of the software architecture used for the product. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Extra High	Don't Know

2.16 Team Cohesion (TEAM). The Team Cohesion cost driver accounts for the sources of project turbulence and extra effort due to difficulties in synchronizing the project's stakeholders: users, customers, developers, maintainers, interfacers, others. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Extra High	Don't Know

2.17 Process Maturity (PMAT). The procedure for determining PMAT is organized around the Software Engineering Institute's Capability Maturity Model (CMM). The time period for reporting process maturity is at the time the project was underway. We are interested in the capabilities practiced at the project level more than the overall organization's capabilities. There are three ways of responding to this question: choose only one. "Key Process Area Evaluation" requires a response for each Key Process Area (KPA). We have provided enough information for you to self-evaluate the project's enactment of a KPA (we hope you will take the time to complete this section). "Overall Maturity Level" is a response that captures the result of an organized evaluation based on the CMM. "No Response" means you do not know or will not report the process maturity either at the Capability Maturity Model or Key Process Area level.

☐ **No Response**

Overall Maturity Level

- ☐ CMM Level 1 (lower half)
- ☐ CMM Level 1 (upper half)
- ☐ CMM Level 2
- ☐ CMM Level 3
- ☐ CMM Level 4
- ☐ CMM Level 5

Basis of estimate:

- ☐ Software Process Assessment (SPA)
- ☐ Software Capability Evaluation (SCE)
- ☐ Interim Process Assessment (IPA)
- ☐ Other: _____

Key Process Area Evaluation

Enough information is provided in the following table so that you can assess the degree to which a KPA was exercised on the project.

- Almost Always (over **90%** of the time) when the goals are consistently achieved and are well established in standard operating procedures.
- Frequently (about **60 to 90%** of the time) when the goals are achieved relatively often, but sometimes are omitted under difficult circumstances.
- About Half (about **40 to 60%** of the time) when the goals are achieved about half of the time.
- Occasionally (about **10 to 40%** of the time) when the goals are sometimes achieved, but less often.
- Rarely If Ever (less than **10%** of the time) when the goals are rarely if ever achieved.
- Does Not Apply when you have the required knowledge about your project or organization and the KPA, but you feel the KPA does not apply to your circumstances (e.g. Subcontract Management).
- Don't Know when you are uncertain about how to respond for the KPA.

This page has been left blank intentionally.

Key Process Area	Goals of each KPA	Almost Always	Very Often	About Half	Some Times	Rarely If Ever	Does Not Apply	Don't Know
Requirements Management: involves establishing and maintaining an agreement with the customer on the requirements for the software project.	System requirements allocated to software are controlled to establish a baseline for software engineering and management use. Software plans, products, and activities are kept consistent with the system requirements allocated to software.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Software Project Planning: establishes reasonable plans for performing the software engineering activities and for managing the software project.	Software estimates are documented for use in planning and tracking the software project. Software project activities and commitments are planned and documented. Affected groups and individuals agree to their commitments related to the software project.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Software Project Tracking and Oversight: provides adequate visibility into actual progress so that management can take corrective actions when the software project's performance deviates significantly from the software plans.	Actual results and performances are tracked against the software plans. Corrective actions are taken and managed to closure when actual results and performance deviate significantly from the software plans. Changes to software commitments are agreed to by the affected groups and individuals.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Software Subcontract Management: involves selecting a software subcontractor, establishing commitments with the subcontractor, and tracking and reviewing the subcontractor's performance and results.	The prime contractor selects qualified software subcontractors. The prime contractor and the software subcontractor agree to their commitments to each other. The prime contractor and the software subcontractor maintain ongoing communications. The prime contractor tracks the software subcontractor's actual results and performance against its commitments.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Software Quality Assurance: provides management with appropriate visibility into the process being used by the software project and of the products being built.	Software quality assurance activities are planned. Adherence of software products and activities to the applicable standards, procedures, and requirements is verified objectively. Affected groups and individuals are informed of software quality assurance activities and results. Noncompliance issues that cannot be resolved within the software project are addressed by senior management.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Key Process Area	Goals of each KPA	Almost Always	Very Often	About Half	Some Times	Rarely If Ever	Does Not Apply	Don't Know
Software Configuration Management: establishes and maintains the integrity of the products of the software project throughout the project's software life cycle.	Software configuration management activities are planned. Selected software work products are identified, controlled, and available. Changes to identified software work products are controlled. Affected groups and individuals are informed of the status and content of software baselines.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization Process Focus: establishes the organizational responsibility for software process activities that improve the organization's overall software process capability.	Software process development and improvement activities are coordinated across the organization. The strengths and weaknesses of the software processes used are identified relative to a process standard. Organization-level process development and improvement activities are planned.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Organization Process Definition: develops and maintains a usable set of software process assets that improve process performance across the projects and provides a basis for cumulative, long- term benefits to the organization.	A standard software process for the organization is developed and maintained. Information related to the use of the organization's standard software process by the software projects is collected, reviewed, and made available.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Training Program: develops the skills and knowledge of individuals so they can perform their roles effectively and efficiently.	Training activities are planned. Training for developing the skills and knowledge needed to perform software management and technical roles is provided. Individuals in the software engineering group and software-related groups receive the training necessary to perform their roles.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Integrated Software Management: integrates the software engineering and management activities into a coherent, defined software process that is tailored from the organization's standard software process and related process assets.	The project's defined software process is a tailored version of the organization's standard software process. The project is planned and managed according to the project's defined software process.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Software Product Engineering: integrates all the software engineering activities to produce and support correct, consistent software products effectively and efficiently.	The software engineering tasks are defined, integrated, and consistently performed to produce the software. Software work products are kept consistent with each other.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Intergroup Coordination: establishes a means for the software engineering group to participate actively with the other engineering groups so the project is better able to satisfy the customer's needs effectively and efficiently.	The customer's requirements are agreed to by all affected groups. The commitments between the engineering groups are agreed to by the affected groups. The engineering groups identify, track, and resolve intergroup issues.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Key Process Area	Goals of each KPA	Almost Always	Very Often	About Half	Some Times	Rarely If Ever	Does Not Apply	Don't Know
Peer Review: removes defects from the software work products early and efficiently.	Peer review activities are planned. Defects in the software work products are identified and removed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Quantitative Process Management: controls the process performance of the software project quantitatively.	The quantitative process management activities are planned. The process performance of the project's defined software process is controlled quantitatively. The process capability of the organization's standard software process is known in quantitative terms.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Software Quality Management: involves defining quality goals for the software products, establishing plans to achieve these goals, and monitoring and adjusting the software plans, software work products, activities, and quality goals to satisfy the needs and desires of the customer and end user.	The project's software quality management activities are planned. Measurable goals for software product quality and their priorities are defined. Actual progress toward achieving the quality goals for the software products is quantified and managed.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Defect Prevention: analyzes defects that were encountered in the past and takes specific actions to prevent the occurrence of those types of defects in the future.	Defect prevention activities are planned. Common causes of defects are sought out and identified. Common causes of defects are prioritized and systematically eliminated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Technology Change Management: involves identifying, selecting, and evaluating new technologies, and incorporating effective technologies into the organization.	Incorporation of technology changes are planned. New technologies are evaluated to determine their effect on quality and productivity. Appropriate new technologies are transferred into normal practice across the organization.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Process Change Management: involves defining process improvement goals and, with senior management sponsorship, proactively and systematically identifying, evaluating, and implementing improvements to the organization's standard software process and the projects' defined software processes on a continuous basis.	Continuous process improvement is planned. Participation in the organization's software process improvement activities is organization wide. The organization's standard software process and the projects' defined software processes are improved continuously.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

2.C Distribution of Effort and Schedule By Phase

This subsection has additional metrics that are required to calibrate the distribution of effort and schedule by phase. Please fill this out if the necessary information is available.

2.21 Total Effort (Person Months). Divide the total effort required for the project into effort (in Person Months) required for each of the following three phases: Inception, Elaboration and Construction.

	LCO	LCA	IOC
	Inception	Elaboration	Construction
Effort Distribution			

2.22 Schedule Months. Divide the total time for development (schedule) required for the project into schedule (in Calendar Months) required for each of the following three phases: Inception, Elaboration and Construction.

	LCO	LCA	IOC
	Inception	Elaboration	Construction
Schedule Distribution			

3. Component Level Information

Component ID

If the whole project is being reported as a single component then skip to the next section.

If the data being submitted is for multiple components that comprise a single project then it is necessary to identify each component with its project. Please fill out this section for each component and attach all of the component sections to the project sections describing the overall project data.

3.1 Affiliate Identification Number. Each separate software project contributing data will have a separate file identification number of the form XXX. XXX will be one of a random set of three-digit organization identification numbers, provided by USC Center for Software Engineering to the Affiliate.

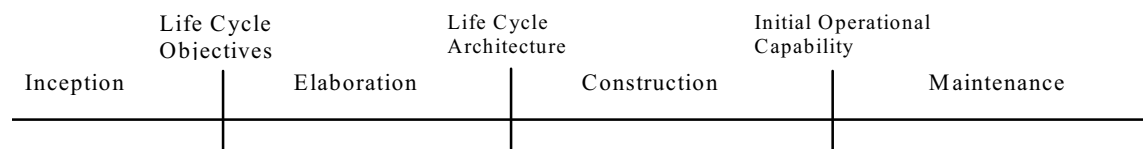
3.2 Project Identification Number. The project identification is a three digit number assigned by the organization. Only the Affiliate knows the correspondence between YYY and the actual project. The same project identification must be used with each data submission.

3.3 Component Identification (if applicable). This is a unique sequential letter that identifies a software module that is part of a project.

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

Cost

3.4 Total Effort (Person Months). Circle the life-cycle phases that the effort estimate covers:



Total Effort

3.5 Hours / Person Month. Indicate the average number of hours per person month experienced by your organization.

3.6 Labor Breakout. Indicate the percentage of labor for different categories, e.g. Managers, S/W Requirement Analysts, Designers, CM/QA Personnel, Programmers, Testers, and Interfacers for each phase of software development:

Categories Phase	Rqts.	Design	Code	Test	Management	CM, QA, Documentation
Inception						
Elaboration						
Construction						

Size

The project would like to collect size in application points, logical lines of code, and unadjusted function points. Please submit all size measures that are available, e.g. if you have a component in lines of code and unadjusted function points then submit both numbers.

3.7 Percentage of Code Breakage. This is an estimate of how much the requirements have changed over the lifetime of the project. It is the percentage of code thrown away due to requirements volatility. For example, a project which delivers 100,000 instructions but discards the equivalent of an additional 20,000 instructions would have a breakage of value of 20. See the Model Definition Manual for more detail.

3.8 Application Points. If the COCOMO II Applications Programming model was used then enter the application point count.

3.9 New Unique SLOC. This is the number of new source lines of code (SLOC) generated.

3.10 SLOC Count Type. When reporting size in source lines of code, please indicate if the count was for logical SLOC or physical SLOC. If both are available, please submit both types of counts. If neither type of count applies to the way the code was counted, please describe the method. An extensive definition for logical source lines of code is given in an Appendix in the Model Definition Manual.

Circle One: Logical SLOC Physical SLOC (carriage returns)
 Physical SLOC (semicolons) Non-Commented/Non-Blank SLOC

Other: _____

3.11 Unadjusted Function Points. If you are using the Early Design or Post-Architecture model, provide the total Unadjusted Function Points for each type. An Unadjusted Function Point is the product of the function point count and the weight for that type of point. Function Points are discussed in the Model Definition Manual.

3.12 Programming Language. If you are using the Early Design or Post-Architecture model, enter the language name that was used in this component, e.g. Ada, C, C++, COBOL, FORTRAN and the amount of usage if more than one language was used.

Language Used	Percentage Used

3.13 Software Maintenance Parameters. For software maintenance, use items 3.8 - 3.12 to describe the size of the base software product, and use the same units to describe the following parameters:

- a. Amount of software added: _____
- b. Amount of software modified: _____
- c. Amount of software deleted: _____

3.14 Application Points Reused. If you are using the Application Composition model, enter the number of application points reused. Do not fill in the fields on DM, CM, IM, SU, or AA.

3.15 ASLOC Adapted. If you are using the Early Design or Post-Architecture model enter the amounts for the SLOC adapted.

3.16 ASLOC Count Type. When reporting size in source lines of code, please indicate if the count was for logical ASLOC or physical ASLOC. If both are available, please submit both types of counts. If neither type of count applies to the way the code was counted, please describe the method. An extensive definition for logical source lines of code is given in an Appendix in the Model Definition Manual.

Circle One: Logical ASLOC Physical ASLOC (carriage returns)
 Physical ASLOC (semicolons) Non-Commented/Non-Blank ASLOC
 Other: _____

3.17 Design Modified - DM. The percentage of design modified.

3.18 Code Modified - CM. The percentage of code modified.

3.19 Integration and Test - IM. The percentage of the adapted software's original integration & test effort expended.

3.20 Software Understanding - SU.

	Very Low	Low	Nom	High	Very High
Structure	Very low cohesion, high coupling, spaghetti code.	Moderately low cohesion, high coupling.	Reasonably well-structured; some weak areas.	High cohesion, low coupling.	Strong modularity, information hiding in data / control structures.
Application Clarity	No match between program and application world views.	Some correlation between program and application.	Moderate correlation between program and application.	Good correlation between program and application.	Clear match between program and application world-views.
Self-Descriptiveness	Obscure code; documentation missing, obscure or obsolete	Some code commentary and headers; some useful documentation.	Moderate level of code commentary, headers, documentations.	Good code commentary and headers; useful documentation; some weak areas.	Self-descriptive code; documentation up-to-date, well-organized, with design rationale.
SU Increment to ESLOC	50	40	30	20	10

Table 1: Rating Scale for Software Understanding Increment SU

The *Software Understanding* increment (SU) is obtained from Table 1. SU is expressed quantitatively as a percentage. If the software is rated very high on structure, applications clarity, and self-descriptiveness, the software understanding and interface checking penalty is 10%. If the software is rated very low on these factors, the penalty is 50%. SU is determined by taking the subjective average of the three categories. Enter the percentage of SU:

3.21 Assessment and Assimilation - AA.

AA Increment	Level of AA Effort
0	None
2	Basic module search and documentation
4	Some module Test and Evaluation (T&E), documentation
6	Considerable module T&E, documentation
8	Extensive module T&E, documentation

Table 2: Rating Scale for Assessment and Assimilation Increment (AA)

The other nonlinear reuse increment deals with the degree of *Assessment and Assimilation* (AA) needed to determine whether a fully-reused software module is appropriate to the application, and to integrate its description into the overall product description. Table 2 provides the rating scale and values for the assessment and assimilation increment. Enter the percentage of AA:

3.22 Programmer Unfamiliarity - UNFM.

UNFM Increment	Level of Unfamiliarity
0.0	Completely familiar
0.2	Mostly familiar
0.4	Somewhat familiar
0.6	Considerably familiar
0.8	Mostly unfamiliar
1.0	Completely unfamiliar

Table 3: Rating Scale for Programmer Unfamiliarity (UNFM)

The amount of effort required to modify existing software is a function not only of the amount of modification (AAF) and understandability of the existing software (SU), but also of the programmer's relative unfamiliarity with the software (UNFM). The UNFM parameter is applied multiplicatively to the software understanding effort increment. If the programmer works with the software every day, the 0.0 multiplier for UNFM will add no software understanding increment. If the programmer has never seen the software before, the 1.0 multiplier will add the full software understanding effort increment. The rating of UNFM is in Table 3. Enter the Level of Unfamiliarity:

Post-Architecture Cost Drivers.

These are the 17 effort multipliers used in the COCOMO II Post-Architecture model used to adjust the nominal effort, Person Months, to reflect the software product under development. They are grouped into four categories: product, platform, personnel, and project.

Product Cost Drivers.

For maintenance projects, identify any differences between the base code and modified code Product Cost Drivers (e.g. complexity).

	Very Low	Low	Nominal	High	Very High	Extra High
RELY	slight inconvenience	Low, easily recoverable losses	moderate, easily recoverable losses	high financial loss	risk to human life	
DATA		DB bytes/Pgm SLOC < 10	$10 \leq D/P < 100$	$100 \leq D/P < 1000$	$D/P \geq 1000$	
RUSE		None	across project	across program	across product line	across multiple product lines
DOCU	Many life-cycle needs uncovered	Some life-cycle needs uncovered.	Right-sized to life-cycle needs	Excessive for life-cycle needs	Very excessive for life-cycle needs	

3.23 Required Software Reliability (RELY). This is the measure of the extent to which the software must perform its intended function over a period of time. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.24 Data Size (DATA). This measure attempts to capture the affect large data requirements have on product development e.g. testing. The rating is determined by calculating D/P, where D is the number of bytes of data in the database at the end of (and necessary for) testing and P is the number of SLOC. See the Model Definition Manual for more details.

Low	Nominal	High	Very High	Don't Know

3.25 Develop for Reuse (RUSE). This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects. See the Model Definition Manual for more details.

Low	Nominal	High	Very High	Don't Know

3.26 Documentation match to life-cycle needs (DOCU). This captures the suitability of the project's documentation to its life-cycle needs. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.27 Product Complexity (CPLX):

	Control Operations	Computational Operations	Device-dependent Operations	Data Management Operations	User Interface Management Operations
Very Low	Straight-line code with a few non-nested structured programming operators: DOs, CASEs, IFTHENELSEs. Simple module composition via procedure calls or simple scripts.	Evaluation of simple expressions: e.g., $A=B+C*(D-E)$	Simple read, write statements with simple formats.	Simple arrays in main memory. Simple COTS-DB queries, updates.	Simple input forms, report generators.
Low	Straightforward nesting of structured programming operators. Mostly simple predicates	Evaluation of moderate-level expressions: e.g., $D=SQRT(B**2-4.*A*C)$	No cognizance needed of particular processor or I/O device characteristics. I/O done at GET/PUT level.	Single file subsetting with no data structure changes, no edits, no intermediate files. Moderately complex COTS-DB queries, updates.	Use of simple graphic user interface (GUI) builders.
Nominal	Mostly simple nesting. Some intermodule control. Decision tables. Simple callbacks or message passing, including middleware-supported distributed processing	Use of standard math and statistical routines. Basic matrix/vector operations.	I/O processing includes device selection, status checking and error processing.	Multi-file input and single file output. Simple structural changes, simple edits. Complex COTS-DB queries, updates.	Simple use of widget set.
High	Highly nested structured programming operators with many compound predicates. Queue and stack control. Homogeneous, distributed processing. Single processor soft real-time control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, roundoff concerns.	Operations at physical I/O level (physical storage address translations; seeks, reads, etc.). Optimized I/O overlap.	Simple triggers activated by data stream contents. Complex data restructuring.	Widget set development and extension. Simple voice I/O, multimedia.
Very High	Reentrant and recursive coding. Fixed-priority interrupt handling. Task synchronization, complex callbacks, heterogeneous distributed processing. Single-processor hard real-time control.	Difficult but structured numerical analysis: near-singular matrix equations, partial differential equations. Simple parallelization.	Routines for interrupt diagnosis, servicing, masking. Communication line handling. Performance-intensive embedded systems.	Distributed database coordination. Complex triggers. Search optimization.	Moderately complex 2D/3D, dynamic graphics, multimedia.
Extra High	Multiple resource scheduling with dynamically changing priorities. Microcode-level control. Distributed hard real-time control.	Difficult and unstructured numerical analysis: highly accurate analysis of noisy, stochastic data. Complex parallelization.	Device timing-dependent coding, micro-programmed operations. Performance-critical embedded systems.	Highly coupled, dynamic relational and object structures. Natural language data management.	Complex multimedia, virtual reality.

Complexity is divided into five areas: control operations, computational operations, device-dependent operations, data management operations, and user interface management operations. Select the area or combination of areas that characterize the product or a sub-system of the product. The complexity rating is the subjective weighted average of these areas.

Very Low	Low	Nominal	High	Very High	Extra High	Don't Know

Platform Cost Drivers. The platform refers to the target-machine complex of hardware and infrastructure software.

	Very Low	Low	Nominal	High	Very High	Extra High
TIME			= 50% use of available execution time	70%	85%	95%
STOR			=50% use of available storage	70%	85%	95%
PVOL		major change every 12 mo.; minor change every 1 mo.	major: 6 mo.; minor: 2 wk.	major: 2 mo.; minor: 1 wk.	major: 2 wk.; minor: 2 days	

3.28 Execution Time Constraint (TIME). This is a measure of the execution time constraint imposed upon a software system. See the Model Definition Manual for more details.

Nominal	High	Very High	Extra High	Don't Know

3.29 Main Storage Constraint (STOR). This rating represents the degree of main storage constraint imposed on a software system or subsystem. See the Model Definition Manual for more details.

Nominal	High	Very High	Extra High	Don't Know

3.30 Platform Volatility (PVOL). "Platform" is used here to mean the complex of hardware and software (OS, DBMS, etc.) the software product calls on to perform its tasks. See the Model Definition Manual for more details.

Low	Nominal	High	Very High	Don't Know

Personnel Cost Drivers.

	Very Low	Low	Nominal	High	Very High
ACAP	≤15th percentile	35th percentile	55th percentile	75th percentile	≥90th percentile
PCAP	≤15th percentile	35th percentile	55th percentile	75th percentile	≥90th percentile
PCON	≥48% / year	24% / year	12% / year	6% / year	≤3% / year
APEX	≤ 2 months	6 months	1 year	3 years	≥6 years
PEXP	≤ 2 months	6 months	1 year	3 years	≥6 years
LTEX	≤ 2 months	6 months	1 year	3 years	≥6 years

3.31 Analyst Capability (ACAP). Analysts are personnel that work on requirements, high level design and detailed design. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.32 Programmer Capability (PCAP). Evaluation should be based on the capability of the programmers as a team rather than as individuals. Major factors which should be considered in the rating are ability, efficiency and thoroughness, and the ability to communicate and cooperate. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.33 Applications Experience (APEX). This rating is dependent on the level of applications experience of the project team developing the software system or subsystem. The ratings are defined in terms of the project team's equivalent level of experience with this type of application. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.34 Platform Experience (PEXP). The Post-Architecture model broadens the productivity influence of PEXP, recognizing the importance of understanding the use of more powerful platforms, including more graphic user interface, database, networking, and distributed middleware capabilities. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.35 Language and Tool Experience (LTEX). This is a measure of the level of programming language and software tool experience of the project team developing the software system or subsystem. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

3.36 Personnel Continuity (PCON). The rating scale for PCON is in terms of the project's annual personnel turnover. See the Model Definition Manual for more details.

Very Low	Low	Nominal	High	Very High	Don't Know

Project Cost Drivers. This table gives a summary of the criteria used to select a rating level for project cost drivers.

	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	edit, code, debug	simple, frontend, backend CASE, little integration	basic lifecycle tools, moderately integrated	strong, mature lifecycle tools, moderately integrated	strong, mature, proactive lifecycle tools, well integrated with processes, methods, reuse	
SITE: Collocation	International	Multi-city and Multi-company	Multi-city or Multi-company	Same city or metro area	Same building or complex	Fully collocated
SITE: Communications	Some phone, mail	Individual phone, FAX	Narrowband email	Wideband electronic communications	Wideband electronic communications, occasional video conferencing.	Interactive multimedia
SCED	75% of nominal	85% of nominal	100% of nominal	130% of nominal	160% of nominal	

3.37 Use of Software Tools (TOOL). See the Model Definition Manual.

Very Low	Low	Nominal	High	Very High	Don't Know

3.38 Multisite Development (SITE). Given the increasing frequency of multisite developments, and indications that multisite development effects are significant, the SITE cost driver has been added in COCOMO II. Determining its cost driver rating involves the assessment and averaging of two factors: site collocation (from fully collocated to international distribution) and communication support (from surface mail and some phone access to full interactive multimedia). See the Model Definition Manual for more details.

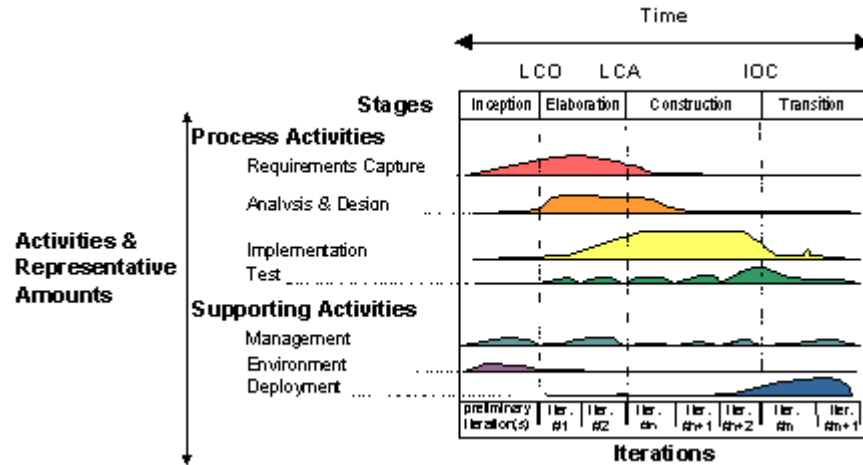
Very Low	Low	Nominal	High	Very High	Extra High	Don't Know

3.39 Required Development Schedule (SCED). This rating measures the schedule constraint imposed on the project team developing the software. The ratings are defined in terms of the percentage of schedule stretch-out or acceleration with respect to a nominal schedule for a project requiring a given amount of effort. See the Model Definition Manual for more details.

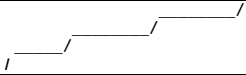
Very Low	Low	Nominal	High	Very High	Don't Know

Appendix A

The lifecycle anchoring concepts are discussed by Boehm⁵. The anchor points are defined as Life Cycle Objectives (LCO), Life Cycle Architecture (LCA), and Initial Operational Capability (IOC). An enhanced version of an illustration from Rational Corporation⁶ showing the phases around the anchor points is shown below.



The correspondence between phases, COCOMO II's submodels and the life cycle anchor points is shown in the following table for the MBASE spiral lifecycle model. The table also has indications of the relative amounts of the different activities.

COCOMO II Submodel Usage	Early Design 			Maintenance
	LCO		LCA	IOC
Activities \ Phase	Inception	Elaboration	Construction	Transition
Requirements Capture	Some usually	Most, peaks here	Minor	None
Analysis & Design	A little	Majority, mostly constant effort	Some	Some, for repair during ODT&E
Implementation	Practically none	Some, usually for risk reduction	Bulk; mostly constant effort	Some, for repair during ODT&E
Test	None	Some, for prototypes	Most for unit, integration and qualification test.	Some, for repaired code.

COCOMO II's effort and schedule estimates are focused on Elaboration and Construction (the phases between LCO and IOC). Inception corresponds to the COCOMO's "Requirements" activity in a waterfall process model. COCOMO's effort for the "Requirements" activity is an additional, fixed percentage of the effort calculated by COCOMO for the development activities. The table also indicates the areas in which the COCOMO II Early Design and Post-Architecture submodels are normally used.

⁵ Barry W. Boehm, "Anchoring the Software Process," *IEEE Software*, 13, 4, July 1996, pp. 73-82

⁶ Rational Corp., "Rational Objectory Process 4.1 – Your UML Process", available at <http://www.rational.com/support/techpapers/toratojprcs/>.