Appendices

Appendix A

Projects Data Collection Form

The actual cost and parameters of completed projects
Object Oriented C++ Projects (console applications)

Software House Name: _____

Name of responsible person (who filled this form): _____

Telephone: _____ **Fax:**

Organization Labor rate _____

Software Project Name: _____

Number of hours for a Person Month: _____ Hours

Number of Person Month to finish this project _____ PM

Mark the suitable scales of the project:

Factors	Very Low	Low	Nominal	High	Very High	Extra High
1. Previous experience with this type of project	☐ (No previous experience)	☐ Largely Unprecedented	☐ Somewhat Unprecedented	☐ Generally Familiar	☐ Largely Familiar	☐ (Completely familiar with this application domain)
2. Development flexibility	☐ (Rigorous)	☐ Occasional Relaxation	☐ Some Relaxation	☐ General Conformity	☐ Some Conformity	☐ (The client sets only general goals)
3. The extent of risk analysis carried out	☐ Little (20%)	☐ Some (40%)	☐ often (60%)	☐ Generally (75%)	☐ mostly (90%)	☐ full (100%)
4. How well the development team knows each other and work together.	☐ (Very difficult interaction)	☐ Some difficult interaction	☐ Basically Cooperative Interactions	☐ Largely Cooperative	☐ Highly Cooperative	☐ (Integrated and effective team with no communication problems)
5. Process maturity of the organization	☐ Initial level (The organization does not have effective management procedure)	☐ Repeatable level (The organization has formal management procedure)	☐ Defined level (the organization has defend its process)	☐ Managed level (the organization has defined process and a formal program of quantitative data collection.)		☐ Optimizing level (the organization is committed to continuous process improvement.)

Fill the actual size attribute of the project:

Total number of source code _____ LOC

(Blanks and comments lines are excluded)**:**

Mark the suitable scales of the project cost drivers:

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
1.Required system reliability	☐ Slight inconvenience	☐ Low, easily recoverable losses	☐ Moderate, easily recoverable losses	☐ High financial loss	☐ Risk to human life	☐

Help information - Module reliability vs. project activity

Rating	Rqts. And product design	Detailed design	Code and unit test	Integration and test
Very low	Little detail Little verification Minimal QA, CM, Draft user manual, test plan	Basic design information. Minimal QA, CM, Draft user manual, test plan Informal design inspections	No test procedures Minimal path test, standards check Minimal QA, CM Minimal I/O & off-nominal tests Minimal user manual	No test procedures Many requirements untested Minimal QA, CM Minimal stress, off-nominal tests Minimal as-built documentation
Low	Basic information, verification Basic QA, CM, standards, Draft user manual, test plan	Moderate detail Basic QA, CM, Draft user manual, test plan	Minimal test procedures Partial path test, standards check Basic QA, CM, user manual Partial I/O & off-nominal tests	Minimal test procedures Frequent requirements untested Basic QA, CM, user manual Partial stress, off-nominal
Nominal	Nominal Project V&V →			
High	Detailed verification, QA, CM, standards, documentation Detailed test plan, procedures	Detailed verification, QA, CM, standards, documentation Detailed test plan, procedures	Detailed test procedures, QA, CM, documentation Extensive off-nominal tests	Detailed test procedures, QA, CM, documentation Extensive stress, off-nominal tests
Very High	Detailed verification, QA, CM, standards, documentation IV & V interface Very Detailed test plan, procedures	Detailed verification, QA, CM, standards, documentation Very through design inspections Very Detailed test plan, procedures IV & V interface	Detailed test procedures, QA, CM, documentation Very through code inspections Very Extensive off-nominal tests IV & V interface	Very Detailed test procedures, QA, CM, documentation Very Extensive stress, off-nominal tests IV & V interface

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
2.Complexity of system modules	☐	☐	☐	☐	☐	☐

Help information - Module Complexity vs. Type of Module

SP: structured programming

Rating	Control operations	Computational operations	Device-dependent operations	Data management operations
Very low	Straight-line code with a few non nested SP operators: Dos, Case, IF Then Else, simple predicates	Evaluation of simple expression. e.g. A=B+C*(D-E)	Simple read write statements with simple formats.	Simple array in main memory
Low	Straightforward nesting of SP operators, mostly simple predicates.	Evaluation of moderate-level expression e.g. D=SQRT(B^2-4*A*C)	No cognizance needed of particular processor or I/O device characteristic. I/O done at Get/Put level. No cognizance of overlap.	Single file sub setting with no data structure changes, no edit, no intermediate files
Nominal	Mostly simple nesting. Some inter module control. Decision tables.	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.
High	Highly nesting of SP operators with many compound predicates. Queue and stack control. Considerable inter module control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round off concerns.	Operations at physical I/O level. Optimized I/O overlaps.	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level.
Very High	Reentrant and recursive coding. Fixed priority interrupt handling.	Difficult but structured N.A.: near singular matrix equations, partial differential equations.	Routines for interrupt diagnosis, servicing, masking, communication line handling.	A generalized, parameter-driven file structuring routine. File building, command processing, search optimization.
Extra High	Multiple resource scheduling with dynamically changing priorities. Micro code-level control.	Difficult and unstructured N.A.: highly accurate analysis of noisy, stochastic data	Device timing-dependent coding, micro-programmed operations.	Highly coupled, dynamic relational structure. Natural language data management.

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product attributes						
3.Size of database used	☐	☐ DB bytes/Pgm SLOC< 10	☐ 10 < D/P < 100	☐ 100 < D/P < 1000	☐ D/P > 1000	☐
Note: 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 and P is the number of SLOCS in the program.						
4.Extent of documentation required	☐ Many life-cycle Needs uncovered	☐ Some life-cycle needs uncovered.	☐ Right-sized to lifecycle needs	☐ Excessive for lifecycle needs	☐ Very excessive for life-cycle needs	☐
Note: This cost driver captures the suitability of the project's documentation to its life-cycle needs.						
5.Required percentage of reusable components	☐	☐ None	☐ Across project	☐ Across program	☐ Across product line	☐ Across multiple product lines
Note: This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects.						
Computer attributes						
6. Execution time constraints	☐	☐	☐ <= 50% use of available execution time.	☐ 70%	☐ 85%	☐ 95%
7. Memory constraints	☐	☐	☐ <= 50% use of available storage	☐ 70%	☐ 85%	☐ 95%
8. Volatility of development platform.	☐	☐ Major change	☐ Major change	☐ Major change	☐ Major change	☐

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
		every 12 mo.; Minor change every 1 mo.	every 6 mo.; Minor change every 2 wk.	every 2 mo.; Minor change every 1 wk.	every 2 wk.; Minor change every 2 days.	
Note: The platform refers to the target-machine complex of hardware and infrastructure software.						
Personal attributes						
9. Capability of project analysts (efficiency, ability to communicate and cooperate)	☐ ≤15th percentile	☐ 35th percentile	☐ 55th percentile	☐ 75th percentile	☐ ≥90th percentile	☐
10. Programmer Capability	☐ ≤15th percentile	☐ 35th percentile	☐ 55th percentile	☐ 75th percentile	☐ ≥90th percentile	☐
11. Personal continuity	☐ ≥48% / year	☐ ≥24% / year	☐ ≥12% / year	☐ ≥6% / year	☐ ≥3% / year	☐
12. Analyst experience in project domain	☐ <= 2 months	☐ 6 months	☐ 1 year	☐ 3 year	☐ >=6 year	☐
13. Programmer experience in project domain	☐ <= 2 months	☐ 6 months	☐ 1 year	☐ 3 year	☐ >=6 year	☐
14. Language and tool experience.	☐ <= 2 months	☐ 6 months	☐ 1 year	☐ 3 year	☐ >=6 year	☐

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Project attributes						
15. Use of software tools	☐ Basic microprocessor tools.	☐ Basic Mini tools	☐ Basic midi/maxi tools	☐ Strong maxi programming, test tools.	☐ Add requirements, design, management, and documentation tools.	☐
16. Extent of multi-site working and quality of site communications	☐ Site collocation (international), site communications (some phone, mail)	☐ Site collocation (multi-city and multi-company), site communications (individual phone, Fax)	☐ Site collocation (multi-city and multi-company), site communications (narrow band email)	☐ Site collocation (Same-city or metro area), site communications (wideband electronic communication)	☐ Site collocation (Same building or complex), site communications (wideband electronic communication, occasional video conf.)	☐ Site collocation (Fully collocated), site communications(Interactive multi-media)
17. Development schedule compression	☐ A schedule compression of 75% of nominal	☐ A schedule compression of 85% of nominal	☐ A schedule is 100% of nominal	☐ A schedule stretch-out of 130% of nominal	☐ A schedule stretch-out of 160% of nominal	☐
Note : This rating measures the schedule constraint imposed on the project team developing the software.						

Mark the suitable scales of 14 General System Characteristics (GSCs)

Note:

0 Not present, or no influence
1 Incidental influence
2 Moderate influences
3 Average influences
4 Significant influences
5Strong influence throughout

GSCs	Brief Description	0	1	2	3	4	5
1. Data communications	How many communication facilities are there to aid in the transfer or exchange of information with the application or system?						
2. Distributed data processing	How are distributed data and processing functions handled?						
3. Performance	Did the user require response time or throughput?						
4. Heavily used configuration	How heavily used is the current hardware platform where the application will be executed?						
5. Transaction rate	How frequently are transactions executed daily, weekly, monthly, etc.?						
6. On-Line data entry	What percentage of the information is entered On-Line?						
7. End-user efficiency	Was the application designed for end-user efficiency?						
8. On-Line update	How many internal information objects are updated by On-Line transaction?						
9. Complex processing	Does the application have extensive logical or mathematical processing?						
10. Reusability	Was the application developed to meet one or many user's needs?						
11. Installation ease	How difficult is conversion and installation?						
12. Operational ease	How effective and/or automated are start-up, back up, and recovery procedures?						
13. Multiple sites	Was the application specifically designed, developed, and supported to be installed at multiple sites for multiple organizations?						
14. Facilitate change	Was the application specifically designed, developed, and supported to facilitate change?						

Appendix B
The Data of 10 historical software projects

Appendix C
Help information for
General System Characteristics GSCs

Help information for
General System Characteristics (GSCs)

Each General System Characteristic (GSC) must be evaluated in terms of its Degree of Influence (DI) on a scale of zero to five:

0 Not present, or no influence

1 Incidental influence

2 Moderate influence

3 Average influence

4 Significant influence

5 Strong influence throughout

There are IFPUG guidelines for assigning the Degree of Influence (DI) for each General System Characteristic (GSC). The remainder of this chapter will provide the IFPUG guidelines for assigning values to each of the fourteen GSCs. Additionally, the authors' comments will assist in assigning these values to different types of applications. If none of the IFPUG guideline descriptions fit the application exactly, a judgment must be made about which Degree of Influence (DI) most closely applies to the application. Our discussion of the fourteen GSCs follows.

1. Data Communications. Data and control information used in the application are sent or received over communication facilities. Terminals connected locally to the control unit are considered to use communication facilities. Protocol is a set of conventions which permit the transfer or exchange of information between two systems or devices. All data communication links require some type of protocol. Score as follows:

1. Application is pure batch processing or a stand alone PC.
2. Application is batch but has remote data entry or remote printing.
3. Application is batch but has remote data entry and remote printing.
4. On-line data collection or TP (teleprocessing) front end to a batch process or query system.
5. More than a front-end, but the application supports only one type of TP communications protocol.

6. More than a front-end, but the application supports more than one type of TP communications protocol.

Authors' notes: We would expect only batch applications with no interactivity to be valued at zero. Most applications, stand-alone PC, as well as batch, have remote data entry as well as printing capability. Those applications that have front end data entry screens, but which update internal logical files through a batch process, should be scored at three. If update occurs immediately, score four. In order to score five there must be multiple types of telecommunication protocol.

2. Distributed Data Processing. Distributed data or processing functions are a characteristic of the application within the application boundary. Score as follows:

0. Application does not aid the transfer of data or processing function between components of the system.

1. Application prepares data for end-user processing on another component of the system such as PC spreadsheets and PC DBMS.

2. Data is prepared for transfer, transferred, and processed on another component of the system (not for end-user processing).

3. Distributed processing and data transfer are on-line and in one direction only.

4. Distributed processing and data transfer are on-line and in both directions.

5. Processing functions are dynamically performed on the most appropriate component of the system.

Authors' notes: Only distributed or real-time applications would be assigned a value within this category. Most applications score zero; primitive distributed applications could score one or two; client server scores a two to four; real-time, telecommunication, or process control systems could score zero through five. in order to score a five there must be multiple servers or processors.

3. Performance. Application performance objectives, stated or approved by the user, in either response or throughput, influenced (or will influence) the design, development installation, and support of the application. Score as follows:

0. No special performance requirements were stated by the user.

1. Performance and design requirements were stated and reviewed but no special actions were required.

2. Response time or throughput is critical during peak hours. No special design for CPU utilization was required. Processing deadline is for the next business day.
3. Response time or throughput is critical during all business hours. No special design for CPU utilization was required. Processing deadline requirements with interfacing systems are constraining.
4. Stated user performance requirements are stringent enough to require performance analysis tasks in the design phase.
5. In addition, performance analysis tools were used in the design, development, and/or implementation phases to meet the stated user performance requirements.
Authors' notes: GSC 3 and GSC 5 are very similar in nature; both require consideration of performance during the design, development, and installation phases of development. Response time typically relates to interactive processing; throughput relates to batch processing. Consider the significance of performance to the particular application. A score of four requires performance analysis tasks during the design phase. A score of five requires the use of performance analysis tools. Typically, batch receives a score of zero to four; on-line scores zero to four; real-time, telecommunication, or process control systems score zero to five.

4. Heavily Used Configuration. A heavily used operational configuration, requiring special design considerations, is a characteristic of the application; for example, the user wants to run the application on existing or committed equipment that will be heavily used. Score as follows:
0. There are no explicit or implicit operational restrictions.
1. Operational restrictions do exist, but are less restrictive than a typical application. No special effect is needed to meet the restrictions.
2. Some security or timing considerations exist.
3. There are specific processor requirements for a specific piece of the application.
4. Stated operation restrictions require special constraints on the application in the central processor or a dedicated processor.
5. In addition, there are special constraints on the application in the distributed components of the system.
Authors' notes: We would expect most applications to be valued at two. In order to score

three through five, you would expect to have a client server, real-time, telecommunication, or process control system. Even then, you would need either a dedicated processor or multiple processors processing the same transactions and searching for the most expeditious means of processing.

5. Transaction Rate. The transaction rate is high and it influenced the design, development, installation, and support of the application. Score as follows:

0. No peak transaction period is anticipated.

1. A peak transaction period (monthly, quarterly, seasonally, annually) is anticipated.

2. A weekly peak transaction period is anticipated.

3. A daily peak transaction period is anticipated.

4.High transaction rates stated by the user in the application requirement or service level agreements are high enough to require performance analysis tasks in the design phase.

5. High transaction rates stated by the user in the application requirements or service level agreements are high enough to require performance analysis tasks and, in addition, require the use of performance analysis tools in the design, development, and/or installation phases.

Authors' notes: GSC 3 and GSC 5 are very similar in nature; both require consideration of performance during the design, development, and installation phases of development. Consider the significance of transaction rates to the particular application. A score of four requires performance analysis tasks during the design phase. A score of five requires the use of performance analysis tools. Typically, batch receives a score of zero to three; on-line scores zero to four, real-time, telecommunication, or process control systems score zero to five.

6. On-Line Data Entry. On-line data entry and control functions are provided in the application. Score as follows:

0. All transactions are processed in batch mode.

1. 1% to 7% of transactions are interactive data entry.

2. 8% to 15% of transactions are interactive data entry.

3. 16% to 23% of transactions are interactive data entry.

4. 24% to 30% of transactions are interactive data entry.

5. Over 30% of transactions are interactive data entry.

Authors' notes: One of the major problems with the scoring for GSCs is that the guidelines have not been updated in years. Consequently, these scores are not realistic. Nevertheless, industry data has been calculated using these guidelines. Typically, batch receives a score of zero to one; on-line, real-time, telecommunication, or process control systems score a five.

7. End-user Efficiency. The on-line functions provided emphasize a design for end-user efficiency. They include the following:

* Navigational aids (for example, function keys, jumps, dynamically generated menus)
* menus
* On-line help/documentation
* Automated cursor movement
* Scrolling
* Remote printing (via on-line transactions)
* Preassigned function keys
* Submission of batch jobs from on-line transactions
* Cursor selection of screen data
* Heavy use of reverse video, highlighting, colors, underlining and other indicators
* Hard copy user documentation of on-line transactions
* Mouse interface
* Popup windows
* As few screens as possible to accomplish a business function
* Bilingual support (supports two languages; count as four items)
* Multilingual support (supports more than two languages; count as six items)

Score as follows:

0. None of the above.
1. One to three of the above.
2. Four to five of the above.
3. Six or more of the above but their are no specific user requirements related to efficiency.
4. Six or more of the above and the requirements for end-user efficiency are strong enough to require design tasks for human factors to be included (for example, minimize

key strokes, maximize defaults, use of templates, etc..).

5. Six or more of the above and stated requirements for end-user efficiency are strong enough to require use of special tools and processes in order to demonstrate that the objectives have been achieved.

Authors' notes: We would expect batch applications with no interactivity to be valued at zero. Most interactive applications have front-end data entry screens, but unless they have templates/defaults built into the application they should be scored at three. If defaults or templates or significant navigational tools are present score four. In order to score five, there must be user labs to test the useability of the application, rather than the functionality. Real-time, telecommunication, or process control systems may not score anything for this GSC.

8. On-Line Update. The application provides on-line update for the Internal Logical Files. Score as follows:

0 None.

1. On-line update of one to three control files. Volume of updating is low, and recovery is easy.

2. On-line update of four or more control files. Volume of updating is low, and recovery is easy.

3. On-line update of major internal logical files.

4. In addition, protection against data loss is essential and has been specially designed and programmed in the system.

5. In addition, high volumes bring cost considerations into the recovery process. Highly automated recovery procedures with minimum of operator intervention. Authors' notes: We would expect batch applications with no interactive update of internal logical files to be valued at zero to two. Most on-line applications update internal logical files and should be scored at three or higher. If protection of data loss has been programmed into the system (not just through back-ups), score four. In order to score five, there must be a highly automated recovery capability built within the application. Real-time, telecommunication, or process control systems often receive a score of four or five.

9. Complex Processing. Complex processing is a characteristic of the application. The categories include the following:

* Sensitive control (for example, special audit processing) and/or application specific security processing
* Extensive logical processing
* Extensive mathematical processing
* Much exception processing resulting in incomplete transactions that must be processed again; for example, incomplete ATM transactions caused by TP interruption, missing data values, or failed edits.
* Complex processing to handle multiple input/output possibilities; for example, multi-media device independence.

Score this characteristic as follows:

0. None of the above.
1. Any one of the above.
2. Any two of the above.
3. Any three of the above.
4. Any four of the above.
5. All five of the above.

Authors' notes: With the prior GSCs, each individual guideline provided a little more than the previous score. This GSC credits five separate and individual characteristics. First, does the application provide security such that certain individuals see or enter data that others cannot? Second, is there a significant amount of logical (if, then, else) processing? Third, is there extensive mathematical (more than addition subtraction/simple math) processing? Fourth, is there complex editing or validation? Fifth, are there multiple media included in the application?

10. Reusability. The application and the code in the application have been specifically designed, developed, and supported to be usable in other applications. Score as follows:

0. There is no reusable code.
1. Reusable code is used within the application.
2. Less than 10% of the application considered more than one user's needs.
3. Ten percent or more of the application considered more than one user's needs.
4. The application was specifically packaged and/or documented to ease reuse, and application is customized to user at source code level.

5. The application was specifically packaged and/or documented to ease reuse, and application is customized to use at source code level by means of user parameter maintenance.

Authors' notes: Function Point counting gives credit here with a score of one to those who reuse code. Standardized reusable software provides increased user/owner functionality through increased reliability and consistency. Scores of two through five am assigned based on the resulting functionality and due to the extra effort dedicated to the development, documentation, and testing of code expected to be used in other applications.

11. Installation Ease. Conversion and installation ease are characteristics of the application. A conversion and installation plan and/or conversion tools were provided and tested during the system test phase. Score as follows:

0. No special considerations were stated by user, and no special set-up is required for installation.

1. No special considerations were stated by user, but special set-up is required for installation.

2. Conversion and installation requirements were stated by the user, and conversion and installation guides were provided and tested. The impact of conversion on the project is not considered to be important

3. Conversion and installation requirements were stated by the user, and conversion and installation guides were provided and tested. The impact of conversion on the project is considered to be important.

4. In addition to (2), automated conversion and installation tools were provided and tested.

5. In addition to (3), automated conversion and installation tools were provided and tested.

Authors' notes: Developers are often required to devote significant effort to the conversion of pre-existing data into new data files or to populate the files with actual data or to develop installation software, such as porting. Functional advantages occur to the users through improved schedules and increased consistency. Consider the difficulty or

ease of conversion and installation requirements, and assign the score in relationship to their significance.

12. Operational Ease. Operational ease is characteristic of the application. Effective start-up, back-up, and recovery procedures were provided and tested during the system test phase. The application minimizes the need for manual activities, such as tape mounts, paper handling, and direct on-location manual intervention. Score as follows:

0. No special operational consideration other than the normal back-up procedures were stated by the user.

1-4 Select the following items that apply to the application. Each item has a point value of one, except as noted otherwise:

* Effective start-up, back-up, and recovery processes were provided but operator intervention is required
* Effective start-up, back-up, and recovery processes were provided but no operator intervention is required (count as two items).
* The application minimizes the need for tape mounts.
* The application minimizes the need for paper handling.
* Application is designed for unattended operation. Unattended operation means no operator intervention is required to operate the system other than to start up or shut down the application. Automatic error recovery is a feature of the application.

Authors' notes: Unless we are counting a legacy system, we should score one each for the lack of tape mounts and the lack of paper (punched cards, punched paper tapes). We should count three if operator intervention is required for start-up, backup, and recovery. Four would be assigned if no operator intervention is required. Five is assigned to an application that runs and recovers automatically from errors on its own-a lights-out operation.

13. Multiple Sites. The application has been specifically designed, developed, and supported to be installed at multiple sites for multiple organizations. Score as follows:

0. There is no user requirement to consider the needs of more than one user/installation site.

1. Needs of multiple sites were considered in the design, and the application is designed to operate only under identical hardware and software environments.

2. Needs of multiple sites were considered in the design, and the application is designed to operate only under similar hardware and/or software environments.
3. Needs of multiple sites were considered in the design, and the application is designed to operate under different hardware and/or software environments.
4. Documentation and support plan are provided and tested to support the application at multiple sites, and application is as described by (1) or (2).
5. Documentation and support plan are provided and tested to support the application at multiple sites, and application is as described by (3).
Authors' notes: We consider within this characteristic the increased user functionality and the effort required to deliver an application which will include software and/or hardware installable at multiple sites; this could reflect just the input devices, such as terminals or PCs. Is the software/hardware identical, similar, or different? Are documentation support plans to be provided and tested?
14. Facilitate Change. The application has been specifically designed, developed, and supported to facilitate change. Examples include the following:
* Flexible query/report capability is provided.
* Business control data is grouped in tables maintainable by the user.
Score as follows:
0. There is no special user requirement to design the application to minimize or facilitate change.
1-5 Select which of the following items apply to the application:
* Flexible query/report facility is provided that can handle simple requests; for example, and/or logic applied to only one internal logical file (count as one item).
* Flexible query/report facility is provided that can handle requests of average complexity; for example and/or logic applied to more than one internal logical file (count as two items).
* Flexible query/report facility is provided that can handle complex requests; for example, and/or logic combinations on one or more internal logical files (count as three items).
* Control data is kept in tables that we maintained by the user with on-line interactive processes, but changes take effect only on the next business day.

* Control data is kept in tables that are maintained by the user with on-line interactive processes, and the changes take effect immediately (count as two items).

Authors' notes. We address two separate categories in this GSC, much the same as GSC 9 had five separate categories. The first area with query/report writer capability often provided by languages such as SQL or Focus; scores of zero to three are assigned to this particular characteristic, which is becoming more popular to computer literate users. The second area and the last two questions relate to the interactivity in which data and/or control is maintained within/by the application. Interactive, real-time, telecommunication, or process control systems would typically be counted with the last two points.

Appendix D
COCOMOII
1- Post-Architecture Model Cost drivers
2- Early Design Model Cost drivers

1- Post-Architecture Model Cost drivers

Product Factors

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.

	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	

Table 1: Required Software Reliability (RELY

Product Complexity (CPLX)

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.

Rating	Control operations	Computational operations	Device-dependent operations	Data management operations
Very low	Straight-line code with a few non nested SP operators: Dos, Case, IF Then Else, simple predicates	Evaluation of simple expression. e.g. A=B+C*(D-E)	Simple read write statements with simple formats.	Simple array in main memory
Low	Straightforward nesting of SP operators, mostly simple predicates.	Evaluation of moderate-level expression e.g. D=SQRT(B^2-4*A*C)	No cognizance needed of particular processor or I/O device characteristic. I/O done at Get/Put level. No cognizance of overlap.	Single file sub setting with no data structure changes, no edit, no intermediate files

Rating	Control operations	Computational operations	Device-dependent operations	Data management operations
Nominal	Mostly simple nesting. Some inter module control. Decision tables.	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.
High	Highly nesting of SP operators with many compound predicates. Queue and stack control. Considerable inter module control.	Basic numerical analysis: multivariate interpolation, ordinary differential equations. Basic truncation, round off concerns.	Operations at physical I/O level. Optimized I/O overlaps.	Special purpose subroutines activated by data stream contents. Complex data restructuring at record level.
Very High	Reentrant and recursive coding. Fixed priority interrupt handling.	Difficult but structured N.A.: near singular matrix equations, partial differential equations.	Routines for interrupt diagnosis, servicing, masking, communication line handling.	A generalized, parameter-driven file structuring routine. File building, command processing, search optimization.
Extra High	Multiple resource scheduling with dynamically changing priorities. Micro code-level control.	Difficult and unstructured N.A.: highly accurate analysis of noisy, stochastic data	Device timing-dependent coding, micro-programmed operations.	Highly coupled, dynamic relational structure. Natural language data management.

Table 2: Module Complexity Ratings versus Type of Module

Data Base Size (DATA)

This measure attempts to capture the affect large data requirements have on product development. The rating is determined by calculating D/P. The reason the size of the database is important to consider it because of the effort required to generate the test data that will be used to exercise the program.

	Very Low	Low	Nominal	High	Very High	Extra High
DATA		DB bytes/Pgm SLOC< 10	10 < D/P < 100	100 < D/P < 1000	D/P > 1000	

Table 3: Data Base Size (DATA

Required Reusability (RUSE)

This cost driver accounts for the additional effort needed to construct components intended for reuse on the current or future projects. This effort is consumed with creating

more generic design of software, more elaborate documentation, and more extensive testing to ensure components are ready for use in other applications.

	Very Low	Low	Nominal	High	Very High	Extra High
RUSE		None	Across project	Across program	Across product line	Across multiple product lines

Table 4: Required Reusability (RUSE

Documentation match to life-cycle needs (DOCU)

Several software cost models have a cost driver for the level of required documentation. In COCOMO II, the rating scale for the DOCU cost driver is evaluated in terms of the suitability of the project's documentation to its life-cycle needs.

	Very Low	Low	Nominal	High	Very High	Extra High
DOCU	Many life-cycle Needs uncovered	Some life-cycle needs uncovered.	Right-sized to lifecycle needs	Excessive for lifecycle needs	Very excessive for life-cycle needs	

Table 5: Documentation match to life-cycle needs (DOCU)

Platform Factors

Execution Time Constraint (TIME)

This is a measure of the execution time constraint imposed upon a software system. The rating is expressed in terms of the percentage of available execution time expected to be used by the system or subsystem consuming the execution time resource.

	Very Low	Low	Nominal	High	Very High	Extra High
TIME			<= 50% use of available execution time.	70%	85%	95%

Table 5: Execution Time Constraint (TIME)

Main Storage Constraint (STOR)

This rating represents the degree of main storage constraint imposed on a software system or subsystem.

	Very Low	Low	Nominal	High	Very High	Extra High
STOR			<= 50% use of available storage	70%	85%	95%

Table 6: Main Storage Constraint (STOR)

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. If the software to be developed is an operating system then the platform is the computer hardware. If a database management system is to be developed then the platform is the hardware and the operating system. If a network text browser is to be developed then the platform is the network, computer hardware, the operating system, and the distributed information repositories.

	Very Low	Low	Nominal	High	Very High	Extra High
PVOL		Major change every 12 mo.; Minor change every 1 mo.	Major change every 6 mo.; Minor change every 2 wk.	Major change every 2 mo.; Minor change every 1 wk.	Major change every 2 wk.; Minor change every 2 days.	

Table 7: Platform Volatility (PVOL)

Personnel Factors

Analyst Capability (ACAP)

The major attributes that should be considered in this rating are Analysis and Design ability, efficiency and thoroughness, and the ability to communicate and cooperate. The rating should not consider the level of experience of the analyst;

	Very Low	Low	Nominal	High	Very High	Extra High
ACAP	≤15th percentile	35th percentile	55th percentile	75th percentile	≥90th percentile	

Table 8: Analyst Capability (ACAP))

Programmer Capability (PCAP)

The major attributes that should be considered in this rating are development ability, efficiency and thoroughness, and the ability to communicate and cooperate. The rating should not consider the level of experience of the programmer;

	Very Low	Low	Nominal	High	Very High	Extra High
PCAP	≤15th percentile	35th percentile	55th percentile	75th percentile	≥90th percentile	

Table 8: Programmer Capability (PCAP)

Personnel Continuity (PCON)

The rating scale for PCON is in terms of the project's annual personnel turnover: from 3%, very high, to 48%, very low.

	Very Low	Low	Nominal	High	Very High	Extra High
PCON	≥48% / year	≥24% / year	≥12% / year	≥6% / year	≥3% / year	

Table 9: Personnel Continuity (PCON)

Analyst Experience (AEXP)

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.

	Very Low	Low	Nominal	High	Very High	Extra High
AEXP	<= 2 months	6 months	1 year	3 year	>=6 year	

Table 10: Analyst Experience (AEXP)

Programmer 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.

	Very Low	Low	Nominal	High	Very High	Extra High
PEXP	<= 2 months	6 months	1 year	3 year	>=6 year	

Table 11: Programmer Experience (PEXP)

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.

	Very Low	Low	Nominal	High	Very High	Extra High
LTEX	<= 2 months	6 months	1 year	3 year	>=6 year	

Table 12: Language and Tool Experience (LTEX)

Project Factors

Use of Software Tools (TOOL)

Software tools have improved significantly since the 1970's projects used to calibrate COCOMO. The tool rating ranges from simple edit and code, very low, to integrated lifecycle management tools, very high.

	Very Low	Low	Nominal	High	Very High	Extra High
TOOL	Basic microprocessor tools.	Basic Mini tools	Basic midi/maxi tools	Strong maxi programming, test tools.	Add requirements, design, management, and documentation tools.	

Table 13: Use of Software Tools (TOOL)

Multisite Development (SITE)

Determining This 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).

	Very Low	Low	Nominal	High	Very High	Extra High
SITE	Site collocation (international), site communications (some phone, mail)	Site collocation (multi-city and multi-company), site communications (individual phone, Fax)	Site collocation (multi-city and multi-company), site communications (narrow band email)	Site collocation (Same-city or metro area), site communications (wideband electronic communication)	Site collocation (Same building or complex), site communications (wideband electronic communication, occasional video conf.)	Site collocation (Fully collocated), site communications (Interactive multi-media)

Table 14: Multisite Development (SITE)

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. Accelerated schedules tend to produce more effort in the later phases of development because more issues are left to be determined due to lack of time to resolve them earlier.

	Very Low	Low	Nominal	High	Very High	Extra High
SCED	A schedule compression of 75% of nominal	A schedule compression of 85% of nominal	A schedule is 100% of nominal	A schedule stretch-out of 130% of nominal	A schedule stretch-out of 160% of nominal	

Table 15: Required Development Schedule (SCED)

2- Early Design Model Cost drivers

Cost Driver	Rating						
	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
RCPX	0.49	0.6	0.83	1	1.33	1.91	2.72
RUSE			0.91	1.00	1.14	1.29	1.49
PDIF			0.87	1	1.29	1.81	2.61

PERS	2.12	1.62	1.26	1	0.83	0.63	0.5
PREX	1.59	1.33	1.22	1	0.87	0.74	0.62
FCIL	1.43	1.3	1.1	1	0.87	0.73	0.62
SCED		1.29	1.10	1.00	1.00	1.00	-

Table 16: Early Design cost drivers rating values

Appendix E
Classes and methods definition
of the collected 10 historical software projects

Project NO 1 Airline Reservation

```
class  Traveler
{
public:
  char Name[MaxName];
private:
};
```

```
class  QueueNode
{
public:
  Traveler   data;
  QueueNode  *next;
private:
};
```

```
class  ListNode
{
public:
  Traveler   data;
  ListNode   *next;
private:
};
```

```
template <class T>
class stackclass
{
public:
              int   top;
              T Flights[MaxFlight];
              InitStack(stackclass & s);
      DestroyStack(stackclass & s);
              bool StackIsEmpty(stackclass s);
      bool StackIsFull(stackclass s);
      Push(stackclass & s,  T Node);
      Pop(stackclass & s,T & Node);
private:
};
```

```
template <class  T>
class queueclass
{
public:
              QueueNode  *front,*rear;
              InitQueue(queueclass & q);
      DestroyQueue(queueclass & q);
              bool QueueIsEmpty(queueclass q);
      bool QueueIsFull(queueclass q);
              bool Search(queueclass q,T D);
      EnQueue(queueclass & q,  T D);
      DeQueue(queueclass & q,  T & D);
private:
};
```

```
template <class T>
class listclass
{
public:
                ListNode  *head,*current;
                InitList(listclass & L);
        DestroyList(listclass & L);
                bool ListIsEmpty(listclass L);
        bool ListIsFull(listclass L);
        Insert(listclass & L,  T Data);
        Delete(listclass & L,  T Data);
                bool Search(listclass &L, T Data);
private:
};
```

```
class  Flight
{
public:
  int   NoOfTrav;
  char  Departure[MaxName];
  char  Arrival[MaxName];
  char  From[MaxName];
  char  To[MaxName];
  int   Price;
  listclass<Traveler>   l;
  queueclass<Traveler>  q;
  int   NoOfl;
  int   NoOfq;
private:
};
```

Project NO 2 Memory Management

```
class  job{
public:
  int arrivetime,burst,cpuleft,waiting;
};
```

```
class  node{
public:
            job J; node *Next;
};
typedef  node *Ptr;
```

```
class queueclass{
public:
            Ptr  head,tail;
            int maxentry;
      enqueue(queueclass & q, job J);
      serve(queueclass & q,job & J);
}queue;
```

Project NO 3 Time Sharing

```
class  process
{
public:
  int id,start,finish,currburst,nofbursts,sumbursts;
  int bursts[maxburst];
private:
};
```

```
class  proc
{
public:
            process data;
    proc *nextproc;
private:
};
```

```
typedef  proc *procpointer;
```

```
class queueclass
{
public:
            procpointer  head,tail;
              queueclass();
       push(queueclass & q, process data);
       pop(queueclass & q,process & data);
       popshortest(queueclass & q,process & data);
private:
}queue;
```

Project NO 4 OO Bag

```
template <class AnyType>
class BAG {
public:
   #define Defaultcapacity 10;
   typedef AnyType *DArray;
   DArray data;
   int capacity;
   int used;
BAG() {
  capacity=Defaultcapacity;
  data= new AnyType [10];
  used=-1;
}
void resize(BAG &BAG1,int Newcapacity);
virtual void insert(BAG &BAG1,AnyType Item);
virtual void remove(BAG &BAG1,AnyType Item);
int occurrences(BAG BAG1,AnyType Item);
char in_BAG(BAG BAG1,AnyType Item);
friend BAG  operator +(BAG BAG1,BAG BAG2);
friend BAG operator +=(BAG BAG1,BAG BAG2);
ostream &operator<<(BAG BAG1);
istream &operator>>(BAG &BAG1);
void InsertAll(BAG BAG1,BAG &BAG2);
};
```

```
template <class AnyType>
class Stack:public BAG<AnyType> {
public:
Stack() {
  capacity=Defaultcapacity;
  data= new AnyType [10];
  used=-1;
}
friend Stack operator+(Stack BAG1,Stack BAG2);
friend Stack operator+=(Stack BAG1,Stack BAG2);
void remove(Stack &BAG1) {if(BAG1.used>-1) BAG1.used--;}
void insert(Stack &BAG1,AnyType Item) {
 int i;
 if (BAG1.capacity==BAG1.used+1) resize(BAG1,BAG1.capacity+1);
 BAG1.used++;for(i=used;i>0;i--)BAG1.data[i]=BAG1.data[i-1];
 BAG1.data[0]=Item;
}
};
```

```
template <class AnyType>
class Queue:public BAG<AnyType>  {
public:
Queue() {
  capacity=Defaultcapacity;
  data= new AnyType [10];
  used=-1;
}
void remove(Queue &BAG1)
{int i; if(BAG1.used>-1)
for(i=0;i<BAG1.used;i++) {BAG1.data[i]=BAG1.data[i+1];} BAG1.used--;}
friend Queue operator+(Queue BAG1,Queue BAG2);
friend Queue operator+=(Queue BAG1,Queue BAG2);
};
```

Project NO 5 OS Simulation

```
class  OSjob
{
public:
  int id,arrivetime,cpuleft;
private:
};
```

```
class  Job
{
public:
    OSjob data;
    Job *nextproc;
private:
};
```

```
typedef  Job *procpointer;
```

```
class queueclass
```

```
{
public:
            procpointer  head,tail;
            int maxentry;
              queueclass();
      enqueue(queueclass & q, OSjob data);
      serve(queueclass & q,OSjob & data);
      serveolder(queueclass & q,OSjob & data);
private:
}queue;
```

Project NO 6 SQL Parser

```
class symbol{
            char Variable[10];
            int  vType;
};
```

```
class symbol SymbolTable[30];
```

```
 class schema
  { int ColumnNumber;
            char  Table_Name[10];
    char Column_Name[10];
    int Data_Type;
    int Data_Length;
    int Member_of_PK;
  } ;
```

Project NO 7 Postfix

```
class stackclass
{
public:
   int   top;
   DT Expression[MaxStackSize];
   InitStack(stackclass & s);
   DestroyStack(stackclass & s);
   bool StackIsEmpty(stackclass s);
   bool StackIsFull(stackclass s);
   Push(stackclass & s, DT Value);
   Pop(stackclass & s,DT & Value);
private:
};
```

Project NO 8 Functional Dependencies

```
typedef char Atr[26];
```

```
class  PNode
{
public:
  Atr    LSide;
  Atr    RSide;
  PNode  *Next;
private:
};
```

```
class Closure
{
public:
                PNode  *Head;
                PNode  *Tail;
       PNode  *Current;
                Closure();
                ~Closure();
                bool Empty();
                bool Last();
                void GoFirst();
       void GoNext();
                bool Insert(Atr,Atr);
                void Remove();
       bool Found(Atr,Atr);
       bool Reflexivity();
       bool Augmentation();
       bool  Transitivity();
                void ShowFD();
                void DestroyFD();
private:
};
```

Project NO 9 Multi Linked List

```
template <class TypeA, class TypeB>
class MultiLL
{
public:
        PNode<TypeA, TypeB> *headA,*currentA;
    PNode<TypeA, TypeB> *headB,*currentB;
        MultiLL();
        ~MultiLL();
        bool empty();
        bool lastA();
        bool lastB();
        void findfirstA();
    void findfirstB();
    void findnextA();
        void findnextB();
        TypeA retrieveA();
        TypeB retrieveB();
        void updateA(TypeA);
        void updateB(TypeB);
        void insert(TypeA, TypeB);
        void removeA();
    void removeB();
private:
};
```

```
template <class TypeA, class TypeB>
class  PNode
{
public:
  TypeA   dataA;
  TypeB   dataB;
 PNode<TypeA, TypeB> *nextA, *nextB;
private:
};
```

Project NO 10 Pharmacy

```
class  Visit
{
public:
  int  times,tlength;
  char doctor[MaxName],medicine[MaxName],dosage[MaxName];
private:
};
```

```
class  patient
{
public:
  int id;
  char pname[MaxName];
private:
};
```

```
class  QueueNode
{
public:
  patient    data;
  QueueNode  *next;
private:
};
```

```
typedef  QueueNode *QueueNodePtr;
```

```
template <class T>
class stackclass
{
public:
              int   top;
              T Hestory[MaxVisits];
              InitStack(stackclass & s);
      DestroyStack(stackclass & s);
              bool StackIsEmpty(stackclass s);
      bool StackIsFull(stackclass s);
      Push(stackclass & s,  T prescription);
      Pop(stackclass & s,T & prescription);
private:
};
```

```
class  PatientRec
{
public:
  patient    data;
  int        status;
  stackclass<Visit> Stack;
private:
};
```

```
class  ListNode
{
public:
  PatientRec  data;
  ListNode  *next;
private:
};
```

```
template <class  T>
class queueclass
{
public:
                QueueNodePtr  front,rear;
                InitQueue(queueclass & q);
        DestroyQueue(queueclass & q);
                bool QueueIsEmpty(queueclass q);
        bool QueueIsFull(queueclass q);
                bool Search(queueclass q,int ID);
        EnQueue(queueclass & q,  T prescription);
        DeQueue(queueclass & q,  T & prescription);
private:
};
```

```
typedef  ListNode *ListNodePtr;
```

```
template <class  T>
class listclass
{
public:
                ListNodePtr  head,current;
                InitList(listclass & L);
        DestroyList(listclass & L);
                bool ListIsEmpty(listclass L);
        bool ListIsFull(listclass L);
        bool CurIsEmpty(listclass L);
                 ToFirst(listclass & L);
                bool AtFirst(listclass L);
                bool AtEnd(listclass L);
                Advance(listclass & L);
        Insert(listclass & L,  T prescription);
                InsertAfter(listclass & L,  T prescription);
        Delete(listclass & L,  T & prescription);
        StoreInfo(listclass & L,  T prescription);
                RetrieveInfo(listclass L,T & prescription);
                RetrieveNextInfo(listclass L,  T &prescription );
                bool Search(listclass &L,int ID,  T &PR);
private:
};
```