Software Cost Estimation
Metrics Manual for
Defense Systems
Acknowledgements

Many people and institutions supported this manual and made it possible. We thank the numerous reviewers for valuable feedback and suggestions on earlier wiki versions. Reviews were provided both online formally and during workshops. Forty-eight workshop participants helped guide our research and improve this material hosted by the University of Southern California Center for Systems and Software Engineering (USC CSSE) for several years, the Massachusetts Institute of Technology Engineering Systems Division, and the Carnegie Mellon University Software Engineering Institute. The DoD-sponsored Cyber Security and Information Systems Information Analysis Center and Quanterion Solutions also helped support this work and hosted web seminars.

The external reviewers for this edition were:

- Jo Ann Lane, University of Southern California
- Daniel Ligett, Softstar Systems
- Daniel Nussbaum, Naval Postgraduate School
- David Seaver, National Security Agency
- Richard Selby, Northrop Grumman
- Daniel Strickland, Missile Defense Agency
- David Zubrow, Software Engineering Institute

This empirical research depended on many data submitters for their project data and time, but they must remain anonymous. Tool vendors provided us with people, tools, and manuals. The cost estimation tool companies supporting this work were:

- Galorath Systems
- PRICE Systems
- Quantitative Software Management
- Softstar Systems

From the estimation tool companies we were assisted by Dan Galorath, Tim Hohmann, Bob Hunt, and Karen McRitchie at Galorath Systems; Arlene Minkiewicz, James Otte and David Seaver from PRICE Systems; Larry Putnam from Quantitative Software Management; and Dan Ligett from Softstar Systems.

We also thank Jairus Hihn and Sherry Stukes from NASA Jet Propulsion Laboratory, Tom McGibbon from Quanterion Solutions, and Don Reifer at
Reifer Consultants for their support and collaboration effort. Dave Olwell from Naval Postgraduate School and Lori Vaughn from Northrop Grumman also helped us with the manual.

Wilson Rosa initiated the research for this manual while at the Air Force Cost Analysis Agency (AFCAA) to help improve software cost estimation practices for analysts and decision makers across the DoD community collaboratively. He worked tirelessly bridging disparate stakeholders. The research was also supported by the Systems Engineering Research Center (SERC) under Contract H98230-08-D-0171, and the US Army Contracting Command, Joint Munitions & Lethality Center, Joint Armaments Center, Picatinny Arsenal, NJ, under RFQ 663074. These research results have been approved for public distribution.
Contributors

Barry Boehm  
University of Southern California  
Los Angeles, California

Bradford Clark  
Software Metrics Inc.  
Haymarket, Virginia

Joseph Dean  
Air Force Cost Analysis Agency  
Washington, DC

Cheryl Jones  
US Army Armament Research Development and Engineering Center  
Picatinny Arsenal, New Jersey

Raymond Madachy  
Naval Postgraduate School  
Monterey, California

John McGarry  
US Army Armament Research Development and Engineering Center  
Picatinny Arsenal, New Jersey

Wilson Rosa  
Naval Center for Cost Analysis  
Washington, DC
Preface

This manual transitions our research analyzing and improving software development cost metrics reported with the United States Department of Defense (DoD) Software Resource Data Report (SRDR). The goal is to create cost estimating relationship (CER) models and productivity benchmarks based on consistent metrics definitions, and provide guidance in their usage. It is primarily intended for DoD software cost analysts at different levels from entry level to seasoned experts. Cost analysts and program management in related and similar application domains will also find it useful.

It includes a description of the metrics data that the SRDR instruments; how to inspect and normalize it for consistency; and characterizes the CERs and productivity benchmarks derived from the current data. Readers can gain useful lessons learned from analyzing the metrics, some of which were used to improve the data reporting requirements.

The manual covers in detail the core metrics definitions relevant to reporting and analyzing DoD cost data. These standardized software metrics enable a consistent description of the data used in creating CERs. It illustrates the steps involved in normalizing SRDR data or software metrics in similar domains. There is a comprehensive discussion on modern estimating challenges that provides guidance for applying CERS on actual and upcoming DoD programs. The full thread of the program software cost estimation process is overviewed including data reporting and CER usage.

This evolving research will continue with additional program data continuously collected in the Defense Automated Cost Information Management System (DACIMS) that we analyze from, and there will be further improvements in the reporting. The SRDR data requirements are periodically revisited in the DoD and updated to collect valuable and higher quality data. The intent is to keep this manual relevant with future editions incorporating new results.

This book has a companion web site at http://softwarecost.org for supplemental resources, content updates and errata. Readers can also submit comments and improvement suggestions.
List of Figures

1.1 Example Defense Operating Environments .......................... 2
1.2 Estimation and Metrics Processes .............................. 3
1.3 Normalized Effort vs. Equivalent Size in SRDRs ............... 4

5.1 Building CERs and Benchmarks ................................. 45
5.2 No Relationship .................................................. 57
5.3 Strong Positive Relationship .................................... 58
5.4 Strong Negative Relationship .................................... 58
5.5 Homoscedastic Relationship ..................................... 59
5.6 Heteroscedastic Relationship .................................... 60
5.7 RTE Size Data Distributions ...................................... 61
5.8 Residuals Scatter Plot ............................................ 67
5.9 RTE CER Prediction Interval and Error Distribution ........... 68
5.10 AV Dataset and CER Summary .................................... 71
5.11 GS Dataset and CER Summary .................................... 72
5.12 GV Dataset and CER Summary .................................... 73
5.13 MV Dataset and CER Summary .................................... 74
5.14 OV Dataset and CER Summary .................................... 75
5.15 CC Dataset and CER Summary .................................... 77
5.16 CAS Dataset and CER Summary .................................... 78
5.17 COM Dataset and CER Summary .................................... 79
5.18 MP Dataset and CER Summary ..................................... 80
5.19 RTE Dataset and CER Summary .................................... 81
5.20 S&S Dataset and CER Summary .................................... 82
5.21 SCP Dataset and CER Summary .................................... 83
5.22 SYS Dataset and CER Summary .................................... 84
5.23 TMDE Dataset and CER Summary .................................. 85
5.24 VC Dataset and CER Summary ..................................... 86
5.25 VP Dataset and CER Summary ..................................... 87
5.26 CCGS Dataset and CER Summary ................................... 89
5.27 CASGS Dataset and CER Summary ............................... 90
5.28 COMGS Dataset and CER Summary ............................... 91
5.29 COMMV Dataset and CER Summary ............................... 92
5.30 MPG Dataset and CER Summary ................................... 93
5.31 RTEAV Dataset and CER Summary ............................... 94
List of Tables

2.1 SRDR Reporting Events .................................... 10
3.1 Software Size Types ........................................ 20
3.2 SLOC Count Definitions .................................... 22
3.3 ESLOC Summary ........................................... 25
3.4 User Function Types ....................................... 27
3.5 Function Point Counting Weights for Internal Logical Files and
    External Interface Files ................................. 28
3.6 Function Point Counting Weights for External Outputs and
    External Inquiries ...................................... 28
3.7 Function Point Counting Weights for External Inputs ........ 28
3.8 Unadjusted Function Point Complexity Weights ............. 29
3.9 Effort Activities .......................................... 29
3.10 Effort Phases ............................................. 30
4.1 Data Quality Rating Scale ................................ 37
4.2 SRDR Activities ........................................... 41
4.3 Proxy DM, CM, and IM Values ............................... 42
5.1 Operating Environments Definition ........................ 47
5.2 Application Types Definitions ............................... 48
5.3 CER Evaluation Statistics ................................ 64
5.4 RTE CER Regression Statistics ............................. 67
5.5 RTE CER Coefficient Statistics ............................ 67
5.6 Model Coverage ............................................ 69
5.7 CERs by Operating Environment ............................ 70
5.8 CERs by AppType Across All Environments ............... 76
5.9 CERs by AppType and OpEnv ............................... 88
5.10 Effort Distributions ....................................... 102
5.11 Productivity Statistic Definitions ......................... 104
5.12 Productivity Benchmarks by Operating Environment .... 105
5.13 Productivity Benchmarks by Application Type ........... 106
5.14 Productivity Benchmarks by Operating Environment and
    Application Type ....................................... 108
6.1 Recommended RVOL Rating Levels ......................... 112

xiii
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.2</td>
<td>IDPD Effort Drivers</td>
<td>113</td>
</tr>
<tr>
<td>6.3</td>
<td>Situation-Dependent Processes and Estimation Approaches</td>
<td>121</td>
</tr>
<tr>
<td>6.4</td>
<td>Process Selection Criteria</td>
<td>122</td>
</tr>
<tr>
<td>7.1</td>
<td>Radar Software WBS Decomposition</td>
<td>133</td>
</tr>
<tr>
<td>A.1</td>
<td>Cost Model Size Inputs</td>
<td>151</td>
</tr>
<tr>
<td>A.2</td>
<td>Cost Model Factors</td>
<td>159</td>
</tr>
<tr>
<td>A.3</td>
<td>Model Lifecycle Phases</td>
<td>162</td>
</tr>
<tr>
<td>A.4</td>
<td>Model Cost Activities</td>
<td>163</td>
</tr>
<tr>
<td>A.5</td>
<td>Model Cost Categories</td>
<td>164</td>
</tr>
</tbody>
</table>
# Contents

1 **Introduction** 1
   1.1 Purpose .................................................. 1
   1.2 Estimation and Metrics Processes ........................ 2
      1.2.1 Dataset ............................................. 4
      1.2.2 Usage Scenarios ...................................... 5
   1.3 Cost Estimating Relationship Approach .................... 5
   1.4 Manual Contents .......................................... 6

2 **Software Resources Data Report** 9
   2.1 Overview .................................................. 9
   2.2 Collecting Organization ................................. 9
   2.3 Repository ............................................... 10
   2.4 Reporting Frequency ..................................... 11
      2.4.1 Ramifications ....................................... 11
   2.5 SRDR Content ............................................. 11
      2.5.1 SRDR Data Elements ................................. 12
         2.5.1.1 Administrative Information .................. 12
         2.5.1.2 Product and Development Description ....... 13
         2.5.1.3 Product Size Reporting ........................ 14
         2.5.1.4 Resource and Schedule Reporting ............ 15
         2.5.1.5 Product Quality Reporting .................... 16
         2.5.1.6 Data Dictionary ................................. 16
   2.6 Further SRDR Resources ................................... 17

3 **Metrics Definitions** 19
   3.1 Overview .................................................. 19
   3.2 Product Size Measures .................................... 19
      3.2.1 Source Lines of Code (SLOC) ....................... 19
         3.2.1.1 SLOC Type Definitions ........................ 19
         3.2.1.2 SLOC Counting Rules ......................... 21
      3.2.2 Equivalent Size ..................................... 23
         3.2.2.1 Purpose in Estimating ......................... 24
         3.2.2.2 Adapted SLOC Adjustment Factors ............ 25
         3.2.2.3 Total Equivalent Size ......................... 26
      3.2.4 Volatility ............................................ 26
   3.3 Functional Size Measures ................................. 26
3.3.1 Function Points ............................................ 26
3.4 Development Effort ........................................... 29
  3.4.1 Activities and Lifecycle Phases ....................... 29
  3.4.2 Labor Categories ....................................... 30
  3.4.3 Labor Hours ............................................ 31
3.5 Development Duration ...................................... 31

4 Data Assessment ............................................. 33
  4.1 Gather Collected Data .................................... 34
  4.2 Inspect each Data Record ................................ 34
  4.3 Determine Data Quality Levels ......................... 36
  4.4 Correct Missing or Questionable Data .................. 38
  4.5 Normalize Size and Effort Data ....................... 38
    4.5.1 Converting to Logical SLOC ......................... 38
    4.5.2 Standardizing Effort ................................ 39
  4.6 Convert Raw SLOC into Equivalent SLOC .............. 40
  4.7 Future Work and Limitations ........................... 42

5 Cost Estimating Relationships ............................. 45
  5.1 Overview .................................................. 45
  5.2 Data Segmentation ....................................... 46
    5.2.1 Operating Environments (OpEnv) .................... 46
    5.2.2 Application Types (AppType) ....................... 48
    5.2.3 MIL-STD-881C WBS Mapping to Application Types 54
  5.3 Estimating Relationships ................................ 56
    5.3.1 Scatter Plots ....................................... 57
    5.3.2 Data Transformation ................................ 60
    5.3.3 CER Models ......................................... 61
    5.3.4 CER Calibration ..................................... 63
    5.3.5 CER Evaluation ..................................... 63
    5.3.6 CER Example ....................................... 66
  5.4 SRDR-Derived CERs ....................................... 69
    5.4.1 Model Coverage ..................................... 69
    5.4.2 Operating Environment CERs ......................... 70
      5.4.2.1 Aerial Vehicle (AV) ............................ 71
      5.4.2.2 Ground Site (GS) ............................... 72
      5.4.2.3 Ground Vehicle (GV) ............................ 73
      5.4.2.4 Maritime Vessel (MV) ........................... 74
      5.4.2.5 Ordnance Vehicle (OV) ......................... 75
    5.4.3 Application Types CERs ............................ 76
      5.4.3.1 Command and Control (C&C) ..................... 77
      5.4.3.2 Custom AIS Software (CAS) ..................... 78
      5.4.3.3 Communication (COM) ............................ 79
      5.4.3.4 Mission Planning (MP) .......................... 80
      5.4.3.5 Real Time Embedded (RTE) ..................... 81
5.4.3.6 Scientific and Simulation (S&S) ................. 82
5.4.3.7 Sensor Control and Signal Processing (SCP) ....... 83
5.4.3.8 System Software(SYS) ............................ 84
5.4.3.9 Test, Measurement and Diagnostic Equipment (TMDE) .......... 85
5.4.3.10 Vehicle Control (VC) ............................ 86
5.4.3.11 Vehicle Payload (VP) ............................ 87
5.4.4 Application Type and Operating Environment CERs ..... 88
5.4.4.1 Command & Control - Ground Site ................. 89
5.4.4.2 Custom AIS Software - Ground Site ................ 90
5.4.4.3 Communications - Ground Site .................... 91
5.4.4.4 Communications - Maritime Vessel ................ 92
5.4.4.5 Mission Planning - Ground Site .................. 93
5.4.4.6 Real Time Embedded - Aerial Vehicle ............. 94
5.4.4.7 Real Time Embedded - Ground Site ................ 95
5.4.4.8 Sensor Control and Signal Processing - Ground Vehicle .... 96
5.4.4.9 Sensor Control and Signal Processing - Ground Site .... 97
5.4.4.10 System Software - Ground Site .................. 98
5.4.4.11 Vehicle Control - Aerial Vehicle ................ 99
5.4.4.12 Vehicle Control - Ground Vehicle ................. 100
5.4.4.13 Vehicle Payload - Aerial Vehicle ................ 101
5.4.5 CER Effort Distribution ............................. 102
5.5 SRDR-Derived Benchmarks ............................. 102
5.5.1 Benchmark Model and Evaluation ........................ 102
5.5.2 Operating Environment Benchmarks ........................ 103
5.5.3 Application Type Benchmarks ............................ 105
5.5.4 AppType & OpEnv Benchmarks ............................ 107
5.6 Limitations and Future Work ............................ 108

6 Modern Estimating Challenges .................................. 111
6.1 Changing Objectives, Constraints and Priorities .......... 111
6.1.1 Rapid Change, Emergent Requirements, and Evolutionary Development ................................ 111
6.1.2 Net-centric Systems of Systems (NCSoS) ............ 114
6.1.3 Model-Driven and Non-Developmental Item (NDI)-Intensive Development ................................ 115
6.1.4 Ultrahigh Software Systems Assurance ................ 116
6.1.5 Legacy Maintenance and Brownfield Development ...... 116
6.1.6 Agile and Kanban Development ........................ 117
6.1.7 Putting It All Together at the Large-Project or Enterprise-Level .................................. 118
6.2 Estimation Approaches for Different Processes .......... 119
7 Estimation Process

7.1 Overview ........................................................................ 125
  7.1.1 Four Estimation Stages ........................................... 126
    7.1.1.1 Initiation and Research ................................. 128
    7.1.1.2 Assessment .................................................. 128
    7.1.1.3 Analysis ..................................................... 129
    7.1.1.4 Documentation and Presentation .................. 129
  7.2 Estimation Purpose ..................................................... 130
  7.3 Program Definition ..................................................... 130
  7.4 Estimation Scope ....................................................... 131
    7.4.1 Ground Rules and Assumptions .......................... 132
    7.4.2 Estimation Structure ....................................... 132
  7.5 Data Collection and Normalization ............................. 134
  7.6 Estimate Creation ..................................................... 135
    7.6.1 Historical Reconstruction .................................. 135
    7.6.2 Estimate Construction ..................................... 135
    7.6.3 Estimate Cost Analysis ................................... 136
    7.6.4 Alternate Estimate .......................................... 136
  7.7 Sensitivity Analysis ................................................... 137
  7.8 Risk and Uncertainty Analysis .................................... 137
    7.8.1 Common Probability Distributions ...................... 138
    7.8.2 Monte Carlo Analysis with a CER ...................... 141
  7.9 Estimate Documentation and Packaging ....................... 143

Appendix A - Cost Model Descriptions .................................... 145
  A.1 Introduction ........................................................ 145
  A.2 Cost Models ........................................................ 146
    A.2.1 COCOMO II .................................................. 146
    A.2.2 True Planning .............................................. 147
    A.2.3 SEER-SEM .................................................. 147
    A.2.4 SLIM .......................................................... 148
  A.3 Cost Model Input Factors .......................................... 149
    A.3.1 Software Size ............................................... 149
      A.3.1.1 Overview and Sizing Units ......................... 149
      A.3.1.2 New Software ....................................... 152
      A.3.1.3 Adapted Software (Modified and Reused) ....... 152
      A.3.1.4 Generated Software ................................ 156
      A.3.1.5 Automatically Translated Software ............. 156
      A.3.1.6 Commercial Off-The-Shelf Software (COTS) .... 157
    A.3.2 Software Cost Drivers ...................................... 158
  A.4 Cost Model Lifecycles and Work Breakdown Structures ....... 161
Chapter 1

Introduction

There is no good way to perform a software cost-benefit analysis, breakeven analysis, or make-or-buy analysis without some reasonably accurate method of estimating software costs and their sensitivity to various product, process, personnel, project, and environmental factors.
-Barry Boehm

1.1 Purpose

This metrics manual helps analysts and decision makers develop accurate, easy and quick early software cost estimates for different types of systems and operating environments oriented for the United States Department of Defense (DoD) and government agencies in related domains. The intent is to improve quality and consistency of early software estimating methods across government cost agencies and program offices through guidance, standardization, and knowledge sharing.

These defense environments may be sea, air, space, or ground-based like the typical examples in Figure 1.1. We have analyzed empirical software cost data for these types of programs and are transitioning the results back in this open access manual. Furthermore we describe our processes for data normalization, analysis, derivation of Cost Estimating Relationships (CERs) and productivity benchmarks.

These processes depend on data submittals from executing programs that adhere to standardized metrics definitions. This submission is the Software Resources Data Report (SRDR). The responsibility to submit consistent and complete SRDRs is incumbent on programs for the continuous metrics feedback loop to improve software cost estimation in the DoD. This manual illustrates how standardized metrics reporting leads to better predictive cost models.

These reported metrics are used to derive costs for future programs. This manual follows the SRDR interpretation of cost as the amount of effort in Person-Months required to develop the software. This can be used to generate cost estimates based on time-varying labor rates in cost per person-month.
1.2 Estimation and Metrics Processes

The context of this manual within overall related estimation and metrics processes in the DoD is shown in Figure 1.2. The Defense Acquisition Management System (DAMS) at the top consists of five phases and major milestones A, B and C that DoD programs transition through [38]. Programs must provide ongoing cost estimates as they proceed through the phases. SRDR submittals are also required throughout as major software builds are completed [44]. These metrics data reports are shown as outputs from different phases. The submitted SRDRs from contractors are stored in the DoD Defense Automated Cost Information Management System (DACIMS).

The SRDRs mostly originate in the Engineering and Manufacturing Development phase where the majority of software development takes place. Large prototyping or early development contracts exceeding $20M in Technology Development also contribute SRDRs. A smaller number originate in Production and Deployment for large system upgrades or Pre-Planned Product improvements more than $20M. This may also rarely happen in the Operations and Support phase, and SRDRs are submitted accordingly as the upgrades become programs themselves. Any software development effort exceeding the $20M threshold must report SRDRs.
Introduction

FIGURE 1.2: Estimation and Metrics Processes

The Program Estimation Process in Figure 1.2 supports the ongoing program estimates depicted by vertical arrows up to the DAMS. These estimates become increasingly detailed with better knowledge over time. Relatively simple CERs are used in early phases because little is known about the future design and implementation of the software. As a program proceeds through development, production and deployment, more information is available so multi-parameter CERs (e.g. detailed cost models) are used to provide estimates-to-complete and track progress. These CERs incorporating additional factors correspond to increased knowledge of the program.

The Program Estimation Process has an Assessment activity which includes deciding the best CER to apply. Available to be used are the simple CERs derived from SRDRs in this manual, or multi-parameter CERs corresponding to the detailed cost models also described herein.

The Software Cost Estimation Metrics Process described in this manual operates with both other major processes. It uses the empirical SRDR data collected from programs transitioning through the DAMS as input data. The collected SRDRs which are stored in the DACIMS are made available to government cost analysts. This data is then prepared by normalizing it to stan-
standardized metric definitions. Simple CERs are created from the data and used in the Program Estimation Process. The same metrics definitions also underpin the multi-parameter CERs which support the Program Estimation Process.

Important metrics feedback loops are also illustrated in Figure 1.2. An overall feedback loop includes the submission of SRDRS to the DACIMS repository being analyzed for future program estimates. Those programs eventually execute and provide ongoing data for continuous CER calibration. Within the software cost estimation metrics process there is a continuous feedback loop for evolving CERs. This happens due to ongoing analysis of current data combined with new data submittals. The program estimation process is inherently iterative and the results are fed back to earlier phases in the estimation process. The iterations may repeat for successive estimates throughout a project’s lifecycle.

1.2.1 Dataset

The analyses in this manual are based on 317 SRDRs reported from recent DoD projects during 2004-2013. That raw data was normalized as described later in this manual. A top-level view of the normalized dataset across the services is shown in Fig. 1.3. Cost is expressed as Person-Months plotted against software size in Thousand Equivalent Source Lines of Code (KESLOC) presented in logarithmic scale. This normalized data was then segmented by application type (described in Chapter 5) to develop the CERs, whereby cost is predicted as a function of size.

![Normalized Effort vs. Equivalent Size in SRDRs](image-url)
1.2.2 Usage Scenarios

The primary usage of this manual is for government analysts estimating software development costs or assessing costs in complex systems. They may be creating an independent cost estimate, or validating and cross-checking software cost estimates provided by contractors. Existing CERs can provide sanity checks without full detailed information on the future programs. In some cases, new CERs need to be developed when existing CERs are inadequate. Productivity benchmarks are also useful for comparison purposes. This manual provides expected software productivities across environments and applications.

CERs and productivity benchmarks are also important to contractors who provide SRDR data. In most cases, the people responsible for reporting the SRDR are the same preparing the cost proposals for source selection. Publicized CERs and benchmarks will help them crosscheck their cost proposal estimates against government benchmarks.

Defensible estimates are mostly needed at the early conceptual phase of a software-intensive system’s definition and development. A cost estimation model with 20-30 input parameters is not very helpful without a defensible approach for specifying the input values for key parameters as the software’s complexity, database size, platform volatility, required schedule compression, tool coverage, or proposed project personnel experience.

This manual provides simple software effort models for early estimates for these reasons. It examines the direct effect of product size and application type on cost. The analysis framework and variables used in this study builds on causal relationships established in past studies.

1.3 Cost Estimating Relationship Approach

This manual describes a bootstrap approach to creating software cost estimation relationships based on continuous data analysis. The method follows these basic steps:

1. Collect software cost metrics data
   - The data used in this manual is collected from the DoD’s DACIMS repository of submitted SRDRs.

2. Prepare the data for analysis (e.g., normalizing different measures of the same metric)
   - This step heavily relies on a set of standard metric definitions
   - Data is then normalized to the standard set of metric definitions

3. Create CER models and benchmarks from the data

- Segmenting the data into logical groups
- Derive estimation formulae with parameters relating size and effort.
- Derive productivity benchmarks or ratios

4. Address challenges in using CERs with modern software development practices.

5. Evolve the CERs based on comparing estimates with actual costs on future completed systems.

1.4 Manual Contents

The manual generally follows the five steps above as reflected in subsequent chapters:

SRDR

The SRDR is discussed first since the collected data on these reports underpins the analysis in this manual. It describes what the data consists of and when it is reported. It is important to understand the SRDR because this defines what the CERs cover. This chapter covers:

- Programs that report SRDRs
- Frequency of data collection
- Scope of data collection
- Data elements collected

Metrics Definitions

The next step is preparing the data for analysis and ensuring consistency. The metrics must be defined in a standard way before doing anything with the data. This chapter covers baseline definitions used to assess the homogeneity of the data collected. When the analysis step begins, the data needs to be as comparable as possible for apples to apples comparison.

Metric definitions also define the inputs and outputs of the CERs. An awareness of the metric definitions is very important to prevent misuse of the CERs, and it also supports data collection activities.

Data Assessment

With an understanding of the data sources and metric definitions, this chapter covers the assessment of the data for use in analysis. Missing data is removed. The completeness of the data is assessed. Each data record is compared to the metric definitions and, where necessary, data is transformed to
be compliant with the metric definition.

**CERs and Benchmarks**

The cost estimating relationships are derived from the data collected and normalized. This chapter shows the use of relatively simple statistical methods to create CERs from the data:

- Data is segmented into two categories: *Operating Environment* and *Application Type*
- Effort and size data are analyzed for CERs within each group
- Data is analyzed for productivity ratios within the groups

**Modern Estimating Challenges**

This chapter presents an overview of the challenges in using the CERs in modern development practices and guidance for dealing with them. These challenges include:

- Rapid Change, Emergent Requirements, and Evolutionary Development
- Net-Centric System of Systems
- Model-Driven and Non-Development Item-Intensive Development
- Ultrahigh Software Systems Assurance
- Agile and Kanban Development

**Estimation Process**

This chapter discusses a software cost estimation process following Government Accountability Office (GAO) guidelines. The guidelines are discussed as they apply to software development cost estimates. Previous chapters in this manual support different steps in the estimation process, which are referenced where applicable.

**Appendices**

A detailed appendix is included on additional parametric cost model descriptions. These cost models are applicable for detailed estimates when more is known later in the lifecycle. Also included are appendices on MIL-STD-881C Work Breakdown Structure (WBS) mapping, code counting, nomograms for each CER, and acronyms found in this manual.
Chapter 2

Software Resources Data Report

2.1 Overview

The Software Resources Data Report (SRDR) is used to obtain both the estimated and actual characteristics of new software developments or upgrades [44]. Both the Government program office and, after contract award, the software contractor submit this report. For contractors, this report constitutes a contract data deliverable that formalizes the reporting of software metric and resource data.

SRDR submittals are required for all contractors developing or producing any software development element with a projected software effort greater than $20M (then year dollars) on major contracts and subcontracts within ACAT I and ACAT IA programs, regardless of contract type. The data collection and reporting applies to developments and upgrades whether performed under a commercial contract or internally by a government Central Design Activity (CDA) under the terms of a Memorandum of Understanding (MOU). Table 2.1 summarizes the reporting events.

2.2 Collecting Organization

The Defense Cost and Resource Center (DCARC), which is part of OSD Cost Assessment and Program Evaluation (CAPE), exists to collect Major Defense Acquisition Program (MDAP) cost and software resource data and make those data available to authorized Government analysts. Their website is the authoritative source of information associated with the Cost and Software Data Reporting (CSDR) system, including but not limited to: policy and guidance, training materials, and data.


## TABLE 2.1: SRDR Reporting Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Report Due</th>
<th>Who Provides</th>
<th>Scope of Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Contract (180 days prior to award)</td>
<td>Initial</td>
<td>Government Program Office</td>
<td>Estimates of the entire completed project. Measures should reflect cumulative grand totals.</td>
</tr>
<tr>
<td>Contract award</td>
<td>Initial</td>
<td>Contractor</td>
<td>Estimates of the entire project at the level of detail agreed upon. Measures should reflect cumulative grand totals.</td>
</tr>
<tr>
<td>Start of each build</td>
<td>Initial</td>
<td>Contractor</td>
<td>Estimates for completion for the build only.</td>
</tr>
<tr>
<td>Estimate corrections</td>
<td>Initial</td>
<td>Contractor</td>
<td>Corrections to the submitted estimates.</td>
</tr>
<tr>
<td>End of each build</td>
<td>Final</td>
<td>Contractor</td>
<td>Actuals for the build only.</td>
</tr>
<tr>
<td>Contract completion</td>
<td>Final</td>
<td>Contractor</td>
<td>Actuals for the entire project. Measures should reflect cumulative grand totals.</td>
</tr>
<tr>
<td>Actuals corrections</td>
<td>Final</td>
<td>Contractor</td>
<td>Corrections to the submitted actuals.</td>
</tr>
</tbody>
</table>

### 2.3 Repository

The DCARC’s Defense Automated Cost Information Management System (DACIMS) is the database for access to current and historical cost and software resource data needed to develop independent, substantiated estimates. DACIMS is a secure website that allows DoD government cost estimators and analysts to browse through almost 30,000 CCDRs, SRDR and associated doc-
2.4 Reporting Frequency

The SRDR Final Developer Report contains measurement data as described in the contractor’s SRDR Data Dictionary. The data reflects the scope relevant to the reporting event. Both estimates (DD Form 2630-1,2) and actual results (DD Form 2630-3) of software (SW) development efforts are reported for new or upgrade projects. SRDR submissions for contract complete event shall reflect the entire software development project. When the development project is divided into multiple product builds, each representing production level software delivered to the government, the submission should reflect each product build. SRDR submissions for completion of a product build shall reflect size, effort, and schedule of that product build.

2.4.1 Ramifications

It is very important to understand the submission criteria because there are critical ramifications. SRDR records are a mixture of complete contracts and individual builds within a contract. These must be distinguished in the reporting. Furthermore, each has initial and final reports along with corrections. Mixing contract data and build data or mixing initial and final results or not using the latest corrected version will produce inconclusive and possibly incorrect results.

2.5 SRDR Content

The SRDR has five sections plus an accompanying Data Dictionary [44]. These sections are:

1. Administrative Information
2. Product and Development Description
3. Product Size Reporting
4. Resource and Schedule Reporting
5. Product Quality Reporting

The Data dictionary contains details on the software that do not match an SRDR data field, explanations for lack of data or data coverage, and definitions of the metrics reported.
2.5.1 SRDR Data Elements

Below is a list of data elements reported in the SRDR as of the 2011 specification. The highlighted data elements are considered important for the analysis conducted for this manual. These data elements are from an Excel spreadsheet implementation of the SRDR Final Report 2011 form [44].

2.5.1.1 Administrative Information (SRDR Section 3.1)

- Security Classification
- **Major Program**
  - Program Name
  - Phase / Milestone
  - Reporting Organization Type (Prime, Subcontractor, Government)
- Name / Address
  - Reporting Organization
  - Division
- Approved Plan Number
- Customer (Direct-Reporting Subcontractor Use Only)
- Contract Type
- **WBS Element Code**
- WBS Reporting Element
- Type Action
  - Contract No
  - Latest Modification
  - Solicitation No
  - Common Reference Name
  - Task Order / Delivery Order / Lot No
- **Period of Performance**
  - Start Date (YYYYMMDD)
  - End Date (YYYYMMDD)
- Appropriation (RDT&E, Procurement, O&M)
- Submission Number
- Resubmission Number
- **Report As Of (YYYYMMDD)**
- Date Prepared (YYYYMMDD)
- Point of Contact
  - Name (Last, First, Middle Initial)
  - Department
  - Telephone Number (include Area Code)
Software Resources Data Report

- Email
- Development Organization

- Software Process Maturity
  - Lead Evaluator
  - Certification Date
  - Evaluator Affiliation

- Precedents (List up to five similar systems by the same organization or team.)
- SRDR Data Dictionary Filename
- Comments (on Report Context and Development Organization)

2.5.1.2 Product and Development Description (SRDR Section 3.2)

- Functional Description. A brief description of its function.
- Software Development Characterization
- Application Type
  - Primary and Secondary Programming Language.
  - Percent of Overall Product Size. Approximate percentage (up to 100%) of the product size that is of this application type.
  - Actual Development Process. Enter the name of the development process followed for the development of the system.
  - Software Development Method(s). Identify the software development method or methods used to design and develop the software product.
  - Upgrade or New Development. Indicate whether the primary development was new software or an upgrade.
  - Software Reuse. Identify by name and briefly describe software products reused from prior development efforts (e.g. source code, software designs, requirements documentation, etc.).

- COTS / GOTS Applications Used
  - Name. List the names of the applications or products that constitute part of the final delivered product, whether they are COTS, GOTS, or open-source products.
  - Integration Effort (Optional). If requested by the CWIPT, the SRD report shall contain the actual effort required to integrate each COTS / GOTS application identified in Section 3.2.4.1.

- Staffing
  - Peak Staff. The actual peak team size, measured in full-time equivalent (FTE) staff.
  - Peak Staff Date. Enter the date when the actual peak staffing occurred.

- Hours per Staff-Month. Enter the number of direct labor hours per staff-month.

- **Personnel Experience in Domain.** Stratify the project staff domain experience by experience level and specify the percentage of project staff at each experience level identified. Sample Format 3 identifies five levels:
  - Very Highly Experienced (12 or more years)
  - Highly Experienced (6 to 12 years)
  - Nominally Experienced (3 to 6 years)
  - Low Experience (1 to 3 years)
  - Inexperienced / Entry Level (less than a year)

2.5.1.3 Product Size Reporting (SRDR Section 3.3)

- **Number of Software Requirements.** Provide the actual number of software requirements.
  - Total Requirements. Enter the actual number of total requirements satisfied by the developed software product at the completion of the increment or project.
  - New Requirements. Of the total actual number of requirements reported, identify how many are new requirements.

- **Number of External Interface Requirements.** Provide the number of external interface requirements, as specified below, not under project control that the developed system satisfies.
  - Total External Interface Requirements. Enter the actual number of total external interface requirements satisfied by the developed software product at the completion of the increment or project.
  - New External Interface Requirements. Of the total number of external interface requirements reported, identify how many are new external interface requirements.

- **Requirements Volatility.** Indicate the amount of requirements volatility encountered during development as a percentage of requirements that changed since the Software Requirements Review.

- **Software Size.**
  - Delivered Size. Capture the delivered size of the product developed, not including any code that was needed to assist development but was not delivered (such as temporary stubs, test scaffolding, or debug statements). Additionally, the code shall be partitioned (exhaustive with no overlaps) into appropriate development categories. A common set of software development categories is new, reused with modification, reused without modification, carry-over code, deleted code, and auto-generated code.
– **Reused Code With Modification.** When code is included that was reused with modification, provide an assessment of the amount of redesign, recode, and retest required to implement the modified or reused code.

– **Reuse Code Without Modification.** Code reused without modification is code that has no design or code modifications. However, there may be an amount of retest required. Percentage of retest should be reported with the retest factors described above.

– **Carryover Code.** Report shall distinguish between code developed in previous increments that is carried forward into the current increment and code added as part of the effort on the current increment.

– **Deleted Code.** Include the amount of delivered code that was created and subsequently deleted from the final delivered code.

– **Auto-generated Code.** If the developed software contains auto-generated source code, report an auto-generated code sizing partition as part of the set of development categories.

– **Subcontractor-Developed Code.**

– **Counting Convention.** Identify the counting convention used to count software size.

– **Size Reporting by Programming Language** (Optional).

– **Standardized Code Counting** (Optional). If requested, the contractor shall use a publicly available and documented code counting tool, such as the University of Southern California Code Count tool, to obtain a set of standardized code counts that reflect logical size. These results shall be used to report software sizing.

### 2.5.1.4 Resource and Schedule Reporting (SRDR Section 3.4)

The Final Developer Report shall contain actual schedules and actual total effort for each software development activity.

- **Effort.** The units of measure for software development effort shall be reported in staff-hours. Effort shall be partitioned into discrete software development activities.

- **WBS Mapping.**

- **Subcontractor Development Effort.** The effort data in the SRDR report shall be separated into a minimum of two discrete categories and reported separately: Prime Contractor Only and All Other Subcontractors.

- **Schedule.** For each software development activity reported, provide the actual start and end dates for that activity.
The activities referred to in the SRDR are based on the ISO/IEC-12207 standard [25]. The possible development activities reported include:

- Software Requirements Analysis
- Software Architecture and Detailed Design
- Software Coding and Unit Testing
- Software Integration
- Software Qualification Testing
- System/Software Qualification Testing
- Software Quality Assurance
- Software Configuration Management
- Software Program Management

The current SRDR data has the top five items in addition to an activity called Development Test and Evaluation. All of the other activities are lumped under a category called Other.

2.5.1.5 Product Quality Reporting (SRDR Section 3.5 - Optional)

Quality should be quantified operationally (through failure rate and defect discovery rate). However, other methods may be used if appropriately explained in the associated SRDR Data Dictionary.

- **Number of Defects Discovered.** Report an estimated number of defects discovered during integration and qualification testing. If available, list the expected defect discovery counts by priority, e.g. 1, 2, 3, 4, 5. Provide a description of the priority levels if used.
- **Number of Defects Removed.** Report an estimated number of defects removed during integration and qualification testing. If available, list the defect removal counts by priority

2.5.1.6 Data Dictionary

The SRDR Data Dictionary contains, at a minimum, the following information in addition to the specific requirements identified in SRDR Sections 3.1 through 3.5:

- **Experience Levels.** Provide the contractor’s specific definition (i.e., the number of years of experience) for personnel experience levels reported in the SRDR report.
- **Software Size Definitions.** Provide the contractor’s specific internal rules used to count software code size.
- **Software Size Categories.** For each software size category identified (i.e., New, Modified, Unmodified, etc.), provide the contractor’s specific rules and / or tools used for classifying code into each category.
- **Peak Staffing.** Provide a definition that describes what activities were included in peak staffing.
• **Requirements Count (Internal).** Provide the contractor’s specific rules and / or tools used to count requirements.

• **Requirements Count (External).** Provide the contractor’s specific rules and / or tools used to count external interface requirements.

• **Requirements Volatility.** Provide the contractor’s internal definitions used for classifying requirements volatility.

• **Software Development Activities.** Provide the contractor’s internal definitions of labor categories and activities included in the SRDR report’s software activity.

• **Product Quality Reporting.** Provide the contractor’s internal definitions for product quality metrics being reported and specific rules and / or tools used to count the metrics.

### 2.6 Further SRDR Resources

An Excel spreadsheet implementation of the SRDR Final Report (2011 form) is on this book’s web site. Supplemental analysis of SRDR data is available through the DoD sponsored Cyber Security and Information Systems Information Analysis Center (CSIAC) [50]. See the DCARC website [43] for the DoD’s CSDR overview and policy.
Chapter 3

Metrics Definitions

3.1 Overview

This chapter defines software product size measures to support data consistency and interpretation, used as input to cost estimation models and productivity analyses. These measures are used throughout the manual including for data normalization, to derive CERs, productivity benchmarks and interpreted for selected cost models.

3.2 Product Size Measures

An accurate size estimate is the most important input to parametric cost models. However, determining size can be challenging. Projects may be composed of new code, code adapted from other sources with or without modifications, automatically generated or translated code, commercial software, and other types.

3.2.1 Source Lines of Code (SLOC)

The common measure of software size used in this manual and the cost models is Source Lines of Code (SLOC). SLOC are logical source statements consisting of data declarations and executables.

3.2.1.1 SLOC Type Definitions

The core software size type definitions used throughout this manual are summarized in Table 3.1. These definitions apply to size estimation, data collection, and analysis. To prevent confusion in reporting measures of size and in storing results in databases, the type of SLOC count should always be recorded.

The size types are applied at the source code file level for the appropriate system-of-interest. If a component, or module, has just a few lines of code
### TABLE 3.1: Software Size Types

<table>
<thead>
<tr>
<th>Size Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>New software created for the first time.</td>
</tr>
<tr>
<td>Adapted</td>
<td>Pre-existing software that is used as-is (Reused) or changed (Modified).</td>
</tr>
</tbody>
</table>
| Reused          | Pre-existing software that is not changed with the adaptation parameter settings:  
|                 | • Design Modification % (DM) = 0%                                           |
|                 | • Code Modification % (CM) = 0%.                                             |
| Modified        | Pre-existing software that is modified for use by making design, code and / or test changes:  
|                 | • Code Modification % (CM) > 0%.                                            |
| Equivalent      | A relative measure of the work done to produce software compared to the code-counted size of the delivered software. It adjusts the size of adapted software relative to developing it all new. |
| Generated       | Software created with automated source code generators. The code to include for equivalent size consists of automated tool generated statements. |
| Converted       | Software that is converted between languages using automated translators.    |
| Commercial Off-The-Shelf (COTS) | Pre-built commercially available software components. The source code is not available to application developers. It is not included for equivalent size. |
|                 | Other unmodified software not included in equivalent size are Government Furnished Software (GFS), libraries, operating systems and utilities. |

changed then the entire component is classified as Modified even though most of the lines remain unchanged. The total product size for the component will include all lines.

COTS normally isn’t reported in the delivered size. However, COTS glue code is often necessary. It should be included and used for estimation.

Open source software is handled, as with other categories of software, depending on the context of its usage. If it is not touched at all by the development team it can be treated as a form of COTS or reused code. However, when open source is modified it must be quantified with the adaptation parameters for modified code and be added to the equivalent size. The costs of integrating open source with other software components should be added into overall project costs.
3.2.1.2 SLOC Counting Rules

Logical Lines

The common measure of software size used in this manual and the cost models is Source Lines of Code (SLOC). SLOC are logical source statements consisting of data declarations and executables. Table 3.2 shows the SLOC definition inclusion rules for what to count. Based on the Software Engineering Institute’s Software Size Measurement: A Framework for Counting Source Statements [47], each checkmark in the Includes column identifies a particular statement type or attribute included in the definition, and vice-versa for the Excludes. See the following details on the software size terms.

Non-Commented Source Statements (NCSS)

The Non-Commented Source Statement count type only counts lines containing a programming language source statement. No blank lines or comment-only lines are counted.

Physical Lines

The Physical SLOC count type is a count type where programming language terminators or delimiters are counted. This count type excludes blank lines in a source code file and includes everything else.

Total Lines

The Total SLOC count type includes a count of everything, including blank lines.

Executables

Executable statements cause runtime actions. They may be simple statements such as assignments, gotos, procedure calls, macro calls, returns, breaks, exits, ends, stops, continues, nulls, no-ops, or empty statements. Or they may be structured or compound statements, such as conditional statements, repetitive statements, and "with" statements. Languages like Ada, C, C++, and Pascal have block statements [begin ... end and { ... }] that are classified as executable when used where other executable statements would be permitted. C and C++ define expressions as executable statements when they terminate with a semicolon, and C++ has a declaration statement that is executable.

Declarations

Declarations are non-executable program elements that affect an assembler’s or compiler’s interpretation of other program elements. They are used to name, define, and initialize; to specify internal and external interfaces; to assign ranges for bounds checking; and to identify and bound modules and sections of code. Examples include declarations of names, numbers, constants, objects, types, subtypes, programs, subprograms, tasks, exceptions, packages, generics, macros, and deferred constants.
## TABLE 3.2: SLOC Count Definitions

<table>
<thead>
<tr>
<th>Statement Type</th>
<th>Logical</th>
<th>NCSS</th>
<th>Physical</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>In</td>
<td>Out</td>
<td>In</td>
<td>Out</td>
</tr>
<tr>
<td>Executable</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Nonexecutable</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Declarations</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Compiler directives</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Comments</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Blank lines</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flow Control</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Logical expressions</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>If - Then - Else</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Go-To, Break, Continue</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Exception Handling</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Flow Control Punctuation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Block Statements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subprogram Label &amp; Delimiters</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Procedure / Function</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Label &amp; Delimiters</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Begin-End Blocks or Punctuation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Block Punctuation</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>How Produced</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programmed New</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Reused</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Modified</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Generated</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generator</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generator statements</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>3GL generated statements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In Maintenance</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>In Development</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Converted</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

(continued)
Declarations also include renaming declarations, use clauses, and declarations that instantiate generics. Mandatory begin .. end and {...} symbols that delimit bodies of programs and subprograms are integral parts of program and subprogram declarations. Language superstructure elements that establish boundaries for different sections of source code are also declarations. Examples include terms such as PROCEDURE DIVISION, DATA DIVISION, DECLARATIVES, END DECLARATIVES, INTERFACE, IMPLEMENTATION, SYS-PROC, and SYS-DD. Declarations, in general, are never required by language specifications to initiate runtime actions, although some languages permit compilers to implement them that way.

Compiler Directives

Compiler directives instruct compilers, preprocessors, or translators (but not runtime systems) to perform special actions. Some, such as Adas PRAGMA and COBOLs COPY, REPLACE, and USE, are integral parts of the source language. In other languages like C and C++, special symbols like # are used along with standardized keywords to direct preprocessor or compiler actions. Still other languages rely on non-standardized methods supplied by compiler vendors. In these languages, directives are often designated by special symbols such as #, $, and {§}.

3.2.2 Equivalent Size

A key element in using software size for effort estimation is the concept of equivalent size. Equivalent size is a quantification of the effort required to use previously existing code along with new code. For cost estimating relationships, the size of previously existing code does not require the same effort as the effort to develop new code of the same size. Equivalent size normalizes the effort relative to new software.

The guidelines in this section will help the estimator in determining the total equivalent size. All of the models described in Appendix A have tools for
doing this. However, for non-traditional size categories (e.g., a model may not provide inputs for auto-generated code), this manual will help the estimator calculate equivalent size outside of the tool and incorporate the size as part of the total equivalent size.

3.2.2.1 Purpose in Estimating

In addition to newly developed software, adapted software that is modified and reused from another source and used in the product under development also contributes to the product’s equivalent size. A method is used to make new and adapted code equivalent so they can be rolled up into an aggregate size estimate.

The size of reused and modified code is adjusted to be its equivalent in new code for use in estimation models. The adjusted code size is called Equivalent Source Lines of Code (ESLOC). The adjustment is based on the additional effort it takes to modify the code for inclusion in the product taking into account the amount of design, code and testing that was changed and as described next.

There are also different ways to produce software that complicate deriving ESLOC including generated and converted software. All of the categories are aggregated for equivalent size. A primary source for the equivalent sizing principles in this section is Chapter 9 of [53].

For traditional Third Generation Language (3GL) software such as C or Java, count the logical 3GL statements. For Model-Driven Development (MDD), Very High Level Languages (VHLL), or macro-based development, count the generated statements. A summary of what to include or exclude in ESLOC for estimation purposes is in Table 3.3.
3.2.2.2 Adapted SLOC Adjustment Factors

The AAF factor is applied to the size of the adapted software to get its equivalent size. The normal Adaptation Adjustment Factor (AAF) is computed as:

$$AAF = (0.4 \cdot DM) + (0.3 \cdot CM) + (0.3 \cdot IM)$$  \hspace{1cm} (3.1)

where:

- **% Design Modified (DM)**
  The percentage of the adapted software’s design which is modified in order to adapt it to the new objectives and environment. This can be a measure of design elements changed such as UML descriptions.

- **% Code Modified (CM)**
  The percentage of the adapted software’s code which is modified in order to adapt it to the new objectives and environment. Code counting tools can be used to measure CM. See the chapter on the Unified Code Count tool in Appendix 9.2 for its capabilities, sample output and access to it.

- **% Integration Required (IM)**
  The percentage of effort required to integrate the adapted software into an overall product and to test the resulting product as compared to the normal amount of integration and test effort for software of comparable size.

Reused software has $DM = CM = 0$. IM is not applied to the total size of the reused software, but to the size of the other software directly interacting
with it. It is frequently estimated using a percentage. Modified software has CM > 0.

3.2.2.3 Total Equivalent Size

Using the AAF to adjust Adapted Code size, the total equivalent size is:

\[ \text{Total Equivalent Size} = \text{New Size} + (\text{AAF} \cdot \text{Adapted Size}) \]  

AAF assumes a linear effort relationship, but there can also be nonlinear effects. Data indicates that the AAF factor tends to underestimate modification effort [51] [8] [53]. Two other factors used to account for these effects are Software Understanding and Programmer Unfamiliarity. These two factors and their usage are discussed in Appendix A.

3.2.2.4 Volatility

Volatility is requirements evolution and change, but not code thrown out. To account for the added effort, volatility is expressed as an additional percentage to size to obtain the total equivalent size for estimation.

\[ \text{Total Equivalent Size} = \text{New Size} + (\text{AAF} \cdot \text{Adapted Size}) \cdot (1 + \text{Volatility}) \]

3.3 Functional Size Measures

3.3.1 Function Points

The function point cost estimation approach is based on the amount of functionality in a software project and a set of project factors [22]. Function points measure software size by quantifying the information processing functionality associated with major external data or control input, output, or file types. Five user function types should be identified as defined in Table 3.4. Each instance of these function types is then classified by complexity level. The complexity levels determine a set of weights, which are applied to their corresponding function counts to determine the Unadjusted Function Points (UFP) quantity. The steps for counting UFPs are listed below.

1. Determine function counts by type. The unadjusted function counts should be counted by a lead technical person based on information in the software requirements and design documents. The number of each of the five user function types should be counted (Internal Logical File (ILF), External Interface File (EIF), External Input (EI), External Output
TABLE 3.4: User Function Types

<table>
<thead>
<tr>
<th>Function Point</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>External Input (EI)</td>
<td>Count each unique user data or user control input type that enters the external boundary of the software system being measured.</td>
</tr>
<tr>
<td>External Output (EO)</td>
<td>Count each unique user data or control output type that leaves the external boundary of the software system being measured.</td>
</tr>
<tr>
<td>Internal Logical File (ILF)</td>
<td>Count each major logical group of user data or control information in the software system as a logical internal file type. Include each logical file (e.g., each logical group of data) that is generated, used, or maintained by the software system.</td>
</tr>
<tr>
<td>External Interface Files (EIF)</td>
<td>Files passed or shared between software systems should be counted as external interface file types within each system.</td>
</tr>
<tr>
<td>External Inquiry (EQ)</td>
<td>Count each unique input-output combination, where input causes and generates an immediate output, as an external inquiry type.</td>
</tr>
</tbody>
</table>

(EO), and External Inquiry (EQ)). See [22] for more detailed interpretations of the counting rules for those quantities.

2. Determine complexity-level function counts. Classify each function count into Low, Average and High complexity levels depending on the number of data element types contained and the number of file types referenced. The weights in Tables 3.5, 3.6 and 3.7 are used.

3. Apply complexity weights. Weight the number of function types at each complexity level using the weights reflecting relative effort required to implement the function in Table 3.8.

4. Compute Unadjusted Function Points. Add all the weighted functions counts to get one number, the Unadjusted Function Points.

The usual Function Point procedure involves assessing the degree of influence of fourteen application characteristics on the software project determined
TABLE 3.5: Function Point Counting Weights for Internal Logical Files and External Interface Files

<table>
<thead>
<tr>
<th>Record Elements</th>
<th>1 - 19</th>
<th>20 - 50</th>
<th>51+</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Low</td>
<td>Low</td>
<td>Avg.</td>
</tr>
<tr>
<td>2 - 5</td>
<td>Low</td>
<td>Avg.</td>
<td>High</td>
</tr>
<tr>
<td>6+</td>
<td>Avg.</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

TABLE 3.6: Function Point Counting Weights for External Outputs and External Inquiries

<table>
<thead>
<tr>
<th>Data Elements</th>
<th>1 - 5</th>
<th>6 - 19</th>
<th>20+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or 1</td>
<td>Low</td>
<td>Low</td>
<td>Avg.</td>
</tr>
<tr>
<td>2 - 3</td>
<td>Low</td>
<td>Avg.</td>
<td>High</td>
</tr>
<tr>
<td>4+</td>
<td>Avg.</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

TABLE 3.7: Function Point Counting Weights for External Inputs

<table>
<thead>
<tr>
<th>Data Elements</th>
<th>1 - 4</th>
<th>5 - 15</th>
<th>16+</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 or 1</td>
<td>Low</td>
<td>Low</td>
<td>Avg.</td>
</tr>
<tr>
<td>2 - 3</td>
<td>Low</td>
<td>Avg.</td>
<td>High</td>
</tr>
<tr>
<td>3+</td>
<td>Avg.</td>
<td>High</td>
<td>High</td>
</tr>
</tbody>
</table>

according to a rating scale of 0.0 to 0.05 for each characteristic. The 14 ratings are added together, and added to a base level of 0.65 to produce a general characteristic adjustment factor that ranges from 0.65 to 1.35. This is used as a multiplier with UFP to obtain the Adjusted Function Point (AFP) count. For consistency, not all of the cost models use this adjustment because they already include cost factors for these.

The cost estimation models based on lines of code require conversion of function points into to lines. This is also called backfiring. Conversion, or backfiring ratios for different programming languages can be determined with available historical data. It is recommended to determine your own ratios for your local environment. Some sources for programming language conversion factors include [19] and [22].
TABLE 3.8: Unadjusted Function Point Complexity Weights

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Low</th>
<th>Average</th>
<th>High</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internal Logical Files</td>
<td>7</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>External Interfaces Files</td>
<td>5</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>External Inputs</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>External Outputs</td>
<td>4</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>External Inquiries</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
</tbody>
</table>

3.4 Development Effort

3.4.1 Activities and Lifecycle Phases

Software development involves much more activity than just coding. It includes the work involved in developing requirements, designs and tests. It involves documentation and reviews, configuration management, and quality assurance. It can be done using different life cycles and different ways of organizing the work (matrix, product lines, etc.). Using the SRDR as the basis, the following work activities/phases are included or excluded for effort.

TABLE 3.9: Effort Activities

<table>
<thead>
<tr>
<th>Activity</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Conceptualization</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Systems Requirements Development</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Software Requirements Analysis</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Software Architecture and Detailed Design</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Software Coding and Unit Test</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Software and System Integration</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Hardware / Software Integration and Test</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>System Test and Evaluation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Operational Test and Evaluation</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Production</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

Software requirements analysis includes any prototyping activities. The excluded activities are normally supported by software personnel but are considered outside the scope of their responsibility for effort measurement. Systems Requirements Development includes equations engineering (for derived requirements) and allocation to hardware and software.

All these activities include the effort involved in documenting, reviewing
and managing the work-in-process. These include any prototyping and the conduct of demonstrations during the development.

Transition to operations and operations and support activities are not addressed by these analyses for the following reasons:

- They are normally accomplished by different organizations or teams.
- They are separately funded using different categories of money within the DoD.
- The cost data collected by projects therefore does not include them within their scope.

From a life cycle point-of-view, the activities comprising the software life cycle are represented for new, adapted, reused, generated and COTS (Commercial Off-The-Shelf) developments. Reconciling the effort associated with the activities in the Work Breakdown Structure (WBS) across life cycle is necessary for valid comparisons to be made between results from cost models.

### 3.4.2 Labor Categories

The labor categories included or excluded from effort measurement are another source of cost data variation. The categories are a decomposition of effort. Most large software projects have staff fulfilling the functions of:

- Project Managers
- Application Analysts
- Implementation Designers
- Programmers
- Testers
- Quality Assurance personnel
- Configuration Management personnel
- Librarians
- Database Administrators
- Documentation Specialists
- Training personnel
- Other support staff

### TABLE 3.10: Effort Phases

<table>
<thead>
<tr>
<th>Phase</th>
<th>Includes</th>
<th>Excludes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Elaboration</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Construction</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Transition</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
Adding to the complexity of measuring what is included in effort data is that staff could be fulltime or part time and charge their hours as direct or indirect labor. The issue of capturing overtime is also a confounding factor in data capture.

3.4.3 Labor Hours

Labor hours, also called Staff Hours or Person-Hours, is the recommended form of measuring software development effort. This measure can be transformed into Labor Weeks, Labor Months and Labor Years. For modeling purposes, when weeks, months or years is required, choose a standard and use it consistently, e.g. 152 labor hours in a labor month.

If data is reported in units other than hours, additional information is required to ensure the data is normalized. Each reporting Organization may use different amounts of hours in defining a labor week, month or year. For whatever unit being reported, be sure to also record the Organization’s definition for hours in a week, month or year. See the Software Engineering Institute’s Framework for Counting Staff-Hours and Reporting Schedule Information [21] for a more detailed discussion.

3.5 Development Duration

Schedule data are the start and end date for different development phases, such as those shown in Table 3.10. Another important aspect of schedule data is entry or start and exit or completion criteria each phase. The criteria could vary between projects depending on its definition. As an example of exit or completion criteria, are the dates reported when:

- Internal reviews are complete
- Formal review with the customer is complete
- Sign-off by the customer
- All high-priority actions items are closed
- All action items are closed
- Products of the activity / phase are placed under configuration management
- Inspection of the products are signed-off by QA
- Management sign-off

An in-depth discussion is provided in the Software Engineering Institute’s Framework for Counting Staff-Hours and Reporting Schedule Information [47].
Chapter 4

Data Assessment

This chapter discusses transforming software engineering cost data into useful information for use in creating Cost Estimating Relationships (CERs) and productivity benchmarks for use in cost estimation and management oversight. There can be many challenges encountered when preparing cost data for analysis. The list below shows common problems with cost datasets:

- Inadequate information on modified code (only size provided)
- Inadequate information on size change or growth
- Size measured inconsistently
- Inadequate information on average staffing or peak staffing
- Inadequate or ineffective information on personnel experience
- Inaccurate effort data in multi-build components
- Missing effort data for all software activities
- Replicated duration (start and end dates) across components
- Inadequate information on schedule compression
- Missing schedule data
- No quality (defect) data

The remedy for some of these challenges is to find a way to normalize the data to the definitions discussed in 3. Other techniques are required to fill in missing data, either by consulting other sources or using statistical means or medians to fill in missing values. A process workflow was developed to assess the data and make it usable.

The data assessment and processing workflow has six steps. This workflow was used in the analysis of the SRDR data by the contributors to this manual. Each of these steps is described in detail in subsequent sections.

1. Gather Collected Data
2. Inspect each Data Record
3. Determine Data Quality Levels.
4. Correct Missing or Questionable Data
5. Normalize Size and Effort Data
6. Convert Raw SLOC to Equivalent SLOC
4.1 Gather Collected Data

The SRDR has evolved over time. There is required data in a SRDR that is not relevant for cost estimation analysis. Likewise, there is missing contextual data that is needed for analysis. The SRDR data necessary for cost estimation analysis requires detail information on:

- Amount of workload (expressed as source lines of code)
- Development and support effort effort
- Project or build duration

Additional contextual data needed for analysis are:

- What does the application do, i.e., a functional description?
- Does the record contain estimated data?
- Does the record represent aggregated data or does it represent a single Computer Software Configuration Item (CSCI)?
- Where in the acquisition lifecycle the software was developed, e.g., technology demonstration, engineering and manufacturing, operations and sustainment?
- Are there extenuating circumstances during software development that would impact size, effort or schedule?

4.2 Inspect each Data Record

The SRDR data that is relevant for data analysis is inspected for completeness, integrity, and reasonable-ness. The first activity is to examine the project context information.

**Project Context**

- Is this record a roll-up of multiple CSCIs or a single item?
- Is this record an aggregation of multiple builds or a single build?
- How would this software component be characterized?
  - What does this component do?
  - Were there any extenuating circumstances concerning development, e.g., management change, large requirements change, stop / restart work?
  - Is the Data Dictionary available for this record?
  - Is there any additional information that can be consulted about the data during analysis, such as:
Data Assessment

* Acquisition Strategy
* Acquisition Support Plan (ASP)
* Contract Plan
* Cost Analysis Requirements Document (CARD)
* Capability Description Document (CDD)
* Software Requirements Specification (SRS)
* Work Breakdown Structure (WBS)
* Earned Value Management System data (EVMS)
* Software Development Plan (SDP)
* System Engineering Plan (SEP)

Next, the size, effort, schedule and productivity data for each record are examined.

Size Data

• Does the size data look sound?
  – Does the size data match the record description, e.g. one million SLOC size may indicate program level data with multiple CSCIs?
  – Is the size part of multiple builds or releases, e.g. is the size data an aggregated number?
  – Was much or all software auto-generated? This will result in an extraordinary productivity unless converted to an equivalent size as discussed in 4.6.
  – Was code rewritten after being auto-generated? This means the auto-code was modified. It should be treated as modified code in this case.

• Was a portion of a legacy system included in the sizing data?
  – How much software was adapted (modified)?
  – How much software was reused (no changes)?

• Is there repeating size data? If there is repeating size data, this record may be an update of another record.

Effort Data
The effort data are for the activities as discussed in Section 2.5.1.4.

• What labor was included in the reported hours?
  – Engineering labor
  – Management labor
  – Support labor: CM, QA, Process Improvement, Safety, Security, Dev. Environment support

• What labor was reported in the “Other” activity?
  – Was requirement analysis effort reported for all builds?
  – Were there continuous integration activities across all builds?
Schedule Data

- Was there schedule compression mentioned on the project?
- Were there multiple parallel builds (same start and end date)?

Productivity Screening

Productivity is the ratio of size to effort as detailed in Section 5.5.1.

- Is a check of the productivity reasonably close to software with similar functionality? Productivities for similar software functionality are grouped by Application Type (discussed in Section 5.5.3).
- Is this record an outlier in a scatter plot with other similar data? This check can only be done if you have access to the SRDR for the appropriate Application Type.

4.3 Determine Data Quality Levels

From the inspection process, assign the SRDR record a data quality rating. The ratings range from 1.0, Best, to 0.0, Worst. The criteria in Table 4.1 can be used to determine rating values based on:

1. Size - counts exist for the different code types as discussed in Table 3.1 (providing size data is present).
2. Size Count Type - the rules for counting the code are explicit as discussed in Section 3.2.1.2.
3. ESLOC Factors - the ESLOC factors for Modified, Reused and Auto-Generated code exist as discussed in Section 3.2.2.2.
4. CSCI-level Data - the data is clearly for one CSCI or it is roll-up data.
5. Effort - effort hours exist for all activities as discussed in Section 2.5.1.4.
6. Schedule - activity duration data exists for same activities as effort.
7. Productivity - productivity is within range of similar software CSCIs.

As each record is rated by the criteria in Table 4.1, an overall quality level is assigned by:

\[
Quality \ Level = \frac{(Size + Size \ Count \ Type + ESLOC \ factor + CSCI \ level + Effort + Schedule + Productivity)}{7}
\]

The quality level is a quick indicator of the degree of issues found in the record. As the record is corrected through supplemental information, the rating is revised. As the rating value decreases, the record becomes less useful for CER and productivity analysis. If all size or effort data is missing, the record cannot be used.

Because the range of the quality level scale is between 0.0 and 1.0, it could be used as a weight during analysis.
### TABLE 4.1: Data Quality Rating Scale

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Value</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>1</td>
<td>Size data present</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>No size data</td>
</tr>
<tr>
<td>Size Count Type</td>
<td>1</td>
<td>Size is Logical SLOC</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>Size is Non-Commented Source Statements</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>Size is Physical Lines (Comment and Source Statements)</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>Size is Total Lines (all lines in file: blank, comment, source)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>No size data</td>
</tr>
<tr>
<td>ESLOC Factors</td>
<td>1</td>
<td>Modification factors are provided for Auto-Gen, Modified &amp; Reuse code counts from outside sources (sometimes found in the Data Dictionary or after contacting the Data Submitter)</td>
</tr>
<tr>
<td>(See Section 3.2.2.2)</td>
<td>0.75</td>
<td>New SLOC and no size data for Auto-Gen, Modified or Reuse (this could happen but it is rare)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>No modification factors are available for either Modified, Auto-Gen, or Reused SLOC counts (including the guidance provided in Table 4.3)</td>
</tr>
<tr>
<td>CSCL-level Data</td>
<td>1</td>
<td>5,000 &lt; Equivalent SLOC &lt; 250,000</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Equivalent SLOC &lt; 5,000 or Equivalent SLOC &gt; 250,000 (if greater than 250,000 may be an indicator of program-level data)</td>
</tr>
<tr>
<td>Effort</td>
<td>1</td>
<td>Effort reported for all phases (as discussed in Section 2.5.1.4)</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>Effort is reported as a total (all hours aggregated)</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Effort is missing for a phase</td>
</tr>
<tr>
<td>Schedule</td>
<td>1</td>
<td>Duration reported for all phases (as discussed in Section 2.5.1.4)</td>
</tr>
<tr>
<td></td>
<td>0.75</td>
<td>Duration is reported as a total (all durations aggregated)</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>Duration is missing for one phase</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Duration data is missing</td>
</tr>
<tr>
<td>Productivity</td>
<td>1</td>
<td>Record is within 1 standard deviation from the mean (see Section 5.5.3 for examples)</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>Record is within between 1 and 3 standard deviations from the mean</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>Record is a clear outlier</td>
</tr>
</tbody>
</table>
4.4 Correct Missing or Questionable Data

The quality level identifies SRDR records that need additional work. There are several approaches available to resolving missing or questionable data. These are listed in a recommended order:

1. Consult the accompanying SRDR Data Dictionary discussed in Chapter 2.
2. Consult any supplemental information on the project that is available, e.g., ASP, CARD, CDD, EVMS, SRS, WBS, SDP, SEP, etc.
3. Scheduling follow-up meetings with SRDR data contributor.

SRDR data quality issues that were fixed in the past by the SRDR contributor may include:

- Revised missing size, effort and duration data
- Obtained ESLOC factors
- Confirmed Application Type and Operating Environment
- Confirmed CSCI-level of reporting
- Asked about problems with - high / low, long / short - size, effort and duration data

As a result of inspecting the data and attempting to correct the issues found, no "bad" data or outliers are excluded from the analysis on arbitrary grounds. However, data with issues that cannot be resolved are excluded from analysis.

4.5 Normalize Size and Effort Data

Normalizing data is making a type of data the same. For example, if SLOC was measured by different criteria, all SLOC counts are converted into a common count method. If effort data covers different lifecycle phases, all effort data is converted to cover the same phases. Normalization reduces noise in the data. Otherwise, it will pose a significant threat to statistical validity.

4.5.1 Converting to Logical SLOC

With the SRDR data, the SLOC were counted using different methods (see the full explanation in Section 3.2.1.2).

- Total Count: a line in a file, e.g. carriage returns including blanks and comment lines
Data Assessment

- Non-Commented Source Statements (NCSS) Count: a line in a file that is not a blank or comment line
- Logical Count: as defined in Chapter 3.

For analysis, the definition of a source line of code needs to be as consistent as possible to eliminate noise in the data. A logical source line of code has been selected as the baseline SLOC definition in Table 3.2.

If a source line of code count was defined as either Total or NCSS, these counts were converted to a Logical SLOC count. The conversion factors are derived from counting public domain software applications and additional contributions from USC-CSSE Affiliates using the Unified Code Count tool [41].

The statistics from the counting experiment produced conversion factors for converting code counted using NCSS rules and Total rules are show below. By inspecting Table 3.2, it can be observed that Logical counting rules count source code statements that effect software application execution. The NCSS and Total count rules show that increasing more statements are counted, hence the factors below are less than one because NCSS and Total counts are reduced to an equivalent Logical count.

\[
\text{Logical SLOC count} = 0.66 \cdot \text{NCSS count}
\]  

(4.1)

\[
\text{Logical SLOC count} = 0.34 \cdot \text{Total count}
\]  

(4.2)

Most of the SRDR SLOC count data was produced using Logical counting rules. However, the Data Dictionary sometimes revealed that counts labeled as Logical were in fact NCSS or Total counts. It is very worthwhile to verify the count type by consulting the Data Dictionary.

4.5.2 Standardizing Effort

Software CERs have a breadth and a depth. The breadth corresponds to the number of lifecycle activities covered and the depth is the type of labor counted in or across each activity. A CERs breadth and depth are defined by the data used to derive them. It is important to use data that is homogeneous in the number of activities covered and the labor categories included.

Normalizing effort consists of selecting records for analysis that cover the same activities and labor categories. This is made challenging because some SRDR records contain development data where effort for an activity was expended and recorded in another record. The requirements activity is an example. The first build of a software CSCI may have all of the requirements activity for that build and subsequent builds, i.e., requirements were analyzed for all builds in the first build. The remedy is to amortize the hours for the requirements activity across all builds using software size as a proportioning factor.

The activity data in the SRDR follows the ISO 12207 [25] processes for software development. These are listed below. The ones covered by SRDR data are in bold. This is the breadth of the CERs reported in this manual.

- System requirements analysis
- System architectural design
- Software requirements analysis
- Software architectural design
- Software detailed design
- Software coding and testing
- Software integration
- Software qualification testing
- System integration
- System qualification testing
- Software installation
- Software acceptance support

The SRDR effort hours also cover a number of labor categories. These categories are discussed in the companion Data Dictionary. The different labor categories in the SRDR data are shown in Table 4.2. Not all of the records had all of the categories. However, the Software Engineering and Assessment categories were reported for in each record.

When using a CER or any cost model, it is important to know the breadth and depth of activities covered by the estimate.

4.6 Convert Raw SLOC into Equivalent SLOC

Equivalent size is a method used to make new and adapted code equivalent so they can be rolled up into an aggregate size estimate (discussed in Section 3.2.2.2). Equivalent Source Lines of Code (ESLOC) is a synthetic measure and is not a actual measure of software size per [7]:

- It is not a measure of software effort
- It is not a measure of delivered software capability
- It is a quantity derived from software component sizes and reuse factors that helps estimate effort
- Once a product or increment is developed, its ESLOC loses its identity
  - Its size expands into full SLOC
  - Can apply reuse factors to this to determine an ESLOC quantity for the next increment (but this would also have no relation to the product’s size)

For new, modified, reused and auto-generated code sized, ESLOC is derived by the formula below:
**TABLE 4.2: SRDR Activities**

<table>
<thead>
<tr>
<th>Category</th>
<th>SRDR Labor Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management</td>
<td>Engineering Management</td>
</tr>
<tr>
<td></td>
<td>Business Management</td>
</tr>
<tr>
<td>Software Engineering</td>
<td>Software Requirements Analysis</td>
</tr>
<tr>
<td></td>
<td>Architecture and Detailed Design</td>
</tr>
<tr>
<td></td>
<td>Coding and Unit Testing</td>
</tr>
<tr>
<td></td>
<td>Test and Integration</td>
</tr>
<tr>
<td>Assessment</td>
<td>Qualification Testing</td>
</tr>
<tr>
<td></td>
<td>Development Test Evaluation Support</td>
</tr>
<tr>
<td>Support</td>
<td>Software Configuration Management</td>
</tr>
<tr>
<td></td>
<td>Software Quality Assurance</td>
</tr>
<tr>
<td></td>
<td>Configuration Audit</td>
</tr>
<tr>
<td></td>
<td>Development Environment Support</td>
</tr>
<tr>
<td></td>
<td>Tools Support</td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
</tr>
<tr>
<td></td>
<td>Data Preparation</td>
</tr>
<tr>
<td></td>
<td>Process Management</td>
</tr>
<tr>
<td></td>
<td>Metrics</td>
</tr>
<tr>
<td></td>
<td>Training</td>
</tr>
<tr>
<td></td>
<td>IT Support / Data Center</td>
</tr>
</tbody>
</table>

\[
ESLOC = \text{New SLOC} + (AAF_M \cdot \text{Modified SLOC}) + (AAF_R \cdot \\
\text{Reused SLOC}) + (AAF_{AG} \cdot \text{AutoGenerated SLOC})
\]

Where \( AAF_i = (0.4 \cdot DM) + (0.3 \cdot CM) + (0.3 \cdot IM) \).

The data in an SRDR does not require the submission of values for DM, CM and IM. Sometimes the values were provided in the Data Dictionary. Sometimes the data submitter was contacted for the values. However, there were data remaining with no DM, CM or IM values. Guidelines for filling-in missing data were derived from the SRDR data that was available.

As shown in the equation above, there are four code types: New, Modified, Reused, and Auto-Generated. The DM, CM and IM factors are not required for each type.

- **New code** does not require any adaption factors. Nothing has been modified.
- **Modified code** requires all three factors for DM, CM and IM, representing modifications to the code’s design, code and integration testing.

- **Reused code** does not require the DM or CM adaption factors but is reasonable to assume that it will require testing captured with the IM factor. If Reused code does require modification, then it becomes Modified code and the adaptation factors for Modified code apply.

- **Auto-Generated code** does not require the DM or CM adaption factors. However, it does require testing, IM. If Auto-Generated code does require modification, then it becomes Modified code and the adaptation factors for Modified code apply.

Table 4.3 shows median values for DM, CM and IM derived from the subset of SRDR data that had values for each factor. The median values were used because the data was not normally distributed. The table shows the code type, the number of data points, the range of DM, CM and IM values, the median (Mdn.) value, and the resulting AAF value.

**TABLE 4.3: Proxy DM, CM, and IM Values**

<table>
<thead>
<tr>
<th>Code Type</th>
<th>#</th>
<th>Range Mdn.</th>
<th>DM Mdn.</th>
<th>CM Range Mdn.</th>
<th>IM Range Mdn.</th>
<th>AAF</th>
</tr>
</thead>
<tbody>
<tr>
<td>New</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>1.0</td>
<td></td>
</tr>
<tr>
<td>Modified</td>
<td>101</td>
<td>0-100%</td>
<td>31%</td>
<td>1-100%</td>
<td>3-100%</td>
<td>0.47</td>
</tr>
<tr>
<td>Reused</td>
<td>145</td>
<td>0%</td>
<td>0%</td>
<td>0-100%</td>
<td>0-100%</td>
<td>0.03</td>
</tr>
<tr>
<td>Auto-Gen</td>
<td>6</td>
<td>0%</td>
<td>0%</td>
<td>0-100%</td>
<td>0-100%</td>
<td>0.15</td>
</tr>
</tbody>
</table>

The median values in Table 4.3 should be used with caution. For instance, the IM factor for the amount of modification to existing test and integration procedures may be 100% or more if the software application has high reliability or security requirements or the integration is complex. For modified software, the DM and CM values for the amount of design and code modified may be higher than the median values due to additional information assurance requirements, time and storage constraints, and product complexity. Conversely the reused and auto-generated IM value may be below median if the software has already been certified in an earlier release or if the software is a prototype.

### 4.7 Future Work and Limitations

The Data Quality Rating Scale in Table 4.1 presents values meant to reflect the quality of the data for CER analysis. Full and complete data is assigned the highest score, one (1), and missing data the lowest score, zero (0). Assuming these two values establish an acceptable scale, the intermediate values are
subjectively assigned. Subjective assignment means the intermediate value and its distance from other intermediate values cannot be verified. The Quality Level formula also assumes that each quality category has an equal weight to the overall rating. This is an assumption and may not be valid either. This section should be used with caution.

A better approach to describing the quality of data is to discuss the lessons learned in reviewing SRDR data grouped into categories such as size, effort, and schedule. It is also not necessary to assign numerical values to the quality of data. Something as simple as using colors (red, yellow, green) would be sufficient. The next edition of this manual will replace the Data Quality Rating Scale with the lessons learned approach.

The guidelines for converting raw SLOC counts for the different code types into equivalent SLOC have gone through many revisions. Providing guidelines is difficult when there is no insight into how much existing software was modified, how well the software is structured for modification, and how much experience existed in understanding how the software implemented it functionality.

- Auto-generated code does require some amount of effort to specify before generation. Yet, %DM is set to zero.
- A guideline for the Deleted code type is not provided. The SRDR is now requesting deleted code counts. This will provide future data that may reveal its contribution to equivalent SLOC.

As more SRDR data is collected, these guidelines will expand and improve.
Chapter 5

Cost Estimating Relationships

5.1 Overview

This chapter discusses an approach to building Cost Estimating Relationships (CERs) and Benchmarks. Figure 5.1 shows the process starting with SRDR data and other supporting historical data. Defining, collecting, evaluating and normalizing historical data were discussed in previous chapters. Data segmentation will be used to partition the data into groups with common project characteristics. Each segment will have its own CER and benchmark.

Each segment is examined for the presence of a relationship between a single predictor and single response variable. The statistical method for Ordinary Least Squares (OLS) regression for one predictor and one response variable is used to quantify the relationship. This approach simplifies CER formulation by eliminating possible effects of multicollinearity found in multivariable models. In the case of a cost estimating relationship, software size is
the predictor and effort is the response. Where relationships exist, CERs and benchmarks are created.

5.2 Data Segmentation

Factors such as application complexity; impact of loss due to reliability; autonomous modes of operation; constraints on timing, storage, and power; security requirements; and complex interfaces influence the cost and time to develop applications. Parametric cost models have a number of adjustable parameters that account for these factors.

This chapter describes developing simple CERs based on grouping software applications with similar characteristics instead of creating CERs with many parameters. This makes for practical estimation and enables better precision within application types vs. a calibration to the global dataset of diverse applications. The groupings have evolved over time starting with a set of 29 software application groups presented at our research workshops. A survey was conducted across best practices of other software cost model vendors in an effort to reduce the number of groups. The resulting set of 17 groups presented in this chapter have also been submitted to the DoD SRDR Working Group sponsored by DCARC. They are currently being adopted as Application Domains for use in filling out the SRDR data collection form.

Two data classification groups are used to develop CERs and Benchmarks. The Application Type groups are based on similar functional characteristics, while the Operating Environment groups distinguish the environments in which the applications operate. Both Operating Environment (OpEnv) and Application Type (AppType) are considered in the CERs and benchmarks.

5.2.1 Operating Environments (OpEnv)

Operating Environments have similar systems, similar products, similar operational characteristics, and similar requirements. These aspects include:

- High-speed vehicle versus stationary
- Battery operated versus ground power
- Unrecoverable platform versus recoverable, accessible platform
- Limited, non-upgradeable computing processor capacity versus racks of processors
- Fixed internal and external memory capacity versus expandable capacity

The Operating Environments for Ground, Maritime, Aerial, Space and Ordnance decompose into 11 groups when fixed versus mobile and manned versus unmanned are considered. These high-level environments described in Table 5.1 are useful when there is a lack of data at the lower levels.
<table>
<thead>
<tr>
<th>Operating Environment (OpEnv)</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ground Site (GS)</td>
<td>Command Post, Ground Operations Center, Ground Terminal, Test Faculties</td>
</tr>
<tr>
<td>Mobile (GSM)</td>
<td>Intelligence gathering stations mounted on vehicles, Mobile missile launcher</td>
</tr>
<tr>
<td>Ground Vehicle (GV)</td>
<td>Tanks, Howitzers, Personnel carrier</td>
</tr>
<tr>
<td>Ground Vehicle (GV)</td>
<td>Manned (GVM)</td>
</tr>
<tr>
<td>Unmanned (GVU)</td>
<td>Robotic vehicles</td>
</tr>
<tr>
<td>Maritime Vessel (MV)</td>
<td>Aircraft carriers, destroyers, supply ships, submarines</td>
</tr>
<tr>
<td>Maritime Vessel (MV)</td>
<td>Manned (MVM)</td>
</tr>
<tr>
<td>Unmanned (MVU)</td>
<td>Mine hunting systems, Towed sonar array</td>
</tr>
<tr>
<td>Aerial Vehicle (AV)</td>
<td>Fixed-wing aircraft, Helicopters</td>
</tr>
<tr>
<td>Aerial Vehicle (AV)</td>
<td>Manned (AVM)</td>
</tr>
<tr>
<td>Unmanned (AVU)</td>
<td>Remotely piloted air vehicles</td>
</tr>
<tr>
<td>Space Vehicle (SV)</td>
<td>Passenger vehicle, Cargo vehicle, Space station</td>
</tr>
<tr>
<td>Space Vehicle (SV)</td>
<td>Manned (SVM)</td>
</tr>
<tr>
<td>Unmanned (SVU)</td>
<td>Orbiting satellites (weather, communications), Exploratory space vehicles</td>
</tr>
<tr>
<td>Ordnance Vehicle (OV)</td>
<td>Air-to-air missiles, Air-to-ground missiles</td>
</tr>
<tr>
<td>Ordnance Vehicle (OV)</td>
<td>Unmanned (OVU)</td>
</tr>
</tbody>
</table>
5.2.2 Application Types (AppType)

Application types are groups of software applications with similar functions and attributes. They are characterized by the following aspects:

- Required software reliability
- Database size, if there is a large data processing and storage component to the software application
- Product complexity
- Integration complexity
- Real-time operating requirements
- Platform volatility or target system volatility
- Special display requirements
- Development re-hosting
- Quality assurance requirements
- Security requirements
- Assurance requirements
- Required testing level

The Application Types are described in Table 5.2.

<table>
<thead>
<tr>
<th>Application Type (AppType)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microcode and Firmware (M&amp;F)</td>
<td>Microcode and Firmware is software stored on target hardware devices that do not have hard disks and use programmable logic devices. It is a combination of persistent memory and the program code and data stored in it.</td>
</tr>
</tbody>
</table>

Examples: Field Programmable Gate Arrays (FPGAs); Microwave controllers; Field Programmable Logic (FPL); Electronic Programmable Logic Device (EPLD); Application Specific Integrated Circuit (ASIC); Programmable Read-Only Memory (PROM); Erasable Programmable Read-Only Memory (EPROM); Electrically Erasable Programmable Read-Only Memory (EEPROM); Complex Programmable Logic Device (CPLD); Programmable Array Logic (PAL).
<table>
<thead>
<tr>
<th>Application Type (AppType)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sensor Control and Signal Processing (SCP)</td>
<td>Software that requires timing-dependent device coding to enhance, transform, filter, convert, or compress data signals. Examples: Lasers; Sonar; Acoustic; Electromagnetic; Signal Processor; Radar Altimeter; Photographic Sensors; Motion Sensors; Infrared Sensors; Sensor Assembly; Electronic Sensors; Seeker Assembly; Signal Electronics; Optical Assembly; Tracking Sensors; Antenna Assembly.</td>
</tr>
<tr>
<td>Vehicle Control (VC)</td>
<td>Software necessary for the control of vehicle primary and secondary mechanical devices and surfaces. Examples: Flight Control; Electrical Power; Hydraulic; Fuel Subsystem; Propulsion; Attitude Control System; Structures &amp; Mechanisms; Bus Flight Software; Thermal Control; Landing Gear; Controls software; Thrust Vector Actuation; Executive.</td>
</tr>
<tr>
<td>Vehicle Payload (VP)</td>
<td>Software which controls and monitors vehicle payloads and provides communications to other vehicle subsystems and payloads. Examples: Fire Control; Mine Warfare; Electronic Attack subsystem controller; Weapons Delivery and Control; Gun fire control system; Missile fire control systems; Antisubmarine warfare fire control and torpedo fire control systems; Pointing; Command &amp; Control Interface; Payload Flight Software; Armament; Survivability Payload; Reconnaissance Payload; Electronic Warfare Payload; Armament/Weapons Delivery; Intelligence; Surveillance Reconnaissance Payload; Mission Payload.</td>
</tr>
</tbody>
</table>

(continued)
### TABLE 5.2: Application Types Definition (continued)

<table>
<thead>
<tr>
<th>Application Type (AppType)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real Time Embedded (RTE)</td>
<td>Interrupt-driven, embedded software in military and consumer appliances, devices, and products, possibly directing and processing sensor inputs/outputs, generally with a very small executive for an operating system interface to basic processor(s).</td>
</tr>
<tr>
<td></td>
<td>Examples: Embedded Electronics/ Appliance; Robotics; PDAs; Telemetry; Tracking &amp; Command (TT&amp;C); Guidance; Navigation and Control; Controls and Displays; Data Links; Radios (device); Remote Control; Receiver; Transmitter; Exciter; Bombing Computer; Video and recorders; Telephones (device); Built-in-Test.</td>
</tr>
<tr>
<td>Command and Control (C&amp;C)</td>
<td>Software that allows humans to manage a dynamic situation and respond in real time.</td>
</tr>
<tr>
<td></td>
<td>Examples: Mission Management; Mission Computer Processing; Mission Control; Command processing; Air traffic control; Data reduction/ analysis; Telemetry Processing; Battlefield command; Battle management.</td>
</tr>
<tr>
<td>Communications (COM)</td>
<td>The transmission of information, e.g. voice, data, commands, images, and video across different mediums. Primarily software systems that control or manage transmitters, receivers and communications channels.</td>
</tr>
<tr>
<td></td>
<td>Examples: Switches; Routers; Integrated circuits; Multiplexing; Encryption; Broadcasting; Transfer modes; Radios (networks); Network management; Network Operations; Satellite communications; Telecommunications; Networks (WAN/LAN); Protocols (VOIP, TCP/IP, PKI, etc.).</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Application Type (AppType)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Software (SYS)</td>
<td>Layers of software that sit between the computing platform and applications. Examples: Operating Systems; Infrastructure; Framework; Middleware; Device Driver; Display Drivers; File Management; Image Processing; Interface Driver; Utilities.</td>
</tr>
<tr>
<td>Process Control (PC)</td>
<td>Software that manages the planning, scheduling and execution of a system based on inputs, generally sensor driven. Examples: Temperature control; Manufacturing process control; Device or instrument control.</td>
</tr>
<tr>
<td>Scientific and Simulation (S&amp;S)</td>
<td>Non real time software that involves significant computations and scientific analysis. Examples: System Integration Lab (SIL) Simulation; Simulators; Offline Data Analysis; Expert Systems; Math &amp; Algorithm Intensive; Graphics; Statistical Analysis; Artificial Intelligence; Simulation &amp; Modeling; Engineering &amp; Science; 3D Modeling &amp; Animation; Trainer Simulations; Computer Aided Design (CAD); Weather models.</td>
</tr>
<tr>
<td>Application Type (AppType)</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Test, Measurement, and Diagnostic Equipment (TMDE)</td>
<td>Software used for testing, measuring, diagnosing, emulating, and evaluating operational hardware and software systems. Software necessary to operate and maintain systems and subsystems which are not consumed during the testing phase and are not allocated to a specific phase of testing. This does not include built-in-test (BIT). Examples: Test equipment software; Test driver; Maintenance and Diagnostic; Fault Tolerance; Diagnostic; Equipment emulators.</td>
</tr>
<tr>
<td>Training (TRN)</td>
<td>Software used for educational and training purposes. Examples: Computer Based Training (CBT); Computer Aided Instruction (CAI); Tutorial Applications; Courseware.</td>
</tr>
<tr>
<td>Software Tools (TOOL)</td>
<td>Software that is used for analysis, design, construction, or testing of computer programs. Examples: Compilers; Linker/loaders; Debuggers; Editors; Assemblers; Requirements analysis &amp; design tool aids; Code generators; Programming aids; Report generators; Code auditors; Test case data recording; Test case data reduction/analysis; Test case generation.</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Application Type (AppType)</th>
<th>Description</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mission Planning (MP)</td>
<td>Provides the capability to maximize the use of the platform. The system supports all the mission requirements of the platform and may have the capability to program onboard platform systems with routing; targeting; and performance, map, and Intel data.</td>
<td>Scenario generation; Planning &amp; Analysis; Target planning; Route planning; Fuel planning; Cargo load planning.</td>
</tr>
<tr>
<td>Custom AIS Software (CAS)</td>
<td>Software needed to build a custom software application to fill a capability gap not captured by COTS/GOTS software packages.</td>
<td>Glue code; External system interfaces; Data transformation; Inter-COTS/GOTS data exchange; Graphical User Interface; Internet Server Applet; Website.</td>
</tr>
<tr>
<td>Enterprise Service Systems (ESS)</td>
<td>Software needed for developing functionality or a software service that are unassociated, loosely coupled units of functionality that have no calls to each other embedded in them.</td>
<td>Enterprise service management; Machine-to-machine messaging; Service discovery; People and device discovery; Metadata discovery; Mediation service; Service security; Content discovery and delivery; Federated search; Enterprise catalog service; Data source integration; Enterprise content delivery network; Session management; Presence and awareness; Text collaboration; White boarding and annotation; Application sharing; Application broadcasting; Virtual spaces; Identity management; Content discovery; Collaboration; User profiling and customization.</td>
</tr>
</tbody>
</table>
### Application Types Definition (continued)

<table>
<thead>
<tr>
<th>Application Type (AppType)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise Information Systems (EIS)</td>
<td>Software needed for building an enterprise information system that uses an integrated database to support typical business processes within business/functional areas and consistent information access across areas and systems. COTS/GOTS attributed to a specific software service or bundle of services.</td>
</tr>
</tbody>
</table>

Examples: Enterprise resource planning; Enterprise data warehouse; General ledger; Accounts payable; Revenue and accounts receivable; Funds control and budgetary accounting; Cost management; Financial reporting; Real property inventory and management; Document management; Logistic or Supply Planning & Control; Transaction Processing; Management Performance Reporting; Office Information System; Reservation System; Geographic or spatial information system; Financial Transactions; Database management; Data Warehousing; Executive Information System; Internet Server Applet; Report Generation; Office Automation; Data Mining.

### 5.2.3 MIL-STD-881C WBS Mapping to Application Types

It is challenging to determine which application type should be used to estimate the cost of an application (that part of the hardware-software complex which comprise a domain). By using a work breakdown structure (WBS), the environment and domain are used to determine the application type. Using the WBS from MIL-STD-881C [39], a mapping is created from environment to Application Type (AppType):

- Environment
  - Subsystem
    * Sub-subsystem
      * Domains → AppType
Starting with the environment, traverse the WBS to the lowest level where the domain is represented. Each domain is associated with a Application Type (AppType). In real-world WBSs, the traverse from environment to AppType will most likely not be the same number of levels. However the 881C WBS provides the context for selecting the AppType and should be transferable to other WBSs.

Two examples for finding the application type using the 881C Aerial Vehicle Manned (AVM) and Space Vehicle Unmanned (SVU) WBS elements are provided below. The highest level WBS element represents the environment. In the AVM environment there are the Avionics subsystem, Fire-Control sub-subsystem. The Fire-Control sub-subsystem contains the sensor, navigation, air data, display, bombing computer and safety domains. Each domain has an associated application type:

- Air Vehicle Manned (AVM)
  - Avionics
    * Fire Control
      - Search, target, tracking sensors $\rightarrow$ SCP
      - Self-contained navigation $\rightarrow$ RTE
      - Self-contained air data systems $\rightarrow$ RTE
      - Displays, scopes, or sights $\rightarrow$ RTE
      - Bombing computer $\rightarrow$ VP
      - Safety devices $\rightarrow$ RTE
    * Data Display and Controls
      - Multi-function display $\rightarrow$ RTE
      - Control display units $\rightarrow$ RTE
      - Display processors $\rightarrow$ RTE
      - On-board mission planning $\rightarrow$ MP

For a space system, the highest level 881C WBS element is the Space Vehicle Unmanned (SVU). The two sub-systems are Bus and Payload. The domains for Bus address controlling the vehicle. The domains for Payload address controlling the onboard equipment. Note the system decomposition has one less level than the decomposition for Air Vehicle Manned. Each sub-subsystem has an associated application type:

- Space Vehicle Unmanned (SVU)
  - Bus
    * Structures & Mechanisms $\rightarrow$ VC
    * Thermal Control $\rightarrow$ VC
    * Electrical Power $\rightarrow$ VC
    * Attitude Control $\rightarrow$ VC
    * Propulsion $\rightarrow$ VC
    * Telemetry, Tracking, & Command $\rightarrow$ RTE
    * Bus Flight Software $\rightarrow$ VC
5.3 Estimating Relationships

There are many statistical techniques available to explore the relationship between X and Y. The exploratory technique described here uses the scatter plot. A scatter plot shows the relationship, if any, between X and Y. Without doing any number crunching, scatter plots immediately reveal X and Y’s relationship. By convention the predictor, X, is presented on the horizontal axis and the response, Y, on the vertical axis of the scatter plot. Examples of different relationships are shown next with example (non-real) data. Additional exploratory data analysis techniques can be found at [42].
5.3.1 Scatter Plots

The plot in Figure 5.2 shows the predictor factor X plotted against the response factor Y. For the X values there are multiple corresponding values for Y with a wide variation across the range. This lack of predictability in determining Y from a given value of X, and the associated amorphous, non-structured appearance of the scatter plot leads to the conclusion that there is no relationship.

Scatter plots are extremely insightful for assessing relationships visually. It should be noted that a Y vs. X relationship could be created from the data in Figure 5.2 using OLS regression. However, a quick visual check of the scatter plot between predictor and response shows there is an extremely weak or no relationship. Therefore the relationship should be discarded and not considered further.

Note in Figure 5.3 how a straight line comfortably fits through the data. This indicates a linear relationship between X and Y. The data scatter about the line is quite small indicating a strong relationship. The slope of the line shows a positive relationship between X and Y, i.e., as X increases, Y also increases. Visual inspection of the scatter data shows that a fitted model is appropriate and further statistical testing is warranted.

Figure 5.4 also shows a straight line comfortably fitting through the scatter data indicating a strong relationship. The negative slope of the line means there is a negative relationship between X and Y, i.e., as X increases, Y decreases. The data scatter about the line is small indicating a strong relationship for prediction.
In Figure 5.5, the scatter plot reveals a positive, homoscedastic (homoscedastic) linear relationship between X and Y. For a value of X, the value of Y varies consistently about the predicted value of Y. This is a condition called homoscedasticity, which is very important as it is an underlying assumption for the regression analysis technique used to create CERs. Its violation leads
to parameter estimates with inflated variances and CER statistics indicating poor accuracy. Thus a homogeneity of variance shown in a homoscedastic linear relationship is assumed for valid CERs.

The scatter plot in Figure 5.6 shows an approximately positive, heteroscedastic (hetero-sce-das-tic) linear relationship between X and Y. However, for any given value of X, the Y variation range about the predicted value of Y depends on the value of X. This condition is called heteroscedasticity meaning the non-constant variation in Y over the values of X. In Figure 5.6, small values of X show a small data scatter in Y while large values of X result in large data scatter in Y. OLS regression would produce larger prediction errors as X increases.

Figure 5.6 also shows another condition, the non-normal distribution of data. Normally distributed data is an underlying assumption for producing valid CERs. In the figure, the non-normal data shows up as clumping of the data scatter for a range of X values. Often with software size and effort data, the clumping occurs for lower values of size. Unless this condition is alleviated CERs should not be created from the data.

One method to overcome the effects of heteroscedasticity is by weighting of the data with noisier data being weighted less. The weight would depend on the value of X where heteroscedasticity is observed. Another approach is to perform a Y variable transformation to achieve homoscedasticity using a technique such as a Box-Cox transformation. This procedure will transform data into a normal distribution shape.
When creating CERs, the two factors of interest in SRDR data are software size and effort. Size is treated as the predictor variable (X in the discussion above) and effort is treated as the response variable (Y). Software size in SRDR data is expressed as ESLOC defined in Section 3.2.2. Effort is expressed as person hours which is then transformed into person months as described in Section 3.4.3.

SRDR size and effort data usually have a positive relationship, are non-normally distributed, and exhibit heteroscedasticity. Both conditions of non-normality and heteroscedasticity can be reduced in SRDR data by transforming the size and effort data using logarithms. This permits the use of OLS to derive CER models.

5.3.2 Data Transformation

The technique for creating CERs (discussed next) assumes the data is normally distributed. In a normal distribution the data is distributed equally above and below the mean data value. Unfortunately, software engineering cost data is rarely normally distributed. A contributing factor to this phenomena is that cost data is bounded on the lower end by zero. Another factor is there are typically more smaller size CSCIs that larger ones.

To address this issue the data should be transformed to a normal distribution before being analyzed. The Anderson-Darling statistical test can help determine if data has a normal distribution. If the test indicates non-normality, the data is transformed using logarithms. Although other transformation can be used, such as square root or squares, a logarithmic transformation has been
found to be the most effective at transforming non-normal cost data into a normally distributed data set. Analysis is carried out in log-space and the results transformed back into unit-space.

Using SRDR data as an example, Figure 5.7 shows non-normally distributed size data for the RTE AppType using a histogram on the left. When the data is transformed using logarithms, the distribution is much closer to a normal distribution per the histogram on the right.

![Histograms showing non-normal and transformed size data](image)

**FIGURE 5.7: RTE Size Data Distributions**

As discussed next, a logarithmic transformation is also used to convert a non-linear CER model form into a linear model form that can be analyzed using advanced statistical techniques.

### 5.3.3 CER Models

The cost estimating relationship between size and effort can take several forms. The following two models could be used to produce CERs:

\[
\text{Effort} (PM) = A \cdot (KESLOC)^B 
\]

\[
\text{Effort} (PM) = C + A \cdot (KESLOC)^B 
\]

Each equation has a primary input parameter, KESLOC, with additional factors. The multiplicative factor, \( A \), roughly represents a production cost in Person-Months per thousands of lines of code. The \( C \) factor represents the fixed start-up and overhead activity costs expressed in Person-Months.

Both models have an exponent \( B \) on the size parameter, which is often referred to as a scaling factor. The impact of \( B \) is greater when size is larger and reduced when size is smaller. A common issue in modeling software engineering cost data is whether \( B \) is greater than or less than 1.0. If \( B \) is greater than one, larger size systems require more effort per line of code (diseconomies of scale). If \( B \) is less than one, larger sizes require less effort per line of code developed (economies of scale).
Banker and Kemerer [4] provide a survey of reasons for economies and diseconomies of scale. Their research substantiates both economies and diseconomies of scale. The economies of scale were observed on small projects and diseconomies of scale were observed on large projects. Economies of scale are attributed to:

- Software development tools that increase productivity
- Specialized personnel that are highly productive
- Fixed overhead that does not increase directly with project size thereby producing economies of scale in larger projects.

Diseconomies of scale are due to:

- Increased coordination effort required
- Larger systems having more complex interface problems
- Increased effort to obtain agreement or consensus
- Increase in overhead activities at a faster than linear rate as product size increases.

Equation 5.2 has the advantage of removing fixed start-up costs from the prediction portion of the equation. During SRDR analysis it was observed that fixed costs can mask the diseconomy of scale relationship between size and effort, especially with smaller size projects (less than 50 KESLOC). When Equation 5.2 was used as the CER model form, the diseconomy of scale was more often observed.

Equation 5.2 also has the advantage of a user being able to assess the reasonableness of the C factor with respect to known fixed costs in their organization and using the other part of the equation for the variable costs. Alternatively, a user could use the variable cost part and add it to their own estimate of project fixed start up costs. Both parts of the equation have a distinct interpretation and can be assessed for their reasonableness. However, there are difficulties using the model form in Equation 5.2.

Both Equations 5.1 and 5.2 are non-linear due to the scaling factor. If the equations were linear (e.g., $PM = A + B \cdot KESLOC$), OLS can be used to find the values for the model factors. This technique finds the values for the factors by minimizing the square of the errors between the actual and predicted effort, PM.

The multiplicative model in Equation 5.1 can be transformed from its non-linear form into a linear form using logarithms shown in Equation 5.3. The CERs presented in this chapter use this log transformed linear model.

$$\log(PM) = \log(A) + B \cdot \log(KESLOC).$$  \hspace{1cm} (5.3)

OLS is then used to find the estimated values for the A and B factors using the SRDR data. The resulting value for the A factor is actually $\log(A)$. It is converted back to unit-number space by raising to the log base (10 or $e$ if using natural logarithms) to the log value A. B is used as the scale factor without conversion.
Despite all of its advantages, 5.2 cannot be transformed into a linear model due to the position of the C factor in the model form. To solve for the three factors in 5.2, a non-regression iterative search technique must be used. Depending on the initial starting conditions, different values can result from the search. It is recommended to use multiple starting conditions to perform multiple searches and to select the factor values that result most often. This requires a specialized software analysis tool to conduct such a search.

Because of the ease of finding the factor values in Equation 5.1 and the difficulty of finding the factor values in 5.2, 5.1 is chosen as the CER model form presented in this chapter.

5.3.4 CER Calibration

The A and B factors in the CER model shown in Equation 5.3 are derived from SRDR data in each of the application types and operating environments. A and B are observable in scatter plots like those shown in Figures 5.3 and 5.4. The A is the intercept of the trend line through the data if it were extended until it crossed the vertical axis. The B is the slope of the trend line representing the average change in the response variable (PM in this analysis) for one unit of change in the predictor variable (KESLOC), i.e., the average change in effort for one unit of change in size.

OLS calculates the CER factors (A and B) that minimizes the distance between the fitted line and all of the data points, i.e. minimizes the error. It does this by mathematically minimizing the sum of the squared residual errors. In general, a model fits the data well if the differences between the observed values and the model’s predicted values (the residual errors) are small and unbiased. Large outlier data perturbs model results by producing larger differences.

Statistical software packages have linear regression tools that will analyze data and produce estimates for the A and B factors. Microsoft Excel also has a regression tool in the Data Analysis Tool Pack that will also produce these estimates.

In addition to estimates of the A and B factors, regression tools also produce statistics on the significance and variability of the estimated factors. Some of these important statistics are presented next with criteria used to evaluate the results for each CER model.

5.3.5 CER Evaluation

OLS regression with one predictor and one response variable provides estimates of factor values based on minimizing error. Therefore every CER has some amount of error. CERs are evaluated with a number of different statistics described in Table 5.3.

The calibration datasets are described by N, Min and Max KESLOC. Groups of data had to have 10 or more data points to be included in the
analysis presented here. The min and max size also set the boundaries for where the CER can safely be used.

The multiplicative factors $A$ and $B$ are the coefficients derived from the OLS regression. They are evaluated with the T-Statistic (T-Stat) and P-value measures for hypothesis testing. Generally a T-stat greater than 2.0 indicates a significant value and one that can be used in the CER model. A predictor that has a low P-value is likely to be a meaningful addition to the CER because changes in the predictor’s value are related to changes in the response variable. Conversely, a larger (insignificant) P-value suggests that changes in the predictor are not associated with changes in the response.

The Lower Confidence Interval (LCI) and Upper Confidence Interval (UCI) specify ranges for the true values of $A$ and $B$. This is because OLS regression provides estimates of factor values based on minimizing error. In practice the confidence interval is set to a desired level of confidence. For instance, we can be 95% confident that the range of the 95% confidence interval includes the real population mean of the effort to develop software of a given size. As the sample size increases, the sampling error decreases and the intervals become narrower. If one could increase the sample size to equal the population, there would be no sampling error. The confidence interval would have a width of zero and be equal to the true population parameter.

The Coefficient of Determination ($R^2$) is a primary measure that describes how much of the variation is explained by the CER regression model. The more variance that is accounted for by the regression model the closer the data points will fall to the fitted regression line. However, $R^2$ cannot determine whether the factor estimates and predictions are biased. Bias is detected in a scatter plot (called residual plots) of estimated versus actual response values. Looking at the scatter plot of residuals is necessary to evaluate the CER model factors.

The Standard Error (SE) is the standard deviation around the regression line. It is used to assess the precision of the predictions. $R^2$ is relevant mainly for assessing the fit of the trendline to the data. However, $R^2$ cannot be used to assess precision which SE is used for.

<table>
<thead>
<tr>
<th>Table 5.3: CER Evaluation Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statistic</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>Min and Max</td>
</tr>
<tr>
<td>KESLOC</td>
</tr>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

(continued)
TABLE 5.3: CER Evaluation Statistics (continued)

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>Scaling factor derived from OLS regression.</td>
</tr>
<tr>
<td>T-Stat</td>
<td>The T-Statistic (T-Stat) indicates the significance of the regression derived factors (multiplicative and scaling). It tells how many standard deviations the factors are from zero.</td>
</tr>
<tr>
<td>P-Value</td>
<td>The P-value for each factor tests the null hypothesis that the factor is equal to zero and has no effect. A low p-value less than $\alpha$ (typically $&lt; 0.05$) indicates that one can reject the null hypothesis. The level of significance $\alpha$ is the probability of achieving data at least as extreme.</td>
</tr>
<tr>
<td>LCI and UCI</td>
<td>Lower Confidence Interval (LCI) and Upper Confidence Interval (UCI) specify a range of values for the CER factors (A and B) that are likely to contain the true value of the unknown population parameter (the real A and B) at a given confidence level.</td>
</tr>
<tr>
<td>$R^2$</td>
<td>The Coefficient of Determination ($R^2$) represents the percentage of total variation explained by the regression model. It provides a measure of how well actual values of effort are estimated by the CER model factors A and B. $R^2$ ranges between 0 and 100% and is the same in log-space as in unit-space.</td>
</tr>
<tr>
<td>SE</td>
<td>Standard Error (SE) represents the average distance that the observed values fall from the regression line (estimated values). It indicates how wrong the regression model is on average using the units of the response variable, effort. Smaller values indicate that the observations are closer to the fitted line.</td>
</tr>
</tbody>
</table>

(continued)
Scatter plots are also used in CER assessment. Two plots are useful, a plot of residuals discussed in Section 5.3 and a plot of KESLOC versus PM with the CER model trend line displayed through the data scatter. Residual plots show the scatter of errors and should be checked before evaluating statistical measures for goodness-of-fit. They can reveal unwanted residual patterns that indicate biased results more effectively than numbers. When the residual plots show a random distribution of errors, the goodness-of-fit statistics are valid.

Both CER evaluation statistics and evaluation scatter plots are provided for each SRDR CER presented in this chapter.

### 5.3.6 CER Example

Creating CERs starts with segmenting the data into similar productivity groups. Scatter plots are used as a means of visually determining if a relationship exists between a predictor, X, and response variable, Y. The data is transformed into log-number space to make it normally distributed and homoscedastic. Factors for each CER parameter are estimated using simple linear regression. Finally the regression results are evaluated with statistics and scatter plots. This section presents an example that will be followed for each SRDR CER presented later.

The CER for the Real-Time Embedded (RTE) application type is presented in Equation 5.4. The effort (PM) and size (KESLOC) were both transformed from unit-number space into log-number space by taking natural logs (log) of all values. A regression tool was given the response values (log PM) and the corresponding predictor values (log KESLOC) for the RTE data. The resulting CER to predict effort in Person-Months (PM) for the RTE Application Type is:

\[
PM = 13.16 \cdot KESLOC^{0.84}
\]  

(5.4)

Because the RTE data is only a sample of the RTE population of data, the factors, 13.16 and 0.84, are estimates of the true population parameters and each has an associated error. Tables 5.4 and 5.5 show the CER evaluation statistics.

Table 5.4 shows that the number of data points was 57. The minimum size in the data was 1.8 KESLOC and the maximum was 200.6 KESLOC. The
estimated factors are $13.16$ for A and $0.84$ for B. Each factor has a very high T-Stat (good) and an extremely low P-Value (good). The confidence interval for A is $\pm 3.9$ and for B is $\pm 0.11$. The R-squared is $83\%$ which is judged to be strong. The SE is $211$ PM of effort which isn’t so good. PRED(30) is $56.4\%$ which is isn’t good.

**TABLE 5.4: RTE CER Regression Statistics**

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>57</td>
</tr>
<tr>
<td>Min KESLOC</td>
<td>1.8</td>
</tr>
<tr>
<td>Max KESLOC</td>
<td>200.6</td>
</tr>
<tr>
<td>A</td>
<td>13.16</td>
</tr>
<tr>
<td>B</td>
<td>0.84</td>
</tr>
<tr>
<td>$R^2$</td>
<td>83%</td>
</tr>
<tr>
<td>SE (PM)</td>
<td>211</td>
</tr>
<tr>
<td>PRED(30)</td>
<td>54%</td>
</tr>
</tbody>
</table>

**TABLE 5.5: RTE CER Coefficient Statistics**

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>13.16</td>
<td>14.7</td>
<td>1.00E-20</td>
<td>9.26</td>
<td>18.69</td>
</tr>
<tr>
<td>B</td>
<td>0.84</td>
<td>16.2</td>
<td>1.48E-22</td>
<td>0.73</td>
<td>0.94</td>
</tr>
</tbody>
</table>

Figure 5.8 shows the RTE CER Residuals scatter plot which is checked for any appearance of patterns. The scatter of errors appears to be randomly distributed, which indicates a good regression fit.
Figure 5.9 shows the RTE CER scatter plot with ± 1 Standard Error in log space, the Upper and Lower Prediction Intervals, and CER error distribution for a value of 10 KESLOC. The solid trendline for the CER is enclosed by the Standard Error and 95% prediction intervals representing estimation error. The ± 1 Standard Error around the line shows the average error around the mean for the regression, and the CER error distribution is a normal distribution with a mean of zero across the prediction interval.

A prediction interval represents the range where the response value is likely to fall given a single new predictor value. Prediction intervals account for the variability around the mean response inherent in any prediction. Like a confidence interval, prediction interval is set by the level of confidence that a new single observation will predict the correct response.

The prediction intervals contain 95% of the actual result values for the log of PM (effort). For instance, we can be 95% confident that this range includes the effort to develop software given a size. Unlike confidence intervals, prediction intervals predict the spread for response values given a predictor value rather than the mean response value. The prediction interval is always wider than the corresponding confidence interval of the prediction because of the added uncertainty involved in predicting a single response versus the mean response.

Using the CER formula in Equation 5.4 for an estimated 10 KESLOC CSCI, the estimated effort is 91 Person Months with the normally distributed
CER error spread shown in Figure 5.9. The error range covers approximately 30-250 Person-Months.

5.4 SRDR-Derived CERs

5.4.1 Model Coverage

The coverage of model-based CERs by operating environment and application type are in Table 5.6 showing the number of project data points. The operating environment acronyms in the columns are defined in Table 5.1. The application types in rows are defined in Table 5.2. Not all application types and environments were covered due to lack of enough data in each group (at least 10 data points are required). The populated cells denote a CER and the number in a cell is the number of data points used to create the CER. The Total column and row are for all operating environments or application types.

<table>
<thead>
<tr>
<th>AppType</th>
<th>AV</th>
<th>GF</th>
<th>GV</th>
<th>MV</th>
<th>OV</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>C&amp;C</td>
<td>9</td>
<td>16</td>
<td>3</td>
<td>5</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>CAS</td>
<td>1</td>
<td>13</td>
<td>2</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>1</td>
<td>22</td>
<td>22</td>
<td>47</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MP</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>RTE</td>
<td>23</td>
<td>21</td>
<td>5</td>
<td>5</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>S&amp;S</td>
<td>6</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>SCP</td>
<td>11</td>
<td>14</td>
<td>1</td>
<td>3</td>
<td>36</td>
<td></td>
</tr>
<tr>
<td>SYS</td>
<td>8</td>
<td>13</td>
<td>3</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TMDE</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td>10</td>
<td>14</td>
<td>3</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>10</td>
<td>1</td>
<td>5</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>87</td>
<td>134</td>
<td>26</td>
<td>42</td>
<td>25</td>
<td>306</td>
</tr>
</tbody>
</table>
5.4.2 Operating Environment CERs

The CERs presented in this and following sections follow the example in Section 5.3.6 and show the statistical criteria discussed in Section 5.3.5. Each CER is presented on its own page. The Operating Environment CERs in this section are derived from the number of data points shown in the Total row at the bottom of Table 5.6. The CER results for Operating Environment are summarized in Table 5.7.

TABLE 5.7: CERs by Operating Environment

<table>
<thead>
<tr>
<th>Op</th>
<th>Env</th>
<th>Effort</th>
<th>Equation</th>
<th>N</th>
<th>$R^2$</th>
<th>SE</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>PM = 9.41 $ \cdot \text{KESLOC}^{0.93}$</td>
<td>87</td>
<td>77%</td>
<td>516</td>
<td>39</td>
<td>0.7</td>
<td>329.9</td>
<td></td>
</tr>
<tr>
<td>GS</td>
<td>PM = 12.6 $ \cdot \text{KESLOC}^{0.79}$</td>
<td>134</td>
<td>74%</td>
<td>478</td>
<td>56</td>
<td>0.8</td>
<td>842.1</td>
<td></td>
</tr>
<tr>
<td>GV</td>
<td>PM = 15.1 $ \cdot \text{KESLOC}^{0.82}$</td>
<td>26</td>
<td>74%</td>
<td>178</td>
<td>54</td>
<td>7.2</td>
<td>283.1</td>
<td></td>
</tr>
<tr>
<td>MV</td>
<td>PM = 5.44 $ \cdot \text{KESLOC}^{1.12}$</td>
<td>42</td>
<td>87%</td>
<td>206</td>
<td>33</td>
<td>0.4</td>
<td>123.8</td>
<td></td>
</tr>
<tr>
<td>OV</td>
<td>PM = 27.45 $ \cdot \text{KESLOC}^{0.71}$</td>
<td>25</td>
<td>79%</td>
<td>11</td>
<td>48</td>
<td>0.9</td>
<td>221.1</td>
<td></td>
</tr>
</tbody>
</table>
5.4.2.1 Aerial Vehicle (AV)

\[ PM = 9.41 \cdot KESLOC^{0.93} \]

Data Points = 87
KESLOC Min = 0.7
KESLOC Max = 329.9

\[ R^2 = 77\% \]
SE = 516 PM
PRED = 39%

![Residuals plot](image)

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>9.41</td>
<td>12.3</td>
<td>1.54E-20</td>
<td>6.55</td>
<td>13.53</td>
</tr>
<tr>
<td>B</td>
<td>0.93</td>
<td>17.0</td>
<td>4.58E-29</td>
<td>0.82</td>
<td>1.03</td>
</tr>
</tbody>
</table>

![Regression line](image)

FIGURE 5.10: AV Dataset and CER Summary
5.4.2.2 Ground Site (GS)

\[ PM = 12.6 \cdot KESLOC^{0.79} \]

Data Points = 134
KESLOC Min = 0.8
KESLOC Max = 842.1

\[ R^2 = 74\% \]
SE = 478 PM
PRED = 56%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>12.6</td>
<td>15.84</td>
<td>2.51E-32</td>
<td>9.18</td>
<td>17.27</td>
</tr>
<tr>
<td>B</td>
<td>0.79</td>
<td>19.56</td>
<td>8.05E-41</td>
<td>.71</td>
<td>0.87</td>
</tr>
</tbody>
</table>

FIGURE 5.11: GS Dataset and CER Summary
5.4.2.3  Ground Vehicle (GV)

\[ PM = 15.1 \cdot KESLOC^{0.82} \]

Data Points = 26
KESLOC Min = 7.2
KESLOC Max = 283.1

\[ R^2 = 74\% \]
\[ SE = 178 \text{ PM} \]
\[ PRED = 54\% \]

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15.1</td>
<td>7.67</td>
<td>6.56E-08</td>
<td>7.3</td>
<td>31.5</td>
</tr>
<tr>
<td>B</td>
<td>0.82</td>
<td>8.18</td>
<td>2.09E-08</td>
<td>.61</td>
<td>1.02</td>
</tr>
</tbody>
</table>

FIGURE 5.12: GV Dataset and CER Summary
5.4.2.4 Maritime Vessel (MV)

\[ PM = 5.44 \cdot KESLOC^{1.12} \]

Data Points = 42
KESLOC Min = 0.4
KESLOC Max = 123.8

\[ R^2 = 87\% \]
SE = 206 PM
PRED = 33%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.44</td>
<td>9.73</td>
<td>4.28E-12</td>
<td>3.83</td>
<td>7.74</td>
</tr>
<tr>
<td>B</td>
<td>1.12</td>
<td>16.61</td>
<td>1.51E-19</td>
<td>.98</td>
<td>1.26</td>
</tr>
</tbody>
</table>

FIGURE 5.13: MV Dataset and CER Summary
5.4.2.5 Ordnance Vehicle (OV)

\[ PM = 27.45 \cdot KESLOC^{0.71} \]

Data Points = 25
KESLOC Min = 0.9
KESLOC Max = 221.1

\[ R^2 = 79\% \]
SE = 11 PM
PRED = 48%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>27.45</td>
<td>11.88</td>
<td>2.70E-11</td>
<td>15.42</td>
<td>48.88</td>
</tr>
<tr>
<td>B</td>
<td>0.71</td>
<td>9.35</td>
<td>2.67E-09</td>
<td>0.55</td>
<td>0.87</td>
</tr>
</tbody>
</table>

FIGURE 5.14: OV Dataset and CER Summary
5.4.3 Application Types CERs

The CERs in this section are derived from the number of data points in Table 5.6 listed as the AppType Total in the right column. The CER results across all environments are summarized in Table 5.8.

<table>
<thead>
<tr>
<th>App Type</th>
<th>Effort Equation</th>
<th>N</th>
<th>$R^2$</th>
<th>SE</th>
<th>PRED Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>$PM = 2.64 \cdot KESLOC^{1.02}$</td>
<td>16</td>
<td>97%</td>
<td>135</td>
<td>88</td>
<td>2.4</td>
</tr>
<tr>
<td>COM</td>
<td>$PM = 7.30 \cdot KESLOC^{0.91}$</td>
<td>47</td>
<td>88%</td>
<td>253</td>
<td>60</td>
<td>0.4</td>
</tr>
<tr>
<td>C&amp;C</td>
<td>$PM = 6.60 \cdot KESLOC^{1.05}$</td>
<td>33</td>
<td>88%</td>
<td>449</td>
<td>52</td>
<td>0.8</td>
</tr>
<tr>
<td>MP</td>
<td>$PM = 6.14 \cdot KESLOC^{0.86}$</td>
<td>20</td>
<td>77%</td>
<td>220</td>
<td>55</td>
<td>9.6</td>
</tr>
<tr>
<td>RTE</td>
<td>$PM = 13.20 \cdot KESLOC^{0.84}$</td>
<td>57</td>
<td>83%</td>
<td>211</td>
<td>54</td>
<td>1.8</td>
</tr>
<tr>
<td>SCP</td>
<td>$PM = 26.5 \cdot KESLOC^{0.87}$</td>
<td>36</td>
<td>91%</td>
<td>346</td>
<td>50</td>
<td>0.8</td>
</tr>
<tr>
<td>S&amp;S</td>
<td>$PM = 7.43 \cdot KESLOC^{0.91}$</td>
<td>17</td>
<td>85%</td>
<td>176</td>
<td>53</td>
<td>4.4</td>
</tr>
<tr>
<td>SYS</td>
<td>$PM = 5.06 \cdot KESLOC^{0.98}$</td>
<td>27</td>
<td>93%</td>
<td>585</td>
<td>48</td>
<td>0.8</td>
</tr>
<tr>
<td>TMDE</td>
<td>$PM = 7.42 \cdot KESLOC^{1.00}$</td>
<td>11</td>
<td>92%</td>
<td>454</td>
<td>27</td>
<td>0.5</td>
</tr>
<tr>
<td>VC</td>
<td>$PM = 9.05 \cdot KESLOC^{1.02}$</td>
<td>27</td>
<td>92%</td>
<td>303</td>
<td>48</td>
<td>0.7</td>
</tr>
<tr>
<td>VP</td>
<td>$PM = 22.27 \cdot KESLOC^{0.81}$</td>
<td>18</td>
<td>89%</td>
<td>111</td>
<td>56</td>
<td>1.1</td>
</tr>
</tbody>
</table>
5.4.3.1 Command and Control (C&C)

\[ PM = 6.60 \cdot KESLOC^{1.05} \]

Data Points = 33
KESLOC Min = 0.8
KESLOC Max = 229.0

\[ R^2 = 83\% \]
SE = 211 PM
PRED = 54%

FIGURE 5.15: CC Dataset and CER Summary
5.4.3.2 Custom AIS Software (CAS)

\[ PM = 2.64 \cdot KESLOC^{1.02} \]

Data Points = 16  
KESLOC Min = 2.4  
KESLOC Max = 417.1

\[ R^2 = 97\% \]

\[ SE = 135 \]

\[ PRED = 88\% \]

---

**Coefficient** | **Value** | **T-Stat** | **P-Value** | **LCI** | **UCI**
--- | --- | --- | --- | --- | ---
A | 2.64 | 5.45 | 8.51E-05 | 1.80 | 3.87
B | 1.02 | 23.1 | 1.47E-12 | 0.93 | 1.12

---

**FIGURE 5.16: CAS Dataset and CER Summary**
5.4.3.3 Communication (COM)

\[ PM = 7.30 \cdot KESLOC^{0.91} \]

Data Points = 47
KESLOC Min = 0.4
KESLOC Max = 532.4

\[ R^2 = 88\% \]
SE = 253 PM
PRED = 60%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7.3</td>
<td>5.45</td>
<td>2.37E-17</td>
<td>1.80</td>
<td>3.87</td>
</tr>
<tr>
<td>B</td>
<td>.91</td>
<td>23.1</td>
<td>3.15E-22</td>
<td>0.93</td>
<td>1.12</td>
</tr>
</tbody>
</table>

FIGURE 5.17: COM Dataset and CER Summary
5.4.3.4 Mission Planning (MP)

\[ PM = 6.14 \cdot KESLOC^{0.86} \]

Data Points = 20
KESLOC Min = 9.6
KESLOC Max = 570.0

\[ R^2 = 77\% \]
SE = 220 PM
PRED = 55%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6.14</td>
<td>3.68</td>
<td>0.0017</td>
<td>2.18</td>
<td>17.31</td>
</tr>
<tr>
<td>B</td>
<td>0.86</td>
<td>7.74</td>
<td>3.93E-07</td>
<td>0.63</td>
<td>1.09</td>
</tr>
</tbody>
</table>

**Figure 5.18**: MP Dataset and CER Summary
5.4.3.5 Real Time Embedded (RTE)

\[ PM = 13.20 \cdot KESLOC^{0.84} \]

Data Points = 57
KESLOC Min = 2
KESLOC Max = 201

\[ R^2 = 83\% \]
SE = 211 PM
PRED = 54%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>13.16</td>
<td>14.7</td>
<td>1.00E-20</td>
<td>9.26</td>
<td>18.69</td>
</tr>
<tr>
<td>B</td>
<td>0.84</td>
<td>16.2</td>
<td>1.48E-22</td>
<td>0.73</td>
<td>0.94</td>
</tr>
</tbody>
</table>

FIGURE 5.19: RTE Dataset and CER Summary
5.4.3.6 Scientific and Simulation (S&S)

\[ PM = 7.43 \cdot KESLOC^{0.91} \]

Data Points = 17
KESLOC Min = 4.4
KESLOC Max = 225.8

\[ R^2 = 85\% \]
SE = 176 PM
PRED = 53%

FIGURE 5.20: S&S Dataset and CER Summary
5.4.3.7 Sensor Control and Signal Processing (SCP)

\[ PM = 26.5 \cdot KESLOC^{0.87} \]

Data Points = 36
KESLOC Min = 0.8
KESLOC Max = 192.9

\[ R^2 = 91\% \]
SE = 346 PM
PRED = 50%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>26.5</td>
<td>21.7</td>
<td>1.72E-21</td>
<td>19.5</td>
<td>36.0</td>
</tr>
<tr>
<td>B</td>
<td>0.87</td>
<td>18.5</td>
<td>2.43E-19</td>
<td>0.77</td>
<td>0.96</td>
</tr>
</tbody>
</table>

FIGURE 5.21: SCP Dataset and CER Summary
5.4.3.8 System Software (SYS)

\[ PM = 5.06 \cdot KESLOC^{0.98} \]

- Data Points = 27
- KESLOC Min = 0.8
- KESLOC Max = 842.1

\[ R^2 = 93\% \]
\[ SE = 585 \text{ PM} \]
\[ PRED = 48\% \]

![Residuals Graph](Image)

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.1</td>
<td>7.1</td>
<td>1.55E-07</td>
<td>3.18</td>
<td>8.05</td>
</tr>
<tr>
<td>B</td>
<td>0.98</td>
<td>17.6</td>
<td>1.35E-15</td>
<td>0.86</td>
<td>1.10</td>
</tr>
</tbody>
</table>

![Person-Months vs. KESLOC Graph](Image)

FIGURE 5.22: SYS Dataset and CER Summary
5.4.3.9 Test, Measurement and Diagnostic Equipment (TMDE)

\[ PM = 7.42 \cdot KESLOC^{1.00} \]

Data Points = 11
KESLOC Min = 0.5
KESLOC Max = 313.0

\[ R^2 = 92\% \]
SE = 454 PM
PRED = 27%

![Graph of residuals](image)

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7.42</td>
<td>5.69</td>
<td>0.000297</td>
<td>3.34</td>
<td>16.5</td>
</tr>
<tr>
<td>B</td>
<td>1.00</td>
<td>10.11</td>
<td>3.26E-06</td>
<td>0.78</td>
<td>1.23</td>
</tr>
</tbody>
</table>

![Graph of KESLOC vs PM](image)

FIGURE 5.23: TMDE Dataset and CER Summary
5.4.3.10 Vehicle Control (VC)

\[ PM = 9.05 \cdot KESLOC^{1.02} \]

Data Points = 27
KESLOC Min = 0.7
KESLOC Max = 329.9

\[ R^2 = 92\% \]
\[ SE = 303 \text{ PM} \]
\[ PRED = 48\% \]

FIGURE 5.24: VC Dataset and CER Summary
5.4.3.11 Vehicle Payload (VP)

\[ PM = 22.27 \cdot KESLOC^{0.81} \]

Data Points = 18
KESLOC Min = 1.1
KESLOC Max = 221.1

\[ R^2 = 89\% \]
SE = 111 PM
PRED = 56%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>22.27</td>
<td>13.8</td>
<td>2.62E-10</td>
<td>13.8</td>
<td>35.9</td>
</tr>
<tr>
<td>B</td>
<td>0.81</td>
<td>11.7</td>
<td>3.10E-09</td>
<td>0.66</td>
<td>0.95</td>
</tr>
</tbody>
</table>

FIGURE 5.25: VP Dataset and CER Summary

5.4.4 Application Type and Operating Environment CERs

The CERs in this section are derived from the number of data points in Table 5.6 where the AppType and OpEnv intersect and the number of data points is at least 10. These CER results are summarized in Table 5.9.

TABLE 5.9: CERs by AppType and OpEnv

<table>
<thead>
<tr>
<th>Group</th>
<th>Effort Equation</th>
<th>N</th>
<th>$R^2$</th>
<th>SE</th>
<th>PRED Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerial Vehicle (AV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTE</td>
<td>$PM = 15.09 \cdot KESLOC^{0.81}$</td>
<td>23</td>
<td>82%</td>
<td>293</td>
<td>52</td>
<td>2.0</td>
</tr>
<tr>
<td>SCP</td>
<td>$PM = 28.26 \cdot KESLOC^{0.86}$</td>
<td>11</td>
<td>92%</td>
<td>528</td>
<td>55</td>
<td>1.1</td>
</tr>
<tr>
<td>VC</td>
<td>$PM = 8.09 \cdot KESLOC^{1.05}$</td>
<td>10</td>
<td>96%</td>
<td>328</td>
<td>50</td>
<td>0.7</td>
</tr>
<tr>
<td>VP</td>
<td>$PM = 13.38 \cdot KESLOC^{0.91}$</td>
<td>10</td>
<td>95%</td>
<td>85</td>
<td>70</td>
<td>1.1</td>
</tr>
<tr>
<td>Ground System (GS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC</td>
<td>$PM = 6.66 \cdot KESLOC^{1.01}$</td>
<td>16</td>
<td>91%</td>
<td>257</td>
<td>63</td>
<td>5.0</td>
</tr>
<tr>
<td>CAS</td>
<td>$PM = 2.57 \cdot KESLOC^{1.03}$</td>
<td>13</td>
<td>79%</td>
<td>146</td>
<td>48</td>
<td>2.4</td>
</tr>
<tr>
<td>COM</td>
<td>$PM = 10.80 \cdot KESLOC^{0.83}$</td>
<td>22</td>
<td>82%</td>
<td>435</td>
<td>36</td>
<td>1.2</td>
</tr>
<tr>
<td>MP</td>
<td>$PM = 6.14 \cdot KESLOC^{0.86}$</td>
<td>20</td>
<td>77%</td>
<td>221</td>
<td>55</td>
<td>9.6</td>
</tr>
<tr>
<td>RTE</td>
<td>$PM = 8.71 \cdot KESLOC^{0.92}$</td>
<td>21</td>
<td>83%</td>
<td>149</td>
<td>48</td>
<td>1.8</td>
</tr>
<tr>
<td>SCP</td>
<td>$PM = 31.88 \cdot KESLOC^{0.79}$</td>
<td>14</td>
<td>86%</td>
<td>346</td>
<td>43</td>
<td>0.8</td>
</tr>
<tr>
<td>SYS</td>
<td>$PM = 7.93 \cdot KESLOC^{0.89}$</td>
<td>13</td>
<td>85%</td>
<td>941</td>
<td>31</td>
<td>10.7</td>
</tr>
<tr>
<td>Ground Vehicle (GV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td>$PM = 7.81 \cdot KESLOC^{1.10}$</td>
<td>14</td>
<td>78%</td>
<td>149</td>
<td>43</td>
<td>10.6</td>
</tr>
<tr>
<td>Maritime Vessel (MV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>$PM = 5.77 \cdot KESLOC^{0.96}$</td>
<td>22</td>
<td>90%</td>
<td>17</td>
<td>68</td>
<td>0.4</td>
</tr>
</tbody>
</table>
5.4.4.1 Command & Control - Ground Site

\[ PM = 6.66 \cdot KESLOC^{1.01} \]

Data Points = 16
KESLOC Min = 5.0
KESLOC Max = 229.0

\[ R^2 = 91\% \]
SE = 257 PM
PRED = 63%

\[ \begin{array}{cccccc}
\text{Coefficient} & \text{Value} & \text{T-Stat} & \text{P-Value} & \text{LCI} & \text{UCI} \\
A & 6.66 & 5.73 & 5.24E-05 & 3.27 & 13.56 \\
B & 1.01 & 12.01 & 9.28E-09 & 0.83 & 1.19 \\
\end{array} \]

FIGURE 5.26: CCGS Dataset and CER Summary
5.4.4.2 Custom AIS Software - Ground Site

\[ PM = 2.57 \cdot KESLOC^{1.03} \]

Data Points = 13
KESLOC Min = 2.4
KESLOC Max = 417.1

\[ R^2 = 79\% \]
SE = 146 PM
PRED = 48%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2.57</td>
<td>4.66</td>
<td>6.89E-04</td>
<td>1.64</td>
<td>4.00</td>
</tr>
<tr>
<td>B</td>
<td>1.03</td>
<td>20.70</td>
<td>3.69E-10</td>
<td>0.92</td>
<td>1.13</td>
</tr>
</tbody>
</table>

FIGURE 5.27: CASGS Dataset and CER Summary
5.4.4.3 Communications - Ground Site

\[ PM = 10.80 \cdot KESLOC^{0.83} \]

Data Points = 22
KESLOC Min = 1.2
KESLOC Max = 532.4

\[ R^2 = 82\% \]
SE = 435 PM
PRED = 36%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10.80</td>
<td>7.75</td>
<td>1.90E-07</td>
<td>5.69</td>
<td>20.49</td>
</tr>
<tr>
<td>B</td>
<td>0.83</td>
<td>9.50</td>
<td>7.43E-09</td>
<td>0.65</td>
<td>1.01</td>
</tr>
</tbody>
</table>

FIGURE 5.28: COMGS Dataset and CER Summary
5.4.4.4 Communications - Maritime Vessel

\[ PM = 5.77 \cdot KESLOC^{0.96} \]

Data Points = 22
KESLOC Min = 0.4
KESLOC Max = 27.1

\[ R^2 = 90\% \]
SE = 17 PM
PRED = 68%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>5.77</td>
<td>12.57</td>
<td>5.94E-11</td>
<td>4.31</td>
<td>7.71</td>
</tr>
<tr>
<td>B</td>
<td>0.96</td>
<td>13.40</td>
<td>1.88E-11</td>
<td>0.81</td>
<td>1.11</td>
</tr>
</tbody>
</table>

![Residuals](image)

**Figure 5.29: COMMV Dataset and CER Summary**
5.4.4.5 Mission Planning - Ground Site

\[ PM = 6.14 \cdot KESLOC^{0.86} \]

Data Points = 20
KESLOC Min = 9.6
KESLOC Max = 570.0

\[ R^2 = 77\% \]
SE = 221 PM
PRED = 55%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6.14</td>
<td>3.68</td>
<td>0.0017</td>
<td>2.18</td>
<td>17.31</td>
</tr>
<tr>
<td>B</td>
<td>0.86</td>
<td>7.74</td>
<td>3.93E-07</td>
<td>0.63</td>
<td>1.09</td>
</tr>
</tbody>
</table>

\[ PM = 6.14 \cdot KESLOC^{0.86} \]

\[ \pm \text{Standard Error} \]

FIGURE 5.30: MPGS Dataset and CER Summary
5.4.4.6 Real Time Embedded - Aerial Vehicle

\[ PM = 15.09 \cdot KESLOC^{0.81} \]

Data Points = 23
KESLOC Min = 2.0
KESLOC Max = 164.2

\[ R^2 = 82\% \]
SE = 293 PM
PRED = 52%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>15.09</td>
<td>10.28</td>
<td>1.20E-09</td>
<td>8.71</td>
<td>26.14</td>
</tr>
<tr>
<td>B</td>
<td>0.81</td>
<td>9.95</td>
<td>2.13E-09</td>
<td>0.64</td>
<td>0.98</td>
</tr>
</tbody>
</table>

FIGURE 5.31: RTEAV Dataset and CER Summary
5.4.4.7 Real Time Embedded - Ground Site

\[ PM = 8.71 \cdot KESLOC^{0.92} \]

Data Points = 21
KESLOC Min = 1.8
KESLOC Max = 112.7

\[ R^2 = 83\% \]
SE = 149 PM
PRED = 48%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8.71</td>
<td>6.85</td>
<td>1.56E-06</td>
<td>4.49</td>
<td>16.88</td>
</tr>
<tr>
<td>B</td>
<td>0.92</td>
<td>9.63</td>
<td>9.60E-09</td>
<td>0.72</td>
<td>1.12</td>
</tr>
</tbody>
</table>

FIGURE 5.32: RTEGS Dataset and CER Summary
5.4.4.8 Sensor Control and Signal Processing - Ground Vehicle

\[ PM = 28.26 \cdot KESLOC^{0.86} \]

Data Points = 11
KESLOC Min = 1.1
KESLOC Max = 178.8

\[ R^2 = 92\% \]
SE = 528 PM
PRED = 55%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>28.26</td>
<td>10.87</td>
<td>1.78E-06</td>
<td>14.10</td>
<td>56.66</td>
</tr>
<tr>
<td>B</td>
<td>0.86</td>
<td>9.92</td>
<td>3.81E-06</td>
<td>0.67</td>
<td>1.06</td>
</tr>
</tbody>
</table>

FIGURE 5.33: SCPAV Dataset and CER Summary
5.4.4.9 Sensor Control and Signal Processing - Ground Site

\[ PM = 31.88 \cdot KESLOC^{0.79} \]

Data Points = 14
KESLOC Min = 0.8
KESLOC Max = 193.0

\[ R^2 = 86\% \]
SE = 346 PM
PRED = 43%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>31.88</td>
<td>12.02</td>
<td>4.74E-08</td>
<td>17.02</td>
<td>59.70</td>
</tr>
<tr>
<td>B</td>
<td>0.79</td>
<td>8.64</td>
<td>1.69E-06</td>
<td>0.59</td>
<td>0.99</td>
</tr>
</tbody>
</table>

FIGURE 5.34: SCPGS Dataset and CER Summary
5.4.4.10 System Software - Ground Site

\[ PM = 7.93 \cdot KESLOC^{0.89} \]

Data Points = 13
KESLOC Min = 10.7
KESLOC Max = 842.1

\[ R^2 = 85\% \]
SE = 941 PM
PRED = 31%

\begin{tabular}{lcccc}
Coefficient & Value & T-Stat & P-Value & LCI & UCI \\
\hline
A & 7.93 & 3.78 & 0.0030 & 2.38 & 26.46 \\
B & 0.89 & 7.94 & 7.01E-06 & 0.64 & 1.14 \\
\end{tabular}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{sygs_dataset_cer_summary}
\caption{SYSGS Dataset and CER Summary}
\end{figure}
5.4.4.11 Vehicle Control - Aerial Vehicle

\[ PM = 8.09 \cdot KESLOC^{1.05} \]

Data Points = 10
KESLOC Min = 0.7
KESLOC Max = 330.0

\[ R^2 = 96\% \]
SE = 328 PM
PRED = 50%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>8.09</td>
<td>8.80</td>
<td>2.18E-05</td>
<td>4.68</td>
<td>13.99</td>
</tr>
<tr>
<td>B</td>
<td>1.05</td>
<td>13.71</td>
<td>7.71E-07</td>
<td>0.87</td>
<td>1.22</td>
</tr>
</tbody>
</table>

\[ PM = 8.09 \cdot KESLOC^{1.05} \]

\[ \pm \text{Standard Error} \]

FIGURE 5.36: VCAV Dataset and CER Summary
5.4.4.12 Vehicle Control - Ground Vehicle

\[ PM = 7.81 \cdot KESLOC^{1.10} \]

Data Points = 14
KESLOC Min = 10.6
KESLOC Max = 74.7

\[ R^2 = 78\% \]
SE = 149 PM
PRED = 43%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>7.81</td>
<td>3.72</td>
<td>0.0029</td>
<td>2.35</td>
<td>26.00</td>
</tr>
<tr>
<td>B</td>
<td>1.10</td>
<td>6.58</td>
<td>2.60E-05</td>
<td>0.73</td>
<td>1.46</td>
</tr>
</tbody>
</table>

\[ PM = 7.81 \cdot KESLOC^{1.10} \]
- Standard Error

FIGURE 5.37: VCGV Dataset and CER Summary
5.4.4.13 Vehicle Payload - Aerial Vehicle

\[ PM = 13.38 \cdot KESLOC^{0.91} \]

Data Points = 10
KESLOC Min = 1.1
KESLOC Max = 86.7

\[ R^2 = 95\% \]
SE = 85 PM
PRED = 70%

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Value</th>
<th>T-Stat</th>
<th>P-Value</th>
<th>LCI</th>
<th>UCI</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>13.38</td>
<td>12.31</td>
<td>1.76E-06</td>
<td>8.23</td>
<td>21.74</td>
</tr>
<tr>
<td>B</td>
<td>0.91</td>
<td>12.69</td>
<td>1.40E-06</td>
<td>0.74</td>
<td>1.07</td>
</tr>
</tbody>
</table>

FIGURE 5.38: VPAV Dataset and CER Summary
### 5.4.5 CER Effort Distribution

Using a CER produces an effort estimate given the size of software development. The estimate is for all activities covered by the data on which the CER was derived (see Section 4.5.2 on standardizing effort). The total effort estimate can be distributed across different development activities.

Average effort percentages for each activity consistently covered by SRDR data were derived for each Application Type with 10 or more data points. The total estimated effort is distributed across development activities using the percentages in Table 5.10.

<table>
<thead>
<tr>
<th>Application Type</th>
<th>Requirement Analysis</th>
<th>Architecture &amp; Design</th>
<th>Code &amp; Unit Test</th>
<th>Integration &amp; QT</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAS</td>
<td>11.6%</td>
<td>27.8%</td>
<td>35.6%</td>
<td>25.0%</td>
</tr>
<tr>
<td>C&amp;C</td>
<td>20.6%</td>
<td>15.8%</td>
<td>28.9%</td>
<td>34.8%</td>
</tr>
<tr>
<td>MP</td>
<td>16.2%</td>
<td>12.3%</td>
<td>50.8%</td>
<td>20.7%</td>
</tr>
<tr>
<td>RTE</td>
<td>15.5%</td>
<td>26.7%</td>
<td>26.7%</td>
<td>31.2%</td>
</tr>
<tr>
<td>S&amp;S</td>
<td>7.4%</td>
<td>39.9%</td>
<td>32.1%</td>
<td>20.7%</td>
</tr>
<tr>
<td>SCP</td>
<td>10.8%</td>
<td>45.2%</td>
<td>20.3%</td>
<td>23.7%</td>
</tr>
<tr>
<td>SYS</td>
<td>17.6%</td>
<td>21.1%</td>
<td>28.8%</td>
<td>32.5%</td>
</tr>
<tr>
<td>VC</td>
<td>18.5%</td>
<td>23.6%</td>
<td>31.3%</td>
<td>26.6%</td>
</tr>
</tbody>
</table>

### 5.5 SRDR-Derived Benchmarks

#### 5.5.1 Benchmark Model and Evaluation

The purpose of software benchmarks are for comparisons between groups of software, and not estimation. The benchmarks provide a sense of which groups are more or less expensive to produce. The benchmarks presented here are expressed as productivity numbers.

Software productivity is the ratio of outputs to inputs per Equation 5.5. The size, expressed as equivalent SLOC, represents the amount of functionality implemented. The effort represents the amount of labor required to implement the functionality as influenced by the inputs discussed above.

\[
\text{Productivity} = \frac{\text{Outputs}}{\text{ Inputs}} = \frac{\text{ESLOC}}{\text{PM}}
\]  

(5.5)

The metric used to express software productivity is Equivalent Source Lines of Code (ESLOC) per Person-Month (PM) of effort. While many other measures exist, ESLOC / PM will be used because most of the data collected
by the DoD on past projects is captured using these two measures. While controversy exists over whether or not ESLOC / PM is a good measure, consistent use of this metric (see Metric Definitions) provides for meaningful comparisons of productivity.

Table 5.6 shows the coverage by the number of data points analyzed by Operating Environment and Application Type. A minimum of 10 or more projects were required to derive a productivity benchmark. Groups with smaller sample sizes weren’t considered large enough to draw conclusions from. The productivity results contain the statistics defined in Table 5.11.

### 5.5.2 Operating Environment Benchmarks

Table 5.12 shows the productivity statistics for the Operating Environments (OpEnv). They are sorted alphabetically with the measures per Table 5.11 for assessment.

For example, the Aerial Vehicle (AV) Operating Environment in Table 5.12 has 80 data points for analysis. The mean productivity was 137 ESLOC/PM. The lower 95% confidence interval is 119 ESLOC/PM and the upper is 156 ESLOC/PM. This means that with 95% confidence, the true population mean for Aerial Vehicle productivity is between 119 and 156 ESLOC/PM. The SE, standard error, is 9.4 ESLOC/PM and is used to calculate the confidence intervals. The Standard Deviation (SD) shows the range where 68% of the data points lie above or below the mean, 54 - 222 ESLOC/PM. Smaller Standard Deviations are desirable to draw benchmarking conclusions.

The median productivity is 118 ESLOC/PM. This is less than the mean productivity and indicates a non-normal data distribution. In situations where the data is not normally distributed, the median productivity is a more reliable value. The first and third quartiles, Q1 and Q3, show the range where 50% of the data points lie above and below the median, 78 to 172 ESLOC/PM. Narrower ranges are desirable.

A conclusion that may be made for the Aerial Vehicle environment is the mean or median productivity is based on a wide range of individual data point productivities. The large spread of productivities indicates the mean and median productivity are not highly reliable to meaningfully benchmark this environment.

The benchmark values in Table 5.12 are further visualized in box and whisker plots in Figure 5.39. The boxes are bounded by the 1st and 3rd quartiles, Q1 and Q3, with the median shown as a line in the box. The tails represent values beyond the 1st and 3rd quartiles. The lower whisker on each is the smallest data value which is larger than the lower quartile. The upper whisker is the largest data value which is smaller than the upper quartile. Data outliers beyond these ranges are indicated with marks.

The widths of the boxes in Figure 5.39 are also proportional to the sample sizes for the groups. Operating Environments with wider boxes had more data used which is desirable.
### TABLE 5.11: Productivity Statistic Definitions

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Number of data points</td>
</tr>
<tr>
<td>Min ESLOC</td>
<td>Minimum size in thousands of equivalent source lines of code</td>
</tr>
<tr>
<td>Max ESLOC</td>
<td>Maximum size in thousands of equivalent source lines of code</td>
</tr>
<tr>
<td>LCI</td>
<td>Lower Confidence Interval is an estimate of an interval below the sample mean within which the population mean is estimated to lie</td>
</tr>
<tr>
<td>Mean</td>
<td>Estimated sample value representing the population central value; equal to the sum of the values divided by the number of values, i.e., arithmetic mean</td>
</tr>
<tr>
<td>UCI</td>
<td>Upper Confidence Interval is an estimate of an interval above the sample mean within which the population mean is estimated to lie</td>
</tr>
<tr>
<td>SE</td>
<td>Standard Error of the mean is an estimate computed from the sample of data being analyzed of the standard deviation of sample means over all possible samples (of a given size) drawn from the population. It is found by dividing the StdDev by the square root of N.</td>
</tr>
<tr>
<td>StdDev</td>
<td>Standard Deviation measures the amount of variation or dispersion from the mean in a sample. Plus or minus one standard deviation from the mean is a range that includes about 68% of the data.</td>
</tr>
<tr>
<td>Q1</td>
<td>Numerical value for the lower 25% of ranked data (1st Quartile), i.e., the value half way between the lowest value and the median in a set of ranked values</td>
</tr>
<tr>
<td>Median</td>
<td>Numerical value separating the higher half of a sample from the lower half, i.e., the middle value in a set of ranked values</td>
</tr>
<tr>
<td>Q3</td>
<td>Numerical value for the lower 75% of ranked data (3rd Quartile), i.e. the value half way between the median and the highest value in a set of ranked values</td>
</tr>
</tbody>
</table>
TABLE 5.12: Productivity Benchmarks by Operating Environment

<table>
<thead>
<tr>
<th>OpEnv</th>
<th>N</th>
<th>LCI</th>
<th>Mean</th>
<th>UCI</th>
<th>SE</th>
<th>SD</th>
<th>Q1</th>
<th>Mdn.</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>AV</td>
<td>80</td>
<td>118.9</td>
<td>137.6</td>
<td>156.3</td>
<td>9.4</td>
<td>84.1</td>
<td>77.7</td>
<td>117.6</td>
<td>172.0</td>
</tr>
<tr>
<td>GF</td>
<td>133</td>
<td>186.5</td>
<td>208.2</td>
<td>230.0</td>
<td>11.0</td>
<td>126.8</td>
<td>118.2</td>
<td>192.0</td>
<td>280.0</td>
</tr>
<tr>
<td>GV</td>
<td>26</td>
<td>110.2</td>
<td>140.6</td>
<td>171.0</td>
<td>14.8</td>
<td>75.3</td>
<td>89.9</td>
<td>121.0</td>
<td>169.5</td>
</tr>
<tr>
<td>MV</td>
<td>42</td>
<td>138.9</td>
<td>163.0</td>
<td>187.2</td>
<td>12.0</td>
<td>77.5</td>
<td>88.1</td>
<td>178.8</td>
<td>217.3</td>
</tr>
<tr>
<td>OV</td>
<td>25</td>
<td>82.8</td>
<td>122.5</td>
<td>162.1</td>
<td>19.2</td>
<td>96.2</td>
<td>59.1</td>
<td>95.4</td>
<td>161.4</td>
</tr>
</tbody>
</table>

Figure 5.39 shows that while the operating environments are physically distinct, the productivities are indistinguishable. By this, productivity is not influenced by the operating environment. There are two caveats to this conclusion:

1. There is the lack of data for the space vehicle, SV, environment. Initial indications are the SV environment has the lowest productivity.
2. There was not enough data to separate each environment into manned and unmanned sub-environments. Some cost estimators feel the manned environment has a lower productivity that an unmanned one.

5.5.3 Application Type Benchmarks

Table 5.13 shows the mean and median productivity across Application Types (AppType). To be included in the table, there had to be 10 or more data points in an application type group. The rows are sorted alphabetically.

Figure 5.40 shows a boxplot comparison of the productivities across application types summarized in Table 5.13. The box widths are proportional to the sample size for each group. Plot interpretation is described in the previous section.
### TABLE 5.13: Productivity Benchmarks by Application Type

<table>
<thead>
<tr>
<th>AppType</th>
<th>N</th>
<th>LCI</th>
<th>Mean</th>
<th>UCI</th>
<th>SE</th>
<th>SD</th>
<th>Q1</th>
<th>Mdn.</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>C&amp;C</td>
<td>33</td>
<td>120.5</td>
<td>140.7</td>
<td>160.8</td>
<td>9.9</td>
<td>56.8</td>
<td>103.1</td>
<td>128.2</td>
<td>177.8</td>
</tr>
<tr>
<td>CAS</td>
<td>16</td>
<td>309.6</td>
<td>353.4</td>
<td>396.9</td>
<td>20.5</td>
<td>82.2</td>
<td>292.1</td>
<td>323.0</td>
<td>406.6</td>
</tr>
<tr>
<td>COM</td>
<td>47</td>
<td>167.8</td>
<td>189.5</td>
<td>211.2</td>
<td>10.8</td>
<td>74.0</td>
<td>139.6</td>
<td>185.0</td>
<td>243.1</td>
</tr>
<tr>
<td>MP</td>
<td>20</td>
<td>263.4</td>
<td>335.0</td>
<td>406.3</td>
<td>34.2</td>
<td>153.0</td>
<td>207.5</td>
<td>328.7</td>
<td>427.1</td>
</tr>
<tr>
<td>RTE</td>
<td>57</td>
<td>125.1</td>
<td>142.7</td>
<td>160.2</td>
<td>8.7</td>
<td>66.1</td>
<td>84.0</td>
<td>141.1</td>
<td>171.6</td>
</tr>
<tr>
<td>S&amp;S</td>
<td>17</td>
<td>159.6</td>
<td>211.5</td>
<td>263.1</td>
<td>24.5</td>
<td>100.9</td>
<td>129.5</td>
<td>209.8</td>
<td>259.9</td>
</tr>
<tr>
<td>SCP</td>
<td>36</td>
<td>51.6</td>
<td>60.2</td>
<td>68.8</td>
<td>4.2</td>
<td>25.5</td>
<td>40.1</td>
<td>55.7</td>
<td>79.5</td>
</tr>
<tr>
<td>SYS</td>
<td>27</td>
<td>196.3</td>
<td>230.8</td>
<td>265.2</td>
<td>16.8</td>
<td>87.2</td>
<td>162.8</td>
<td>235.4</td>
<td>298.5</td>
</tr>
<tr>
<td>TMDE</td>
<td>11</td>
<td>88.3</td>
<td>160.5</td>
<td>231.9</td>
<td>32.4</td>
<td>107.5</td>
<td>83.7</td>
<td>118.3</td>
<td>205.6</td>
</tr>
<tr>
<td>VC</td>
<td>27</td>
<td>94.5</td>
<td>115.2</td>
<td>135.9</td>
<td>10.1</td>
<td>52.4</td>
<td>70.2</td>
<td>110.1</td>
<td>126.4</td>
</tr>
<tr>
<td>VP</td>
<td>18</td>
<td>68.0</td>
<td>88.5</td>
<td>108.8</td>
<td>9.7</td>
<td>41.1</td>
<td>43.0</td>
<td>90.9</td>
<td>119.5</td>
</tr>
</tbody>
</table>

**FIGURE 5.40: Application Type Productivities Boxplot**
The productivities of some of the Application Types are indistinguishable from each other. Yet the SCP, VP, and VC productivities are different than the productivities for CAS, MP, SYS and S&S. This suggests that groups of application types may be distinguished by productivity.

### 5.5.4 AppType & OpEnv Benchmarks

Table 5.14 shows the productivity means, medians and related statistics across 13 Application Types and Operating Environment subgroups. These are sorted by Operating Environment for each subgroup with 10 or more data points. These benchmarks are shown in the Figure 5.41 boxplot.

**FIGURE 5.41**: Application Type and Operating Environment Productivities Boxplot
TABLE 5.14: Productivity Benchmarks by Operating Environment and Application Type

<table>
<thead>
<tr>
<th>Subgroup</th>
<th>N</th>
<th>LCI</th>
<th>Mean</th>
<th>UCI</th>
<th>SE</th>
<th>SD</th>
<th>Q1</th>
<th>Mdn.</th>
<th>Q3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aerial Vehicle (AV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RTE</td>
<td>23</td>
<td>102.1</td>
<td>133.7</td>
<td>165.2</td>
<td>15.2</td>
<td>73.1</td>
<td>79.6</td>
<td>124.0</td>
<td>164.7</td>
</tr>
<tr>
<td>SCP</td>
<td>11</td>
<td>42.3</td>
<td>60.1</td>
<td>77.7</td>
<td>8.0</td>
<td>26.5</td>
<td>41.0</td>
<td>51.2</td>
<td>94.0</td>
</tr>
<tr>
<td>VC</td>
<td>10</td>
<td>82.2</td>
<td>122.2</td>
<td>161.6</td>
<td>17.7</td>
<td>55.9</td>
<td>76.4</td>
<td>110.9</td>
<td>169.3</td>
</tr>
<tr>
<td>VP</td>
<td>10</td>
<td>75.3</td>
<td>99.7</td>
<td>123.8</td>
<td>10.8</td>
<td>34.1</td>
<td>80.8</td>
<td>92.7</td>
<td>121.7</td>
</tr>
<tr>
<td>Ground System (GS)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C&amp;C</td>
<td>16</td>
<td>126.2</td>
<td>153.6</td>
<td>180.9</td>
<td>12.9</td>
<td>51.4</td>
<td>105.4</td>
<td>143.4</td>
<td>199.8</td>
</tr>
<tr>
<td>CAS</td>
<td>13</td>
<td>309.9</td>
<td>363.4</td>
<td>416.5</td>
<td>24.6</td>
<td>88.6</td>
<td>288.1</td>
<td>353.1</td>
<td>425.9</td>
</tr>
<tr>
<td>COM</td>
<td>22</td>
<td>146.5</td>
<td>186.7</td>
<td>226.8</td>
<td>19.3</td>
<td>90.6</td>
<td>137.3</td>
<td>175.3</td>
<td>243.2</td>
</tr>
<tr>
<td>MP</td>
<td>20</td>
<td>263.4</td>
<td>335.0</td>
<td>406.3</td>
<td>34.2</td>
<td>153.0</td>
<td>207.5</td>
<td>328.7</td>
<td>427.1</td>
</tr>
<tr>
<td>RTE</td>
<td>21</td>
<td>133.4</td>
<td>162.4</td>
<td>191.3</td>
<td>13.9</td>
<td>63.7</td>
<td>133.4</td>
<td>158.6</td>
<td>198.3</td>
</tr>
<tr>
<td>SCP</td>
<td>14</td>
<td>46.1</td>
<td>63.9</td>
<td>81.6</td>
<td>8.2</td>
<td>30.8</td>
<td>31.4</td>
<td>64.8</td>
<td>85.7</td>
</tr>
<tr>
<td>SYS</td>
<td>13</td>
<td>174.8</td>
<td>235.6</td>
<td>295.8</td>
<td>27.9</td>
<td>100.6</td>
<td>144.5</td>
<td>246.2</td>
<td>310.2</td>
</tr>
<tr>
<td>Ground Vehicle (GV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VC</td>
<td>14</td>
<td>79.8</td>
<td>97.9</td>
<td>116.0</td>
<td>8.4</td>
<td>31.4</td>
<td>65.6</td>
<td>104.0</td>
<td>120.7</td>
</tr>
<tr>
<td>Maritime Vessel (MV)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>COM</td>
<td>22</td>
<td>169.7</td>
<td>193.8</td>
<td>218.0</td>
<td>11.6</td>
<td>54.6</td>
<td>172.3</td>
<td>190.0</td>
<td>224.8</td>
</tr>
</tbody>
</table>

5.6 Limitations and Future Work

All the CERs are amenable to change with new data. They reflect only the current dataset and for continuous CER improvement they will be reexamined and derived anew when more SRDR data is available. Future work will also expand the number of CERs for application types and operating environments.

A major limitation of the CERs presented here is the lack of a corresponding schedule estimating relationship, SER. The presented CERs represent the nominal effort required to develop a CSCI of a specific size. But there is no information on what development duration is assumed by the nominal effort estimate.

The amount of allotted development time impacts the amount of required
effort and therefore cost. Development time can be shortened within reason with more people working on the project. Future work will focus on developing SERs to show a nominal duration for a nominal effort estimate and the impact on effort when duration is shortened.

The set of CERs presented are based on software size expressed as estimated SLOC. As discussed in Chapter 1, these CERs would be most useful in the early acquisition phase of a program. Therefore it would be useful to use a size metric that is available in the early phases such as requirements counts. Future work will investigate the use of requirements counts, as reported in the initial SRDR, as a size metric in CERs.
Chapter 6

Modern Estimating Challenges

Several trends present significant future challenges for the sizing and cost estimation of 21st century software systems. Prominent among these trends are:

- Rapid change, emergent requirements, and evolutionary development
- Net-centric systems of systems
- Model-Driven and Non-Developmental Item (NDI)-intensive systems
- Ultrahigh software system assurance
- Legacy maintenance and brownfield development
- Agile and Kanban development

This chapter summarizes each trend and elaborates on its challenges for software sizing and cost estimation.

6.1 Changing Objectives, Constraints and Priorities

6.1.1 Rapid Change, Emergent Requirements, and Evolutionary Development

21st century software systems will encounter increasingly rapid change in their objectives, constraints, and priorities. This change will be necessary due to increasingly rapid changes in their competitive threats, technology, organizations, leadership priorities, and environments. It is thus increasingly infeasible to provide precise size and cost estimates if the systems’ requirements are emergent rather than pre-specifiable. This has led to increasing use of strategies such as incremental and evolutionary development, and to experiences with associated new sizing and costing phenomena such as the Incremental Development Productivity Decline. It also implies that measuring the system’s size by counting the number of source lines of code (SLOC) in the delivered system may be an underestimate, as a good deal of software may be developed and deleted before delivery due to changing priorities.

There are three primary options for handling these sizing and estimation challenges. The first is to improve the ability to estimate requirements volatility during development via improved data collection and analysis, such as the
use of code counters able to count numbers of SLOC added, modified, and deleted during development [37].

If such data is unavailable, the best one can do is to estimate ranges of requirements volatility. For uniformity, Table 6.1 presents a recommended set of Requirements Volatility (RVOL) ranges over the development period for rating levels of 1 (Very Low) to 5 (Very High), such as in the DoD SRDR form [44].

<table>
<thead>
<tr>
<th>Rating Level</th>
<th>RVOL Range</th>
<th>RVOL Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Low</td>
<td>0-6%</td>
<td>3%</td>
</tr>
<tr>
<td>Low</td>
<td>6-12%</td>
<td>9%</td>
</tr>
<tr>
<td>Nominal</td>
<td>12-24%</td>
<td>18%</td>
</tr>
<tr>
<td>High</td>
<td>24-48%</td>
<td>36%</td>
</tr>
<tr>
<td>Very High</td>
<td>&gt;48%</td>
<td>72%</td>
</tr>
</tbody>
</table>

For incremental and evolutionary development projects, the second option is to treat the earlier increments as reused software, and to apply reuse factors to them (such as the percent of the design, code, and integration modified, perhaps adjusted for degree of software understandability and programmer unfamiliarity [8].

This can be done either uniformly across the set of previous increments, or by having these factors vary by previous increment or by subsystem. This will produce an equivalent-SLOC (ESLOC) size for the effect of modifying the previous increments, to be added to the size of the new increment in estimating effort for the new increment. In tracking the size of the overall system, it is important to remember that these ESLOC are not actual lines of code to be included in the size of the next release.

The third option is to include an Incremental Development Productivity Decline (IDPD) factor, or perhaps multiple factors varying by increment or subsystem. Unlike hardware, where unit costs tend to decrease with added production volume, the unit costs of later software increments tend to increase, due to previous-increment breakage and usage feedback, and due to increased integration and test effort. Thus, using hardware-driven or traditional software-driven estimation methods for later increments will lead to underestimates and overruns in both cost and schedule.

A relevant example was a large defense software system that had the following characteristics:

- 5 builds, 7 years, $100M
- Build 1 productivity over 300 SLOC / person-month
- Build 5 productivity under 150 SLOC / person-month
  - Including Build 1-4 breakage, integration, rework
A factor-of-2 decrease in productivity across four new builds corresponds to an average build-to-build IDPD factor of 19%. A recent quantitative IDPD analysis of a smaller software system yielded an IDPD of 14%, with significant variations from increment to increment [54]. Similar IDPD phenomena have been found for large commercial software such as the multi-year slippages in the delivery of Microsoft’s Word for Windows [20] and Windows Vista, and for large agile-development projects that assumed a zero IDPD factor [16].

Based on experience with similar projects, the following impact causes and ranges per increment are conservatively stated in the Table 6.2. The decreased effort due to more experienced personnel assumes a reasonable initial experience level.

<table>
<thead>
<tr>
<th>TABLE 6.2: IDPD Effort Drivers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Less effort due to more experienced personnel</strong></td>
</tr>
<tr>
<td>Variation depending on personnel turnover rates</td>
</tr>
<tr>
<td><strong>More effort due to code base growth</strong></td>
</tr>
<tr>
<td>Breakage, maintenance of full code base</td>
</tr>
<tr>
<td>Diseconomies of scale in development, integration</td>
</tr>
<tr>
<td>Requirements volatility, user requests</td>
</tr>
</tbody>
</table>

In the best case, there would be 20% more effort (from above - 20+20+10+10); for a 4-build system, the IDPD would be 6%. In the worst case, there would be 85% more effort (from above 40+25+25-5); for a 4-build system, the IDPD would be 23%.

With fixed staff size, there would be either a schedule increase or incomplete builds in any case. The difference between 6% and 23% may not look too serious, but the cumulative effects on schedule across a number of builds is very serious.

A simplified illustrative model relating productivity decline to number of builds needed to reach 4M ESLOC across 4 builds follows. Assume that the two-year Build 1 production of 1M SLOC can be developed at 200 SLOC / PM for a total of 5000 PM. This means it will need 208 developers (5000 PM / 24 Months), assuming a constant staff size for all builds.

The analysis shown in Figure 6.1 shows the impact on the amount of software delivered per build and the resulting effect on the overall delivery schedule as a function of the IDPD factor. Many incremental development cost estimates assume an IDPD of zero, and an on-time delivery in 4 builds. However, as the IDPD factor increases and the staffing level remains constant, the productivity decline per build stretches the schedule out to twice as long for an IDPD of 20%.

Thus, it is important to understand the IDPD factor and its influence
when doing incremental or evolutionary development. Ongoing research [35] indicates that the magnitude of the IDPD factor may vary by type of application (infrastructure software having higher IDPDs since it tends to be tightly coupled and touches everything; applications software having lower IDPDs if it is architected to be loosely coupled), or by recency of the build (older builds may be more stable). Research in [35] and [36] determined that IDPD factors were generally not constant, due to personnel turnover, COTS volatility, or pro-active productivity enablers. Further data collection and analysis would be very helpful in improving the understanding of the IDPD factor.

6.1.2 Net-centric Systems of Systems (NCSoS)

If one is developing software components for use in a NCSoS, changes in the interfaces between the component systems and independently-evolving NCSoS-internal or NCSoS-external systems will add further effort. The amount of effort may vary by the tightness of the coupling among the systems; the complexity, dynamism, and compatibility of purpose of the independently-evolving systems; and the degree of control that the NCSoS protagonist has over the various component systems. The latter ranges from Directed SoS (strong control), through Acknowledged (partial control) and Collaborative (shared interests) SoSs, to Virtual SoSs (no guarantees) [40].

For estimation, one option is to use requirements volatility as a way to assess increased effort. Another is to use existing models such as COSYSMO [55] to estimate the added coordination effort across the NCSoS [27]. A third
approach is to have separate models for estimating the systems engineering, NCSoS component systems development, and NCSoS component systems integration to estimate the added effort [28].

6.1.3 Model-Driven and Non-Developmental Item (NDI)-Intensive Development

Model-driven development and Non-Developmental Item (NDI)-intensive development are two approaches that enable large portions of software-intensive systems to be generated from model directives or provided by NDIs such as Commercial-Off-The-Shelf (COTS) components, open source components, and purchased services such as Cloud services.

Such applications are highly cost-effective, but present several sizing and cost estimation challenges:

- Model directives generate source code in Java, C++, or other third-generation languages, but unless the generated SLOC are going to be used for system maintenance, their size as counted by code counters should not be used for development or maintenance cost estimation. In general, auto-generated code is not efficient, does not have meaningful variable names for the internal logic, is relatively cryptic with no comments unless the developer adds them.

- Counting model directives is possible for some types of model-driven development, but presents significant challenges for others (e.g., GUI builders).

- Except for customer-furnished or open-source software that is expected to be modified, the size of NDI components should not be used for estimating.

- A significant challenge is to find appropriately effective size measures for such NDI components. One approach is to use the number and complexity of their interfaces with each other or with the software being developed. Another is to count the amount of glue-code SLOC being developed to integrate the NDI components, with the proviso that such glue code tends to be about 3 times as expensive per SLOC as regularly-developed code [9]. A similar approach is to use the interface elements of function points for sizing [17].

- A further challenge is that much of the effort in using NDI is expended in assessing candidate NDI components and in tailoring them to the given application. Some initial guidelines for estimating such effort are provided in the COnstructive Commercial-off-the Shelf (COCOTS) model [1].

- Another challenge is that the effects of COTS and Cloud-services evolution are generally underestimated during software maintenance. COTS
products generally provide significant new releases on the average of about every 10 months, and generally become unsupported after three new releases. With Cloud services, one does not have the option to decline new releases, and updates occur more frequently. One way to estimate this source of effort is to consider it as a form of requirements volatility.

- Another serious concern is that functional size measures such as function points, use cases, or requirements will be highly unreliable until it is known how much of the functionality is going to be provided by NDI components or Cloud services.

6.1.4 Ultrahigh Software Systems Assurance

The increasing criticality of software to the safety of transportation vehicles, medical equipment, or financial resources; the security of private or confidential information; and the assurance of 24/7 Internet, web, or Cloud services will require further investments in the development and certification of software than are provided by most current software-intensive systems.

While it is widely held that ultrahigh-assurance software will substantially raise software project cost, different models vary in estimating the added cost. For example, the 1990’s Softcost-R model estimates a factor of 3.43 [49]; the SEER model uses a similar value of 3.47 [17].

A recent experimental extension of the COCOMO II model called COSECMO used the 7 Evaluated Assurance Levels (EALs) in the ISO Standard Common Criteria for Information Technology Security Evaluation (CC) [24], and quoted prices for certifying various EAL security levels to provide an initial estimation model in this context [15]. Its added-effort estimates were a function of both EAL level and software size: its multipliers for a 5000-SLOC secure system were 1.50 for EAL 4 and 8.8 for EAL 7.

A further sizing challenge for ultrahigh-assurance software is that it requires more functionality for such functions as security audit, communication, cryptographic support, data protection, etc. These may be furnished by NDI components or may need to be developed for special systems.

6.1.5 Legacy Maintenance and Brownfield Development

Fewer and fewer software-intensive systems have the luxury of starting with a clean sheet of paper or whiteboard on which to create a new Greenfield system. Most software-intensive systems are already in maintenance; [12] estimated in 2009 that there were roughly 200 billion SLOC in service worldwide. Also, most new applications need to consider continuity of service from the legacy system(s) they are replacing. Many such applications involving incremental development have failed because there was no way to separate out the incremental legacy system capabilities that were being replaced. Thus, such
Modern Estimating Challenges

Applications need to use a brownfield development approach that concurrently architects the new version and its increments, while re-engineering the legacy software to accommodate the incremental phase-in of the new capabilities [23] [29] [6].

Traditional software maintenance sizing models have determined an equivalent SLOC size by multiplying the size of the legacy system by its Annual Change Traffic (ACT) fraction (% of SLOC added + % of SLOC modified) / 100. The resulting equivalent size is used to determine a nominal cost of a year of maintenance, which is then adjusted by maintenance-oriented effort multipliers. These are generally similar or the same as those for development, except for some, such as required reliability and degree of documentation, in which larger development investments will yield relative maintenance savings. Some models such as SEER [17] include further maintenance parameters such as personnel and environment differences. An excellent summary of software maintenance estimation is in Stutzke [53].

However, as legacy systems become larger and larger (a full-up BMW contains roughly 100 million SLOC [13], the ACT approach becomes less stable. The difference between an ACT of 1% and an ACT of 2% when applied to 100 million SLOC is 1 million SLOC. A recent revision of the COCOMO II software maintenance model sizes a new release as

\[
ESLOC = 2 \cdot (\text{Modified SLOC}) + \text{Added SLOC} + 0.5 \cdot (\text{Deleted SLOC}) \tag{6.1}
\]

The coefficients are rounded values determined from the analysis of data from 24 maintenance activities [37], in which the modified, added, and deleted SLOC were obtained from a code counting tool. This model can also be used to estimate the equivalent size of re-engineering legacy software in brownfield software development. At first, the estimates of legacy SLOC modified, added, and deleted will be very rough, and can be refined as the design of the maintenance modifications or brownfield re-engineering is determined.

6.1.6 Agile and Kanban Development

The difficulties of software maintenance estimation can often be mitigated by using workflow management techniques such as Kanban [3]. In Kanban, individual maintenance upgrades are given Kanban cards (Kanban is the Japanese word for card; the approach originated with the Toyota Production System). Workflow management is accomplished by limiting the number of cards introduced into the development process, and pulling the cards into the next stage of development (design, code, test, release) when open capacity is available (each stage has a limit of the number of cards it can be processing at a given time). Any buildups of upgrade queues waiting to be pulled forward are given management attention to find and fix bottleneck root causes or to rebalance the manpower devoted to each stage of development. A key Kanban principle is to minimize work in progress.
An advantage of Kanban is that if upgrade requests are relatively small, non-critical and uniform, that there is no need to estimate their required effort; they are pulled through the stages as capacity is available, and if the capacities of the stages are well-tuned to the traffic, work gets done on schedule. However, if a too-large upgrade is introduced into the system, it is likely to introduce delays as it progresses through the stages. Thus, some form of estimation is necessary to determine right-size upgrade units, but it does not have to be precise as long as the workflow management pulls the upgrade through the stages. For familiar systems, performers will be able to right-size the units. For Kanban in less-familiar systems, and for sizing builds in agile methods such as Scrum, group consensus techniques such as Planning Poker [14] or Wideband Delphi [5] can generally serve this purpose.

The key point here is to recognize that estimation of knowledge work can never be perfect, and to create development approaches that compensate for variations in estimation accuracy. Kanban is one such; another is the agile methods’ approach of timeboxing or schedule-as-independent-variable (SAIV), in which maintenance upgrades or incremental development features are prioritized, and the increment architectured to enable dropping of features to meet a fixed delivery date (With Kanban, prioritization occurs in determining which of a backlog of desired upgrade features gets the next card). Such prioritization is a form of value-based software engineering, in that the higher-priority features can be flowed more rapidly through Kanban stages [3], or in general given more attention in defect detection and removal via value-based inspections or testing [11], [30]. Another important point is that the ability to compensate for rough estimates does not mean that data on project performance does not need to be collected and analyzed. It is even more important as a sound source of continuous improvement and change adaptability efforts.

6.1.7 Putting It All Together at the Large-Project or Enterprise-Level

The biggest challenge of all is that the six challenges above need to be addressed concurrently. Suboptimizing on individual-project agility runs the risks of easiest-first lock-in to unscalable or unsecureable systems, or of producing numerous incompatible stovepipe applications. Suboptimizing on security assurance and certification runs the risks of missing early-adopter market windows, of rapidly responding to competitive threats, or of creating inflexible, user-unfriendly systems.

One key strategy for addressing such estimation and performance challenges is to recognize that large systems and enterprises are composed of subsystems that have different need priorities and can be handled by different estimation and performance approaches. Real-time, safety-critical control systems and security kernels need high assurance, but are relatively stable. GUIs need rapid adaptability to change, but with GUI-builder systems, can largely compensate for lower assurance levels via rapid fixes. A key point here is that
for most enterprises and large systems, there is no one-size-fits-all method of sizing, estimating, and performing.

6.2 Estimation Approaches for Different Processes

This diversity of application complexities, objectives and constraints implies a need for guidance on what kind of process to use for what kind of system or subsystem, and on what kinds of sizing and estimation capabilities fit what kinds of processes. A start toward such guidance is provided in [10].

Figure 6.2 summarizes the traditional single-step waterfall process plus several forms of incremental development, each of which meets different competitive challenges and which are best served by different cost estimation approaches. The time phasing of each form is expressed in terms of the increment 1, 2, 3, ... content with respect to the Rational Unified Process (RUP) phases of Inception (I), Elaboration (E), Construction (C), and Transition (T).

The “Single Step” model is the traditional waterfall model, in which the requirements are pre-specified, and the system is developed to the requirements in a single increment. Single-increment parametric estimation models, complemented by expert judgment, are best for this process. The “Pre-specified Sequential” incremental development model is not evolutionary. It just splits up the development in order to field an early Initial Operational Capability, followed by several Pre-Planned Product Improvements (PPIs). When requirements are pre-specifiable and stable, it enables a strong, predictable process. When requirements are emergent and / or rapidly changing, it often requires expensive rework to undo architectural commitments. Cost estimation can be performed by sequential application of single-step parametric models plus the use of an IDPD factor, or by parametric model extensions supporting increments, including options for increment overlap and breakage of existing increments, such as the COCOMO II Incremental Development Model described in [8].

The “Evolutionary Sequential” model rapidly develops an initial operational capability and upgrades it based on operational experience. Pure agile software development fits this model: if something is wrong, it will be fixed in 30 days in the next release. Rapid fielding also fits this model for larger or hardware-software systems. Its strength is getting quick-response capabilities into the field. For pure agile, it can fall prey to an easiest-first set of architectural commitments which break when, for example, it tries to add security or scalability as a new feature in a later increment. For rapid fielding, it may be expensive to keep the development team together while waiting for usage feedback, but it may be worth it. For small agile projects, group consensus techniques such as Planning Poker are best; for larger projects, parametric models with an IDPD factor are best.
Single Step

Prespecified Sequential

Evolutionary Sequential

Evolutionary Overlapped

Evolutionary Concurrent

FIGURE 6.2: Summary of Different Processes
“Evolutionary Overlapped” covers the special case of deferring the next increment until critical enablers such as desired new technology, anticipated new commercial product capabilities, or needed funding become available or mature enough to be added.

“Evolutionary Concurrent” has the systems engineers handling the change traffic and re-baselining the plans and specifications for the next increment, while keeping the development stabilized for the current increment. Its example and pros and cons are provided Table 6.3. All Cost Estimation approaches also include an expert-judgment cross-check.

### TABLE 6.3: Situation-Dependent Processes and Estimation Approaches

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
<th>Pros</th>
<th>Cons</th>
<th>Cost Estimation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Step</td>
<td>Stable; High Assurance</td>
<td>Pre-specifiable full-capability requirements</td>
<td>Emergent requirements or rapid change</td>
<td>Single-increment parametric estimation models</td>
</tr>
<tr>
<td>Pre-specified Sequential</td>
<td>Platform base plus P3Is</td>
<td>Pre-specifiable full-capability requirements</td>
<td>Emergent requirements or rapid change</td>
<td>COINCOMO or repeated single-increment parametric model estimation with IDPD</td>
</tr>
<tr>
<td>Evolutionary Sequential</td>
<td>Small: Agile</td>
<td>Adaptability to change</td>
<td>Easiest-first; late, costly breakage</td>
<td>Small: Planning-poker-type</td>
</tr>
<tr>
<td>Evolutionary Overlapped</td>
<td>COTS-intensive systems</td>
<td>Immaturity risk avoidance</td>
<td>Delay may be non-competitive</td>
<td>Parametric with IDPD and Requirements Volatility</td>
</tr>
<tr>
<td>Evolutionary Concurrent</td>
<td>Mainstream product lines</td>
<td>High assurance with rapid change</td>
<td>Highly coupled systems with very rapid change</td>
<td>COINCOMO with IDPD for development; COSYSMO for re-baselining</td>
</tr>
<tr>
<td></td>
<td>Systems of systems</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 6.4 provides criteria for deciding which of the five classes of incremental and evolutionary acquisition (EvA) defined in Table 6.3 to use, plus the choice of non-incremental, single-step development.

<table>
<thead>
<tr>
<th>Process Type</th>
<th>Stable pre-specifiable requirements?</th>
<th>OK to wait for full system to be developed?</th>
<th>Need to wait for next-increment priorities?</th>
<th>Need to wait for next-increment enablers?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Step</td>
<td>Yes</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Pre-specified Sequential</td>
<td>Yes</td>
<td>No</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Evolutionary Sequential</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Evolutionary Overlapped</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Evolutionary Concurrent</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

The “Single-Step-to-Full-Capability” process exemplified by the traditional waterfall or sequential Vee model is appropriate if the products requirements are pre-specifiable and have a low probability of significant change; and if there is no value in or opportunity to deliver a partial product capability. A good example would be the hardware portion of a geosynchronous satellite.

The “Pre-specified Sequential” process is best if the product’s requirements are pre-specifiable and have a low probability of significant change; and if waiting for the full system to be developed incurs a loss of important and deliverable incremental mission capabilities. A good example would be a well-understood and well-prioritized sequence of software upgrades to a programmable radio.

The “Evolutionary Sequential” process is best when there is a need to get operational feedback on a quick-response capability before defining and developing the next increment’s content. Agile methods fit into this category, as do systems undergoing rapid competitive change.

The “Evolutionary Overlapped” process is best when one does not need to wait for operational feedback, but may need to wait for next-increment enablers such as technology maturity, external system capabilities, or needed resources. A good example is the need to wait for a mature release of an
anticipated commercial product. The Evolutionary Concurrent process is best when the enablers are available, but there is a great deal of change traffic to be handled that would destabilize the team developing the current increment. Examples may be new competitive threats, emergent user capability needs, external system interface changes, technology matured on other programs, or COTS upgrades.

As is frequently stated, “Prediction is difficult, especially about the future.” Emerging trends such as Internet of Things, Big Data Analytics, social networking and crowdsourcing, self-modifying autonomic systems, megacore parallel processing chips, and human-capability augmentation devices will present further challenges in estimating costs of their software. Even more challenging game-breakers lie ahead.
Chapter 7

Estimation Process

7.1 Overview

The software estimation process described in this chapter is based on the U.S. Government Accountability Office Cost Estimating and Assessment Guide [18]. The recommended phases and steps within each phase are explained in terms of software cost estimation. Not all of the steps are described, as some of the steps are specific to how an organization organizes, executes and presents estimates. Where applicable, references to other sections of this manual are used to provide examples of employing metrics data assessment and analysis.

A system level cost estimate involves many cost elements, and the software cost estimate is only a part. This chapter discusses a software estimation process in the context of a system estimate. The following generic system cost element list shows where this software estimation process applies:

Elements covered by this estimation process:

- **Integration, assembly, test, and checkout**
  All effort of technical and functional activities associated with software required to assemble equipment (hardware and software) elements into mission equipment (hardware and software). This includes the activities of software requirements analysis, software architecting, software design, coding and unit testing, software integration and testing, and qualification testing.

Elements not covered by this estimation process:

- **System engineering** The technical and management efforts of directing and controlling a totally integrated engineering effort of a system or program.
- **Program management**
  The business and administrative planning, organizing, directing, coordinating, controlling, and approval actions designated to accomplish overall program objectives not associated with specific hardware elements and not included in systems engineering.
- **Training**
Deliverable training services, devices, accessories, aids, equipment, and parts used to facilitate instruction in which personnel will learn to operate and maintain the system with maximum efficiency.

- **Data**
The deliverable data that must be on a contract data requirements list, including technical publications, engineering data, support data, and management data needed to configure management, cost, schedule, contractual data management, and program management.

- **System test and evaluation**
The use of prototype, production, or specifically fabricated hardware and software to obtain or validate engineering data on the performance of the system in develop (in DoD, normally funded from research, development, test, and evaluation appropriations); also includes all effort associated with design and production of models, specimens, fixtures, and instrumentation in support of the system-level test program.

- **Peculiar support equipment**
Equipment uniquely needed to support the program: vehicles, equipment, tools, and the like to fuel, service, transport, hoist, repair, overhaul, assemble and disassemble, test, inspect, or otherwise maintain mission equipment, as well as equipment or software required to maintain or modify the software portions of the system.

- **Common support equipment**
Operational and site activation; facilities equipment not unique to the program and available in inventory for use by many programs; and installation of mission and support equipment in the operations or support facilities and complete system checkout or shakedown to ensure operational status.

- **Initial spares and repair parts**
Includes the deliverable spare components, assemblies, and subassemblies used for initial replacement purposes in the materiel system equipment end item.

### 7.1.1 Four Estimation Stages

There are four major stages of estimation as shown in Figure 7.1: Initiation and Research, Assessment, Analysis, and Documentation and Presentation. Each stage is described briefly.
Estimation Process

1. Define the Estimate’s Purpose
2. Develop the Estimating Plan
3. Conduct Sensitivity Analysis
4. Conduct a Risk and Uncertainty Analysis
5. Document the Estimate
6. Present Estimate to Management for Approval
7. Update the Estimate to Reflect Actual costs/changes

FIGURE 7.1: GAO’s Cost Estimation Process
7.1.1.1 Initiation and Research

This describes to the recipient of the estimate what is being estimating, and why it is being estimated. There are two steps in this stage:

1. Define the estimate purpose. The purpose of the estimate defines the scope of the estimate, i.e., what is in and what is out of scope. More on this is in Section 7.2.

2. Develop the estimating plan. This step defines the estimation team, the schedule/timeline of estimation activities, the estimation methodology and the alternative methodology to be used. This step is organization specific and is not discussed here.

7.1.1.2 Assessment

The steps in cost assessment are iterative and can be accomplished in varying order or concurrently. These steps are:

1. Define the program (software) that is to estimated. The description of the software program under estimation is important for correctly classifying the software configuration items by Operating Environment and Application Type, discussed earlier in the manual. More on this is in Section 7.3.

2. Determine the structure of the estimate. The structure addresses how the different software configuration items are related, e.g., sequential, overlapped, and how their individual estimates will be aggregated into an overall estimate. More on this is in Section 7.4.2.

3. Identify the ground rules and assumptions. Every estimate is based on some rationale, e.g., analogy, data, cost estimating relationships (as presented in this manual), and, because the estimate is for future development, assumptions about what is going to happen. More on this is in Section 7.4.1.

4. Obtain the data. Estimates should be based data either collected immediately from past project data (if the estimate is an update to an ongoing project) or from historical data from similar projects. The SRDR data discussed in Chapter 2 of this manual is an example of the latter. More on this is in Section 7.5.

5. Develop the point estimate and compare it to an independent estimate. The cost estimating relationships described in this manual produce point estimates. A second estimation method should be used to compare to the point estimate. The sources of the second estimate can be analogous projects or commercial software cost estimate models. More on this is in Section 7.6.
This is an important stage where the relevant cost modeling method(s) are chosen before the estimates are calculated. Simple or detailed CERs such as those provided in this manual may apply here, depending on the program context and acquisition phase.

7.1.1.3 Analysis

Analysis of the estimate expresses the confidence in the point or range of the estimate. There are two steps in this phase:

1. Conduct sensitivity analysis. Once the point estimate is developed, decision makers need to understand how sensitive the total cost estimate is to changes in the data input. Sensitivity analyses should be performed to identify the major cost drivers for the estimate. It determines how the different input ranges affect the point estimates. Cost drivers are varied to see which ones impact the greatest changes in cost. More on this is in Section 7.7.

2. Conduct risk and uncertainty analysis. This is a critical review of the estimate considering the estimation methodology, e.g. cost estimating relationships and assumptions. More on this is in Section 7.8.

7.1.1.4 Documentation and Presentation

Documentation and presentation are the final products of an estimate. This stage has three steps:

1. Document the estimate. This is an important step because it serves to record how the estimate was constructed. This will enable others to review the estimate. It will also be an important input to future estimates. More on this is in Section 7.9.

2. Present the estimate to management for approval. This step is a management presentation of the life cycle cost estimate including a description of the baseline, estimation assumptions, comparison of the estimate to the allocated budget and schedule, discussion of the largest cost elements and cost drivers, all presented in an clear logical manner. This step is organization specific and is not discussed here.

3. Update the estimate to reflect actual costs/changes. Perhaps the biggest mistake is to create an estimate once and never update it with changes in the project. The estimate should be a continuously evolving chronicle of forecasted project costs. The last step above begins with the first step, the purpose of the estimate.
7.2 Estimation Purpose

This is not a large step but it is important because it sets the context of what the estimate is being done. The purpose of the software cost estimate reflects its intended use and drives the amount of resources required to create the estimate. There are many purposes for a software estimate including:

- Evaluation of software affordability (the funds available)
- Analysis of alternatives
- Estimation of the funding required to efficiently execute a program.
- Provide an independent review of software costs
- Assess the cost impact of software baseline changes

The purpose of an estimate is also influenced by two other factors:

1. The software projects acquisition lifecycle stage. See Figure 1.2 on the acquisition lifecycle. Early lifecycle stage estimates will rely mostly on analogous projects. A detailed estimate will be difficult to create with early lifecycle stage projects. Later lifecycle stage estimates will have historical data on which to base the estimate providing more insight into development activity cost and durations.

2. The timeframe covered by the estimate. Different timeframes include:
   - Full lifecycle cost of the software (development and maintenance)
   - Total development cost (requirements analysis to qualification testing for all deliveries)
   - Incremental cost (the cost of the next increment to be delivered)
   - Annual cost

7.3 Program Definition

This is an important step because it is information the estimator will use in creating the estimate. The goal is to define the program in enough detail to understand what software configuration items are going to be developed and in which categories of operating environments and productivity types they belong. As with the purpose of the estimate section above, the description of the program is an important documentation piece explaining what software is to be estimated.

Information on the following should be collected (depending on the softwares acquisition lifecycle phase, some of this information may not be available):
• Name of the program
• Programs purpose, its performance characteristics and all system configurations. This is an indication of complexity.
• The programs acquisition strategy and schedule including the deployment and maintenance plan if that will be required in the estimate.
• Technical details such as:
  – Key functional requirements and performance characteristics for the software
  – The number of platforms the software runs on
  – Who will develop, operate, and maintain the system
  – Descriptions of software components (including interactions, technical maturity of critical components, and standards). A high-level system architecture diagram is very helpful in identifying components and interfaces.
  – The programs relationship to other existing systems, including predecessor or similar legacy systems;
  – Software architecture and the different required configurations
• How the software will interface to other systems
• Information assurance requirements
• Requirements for reliability, security and safety
• Integration test and evaluation concepts and plans.

The technical details also need to provide the estimator with enough information to understand the software application boundaries and interfaces. Application boundaries are one part of defining the estimation scope. Known risks to software development should be collected, e.g.,

• Hard real-time execution requirements
• COTS product incorporation
• Number of interfaces these CSCIs depend upon
• Technical readiness of algorithms
• Security requirements complexity
• Data volume throughput
• Possible external stakeholders contention

7.4 Estimation Scope

It is very important to make explicit what is being estimated. The estimation purpose and program definition drive the estimation scope. The ground rules and assumptions bound what is being estimated and capture uncertainty. The structure of the estimate shows how the estimate will be composed.

At the conclusion of this step, it is highly recommended that the recipients
of the estimate be presented with the estimation scope. This will serve to validate and verify the software estimate.

7.4.1 Ground Rules and Assumptions

The ground rules and assumptions clearly state up front the conditions of the software estimate. Multiple items should be addressed:

- Estimation boundaries, i.e., what the estimate includes and excludes
- Global and program-specific assumptions (these may be expanded later after reconstructing historical development performance)
- Program acquisition lifecycle phase
- Planned build or release schedule
- Schedule or budget constraints
- List of software configuration items in each build or release
- Legacy software that will be used in the development (if applicable)
- Software the government is to furnish as well as the use of existing unique software development facilities (if applicable)
- COTS software that will be used in the development (if applicable)
- Prime contractors and major subcontractors
- Technology refresh strategy, i.e., what and how often refresh will occur
- New technology to be developed, e.g. new algorithms.

Some of this information may only be available from the Program Office. If this is the case, during the data collection step discussed in Section 7.5, the missing information should be collected.

7.4.2 Estimation Structure

Based on the ground rules and assumptions, an estimating structure is developed. Figure 7.2 shows an example decomposition of an aircraft system down to the software elements. The estimating structure resembles these elements and will include all software elements in the aircraft. The structure serves to decompose the estimate into its individual components. It clearly shows how estimates are rolled-up into an aggregate estimates.

Using the WBS in Figure 7.2, the Radar Software would be the top level of the estimation structure in Table 7.1. Radar is decomposed into multiple builds and each build is decomposed into its constituent configuration items. The CSCIs may include COTS software, GOTS software, adapted software (from internal or external sources), auto-generated software and software services in addition to custom-built software.

Each configuration item should have a function description and an application type. The operating environment should be confirmed as there may be mix of environments, e.g. an aircraft could have ground-based software.
FIGURE 7.2: High-Level Aircraft WBS Decomposed to Software Elements

TABLE 7.1: Radar Software WBS Decomposition

**Radar Software**

- **Build-1**
  - CSCI-1 Control Unit (activity estimation for Requirements, Architecture, Design, Implementation and CSCI testing)
  - CSCI-2 Item Processing
  - CSCI-3 Track Database
  - CSCI-CSCI Integration
  - Hardware Installation and Checkout
  - Acceptance Testing

- **Build-2**
  - CSCI-1 Control Unit
  - CSCI-2 Item Processing
  - CSCI-4 Display Manager
  - CSCI-5 Built-In-Test
  - CSCI-CSCI Integration
  - Hardware Installation and Checkout
  - Acceptance Testing
### 7.5 Data Collection and Normalization

In this step, create a data collection plan and collect historical data with actual results. Much of the data will come from the Program Office. An interview with the Program Office should be part of this plan. The data collection plan would include:

- All of the data not found during the Program Definition step
- Software size from previous builds or releases.
- Staffing levels and/or effort for each build or release
- Software build or release durations with start/end dates
- Document the start/end criteria for each build or release
- Cost model parameters, such as those in Appendix A, if they are going to be used in creating the estimate
- Other cost model calibration data if required
- Average labor rate ($) to convert estimated effort to dollars
- Number of labor hours in a year
- Software risks, e.g., requirements stability, algorithm maturity.

There are at least two sources of historical data.

1. Historical data from this program. This is the best source of data. The size, effort and duration data show actual development performance.
2. Similar historical SRDR data for each CSCI’s application type. This manual discusses SRDR data in Chapter 2 and what is available.

Document all pertinent information, including an assessment of data reliability and accuracy, discussed in Chapter 4. This will be an input to documenting the estimate.

After the data is collected it must be normalized, i.e., converted to the same measurement units and cover the same activities. Chapter 4 discusses data normalization.

- Use the size definition checklists in Tables 3.2 and 3.3 to collect size data and normalize it to a logical count.
- Use the definition of labor activities in Table 3.9 and the guidelines in section 3.4 to document the data collected. SRDR activities and labor categories are discussed in Section 4.5.2.
- Match duration with effort activities above, e.g. requirements, architecting, detailed design, implementation, etc.

Analyze the data for anomalies and trends. Compare results against the application type benchmarks in section 5.5.1 provided in this manual. Finally, store the collected data for model calibration and for future estimates.
7.6 Estimate Creation

This step starts with either developing productivity benchmarks and cost estimating relationships as discussed in Chapter 5, using the benchmarks and CERs contained in this manual, or calibrating existing cost estimation models. Commercial cost models in Appendix A have their own calibration requirements and are not discussed here. The use of productivity benchmarks, cost estimating relationships or commercial cost models are referred to here as estimating methodologies.

7.6.1 Historical Reconstruction

As a final step in calibration, perform a historical reconstruction of past software builds or releases for both effort and duration. This important activity informs the estimator about the suitability of the estimation methodology to predict past software development activities. A large difference between the estimation methodology output and past performance may require further calibration.

Historical reconstruction should be reviewed with the Program Office for feedback and insights. Questions and assumptions required to recreate the past development effort and duration should be discussed. These discussions may lead to another round of calibration of the estimation methodology. Feedback from the Program Office should be documented.

Another benefit of historical reconstruction and Program Office feedback is the identification of addition assumptions not apparent from earlier analysis of the program. All or some of these assumptions may be present in creating the future estimate. These need to be documented.

7.6.2 Estimate Construction

To develop a software estimate for future development, select an estimation methodology suitable for each CSCI in the estimation structure discussed earlier. This usually requires a mixture of methodologies similar to those used in reconstruction.

The estimated effort should be distributed over the development activity durations for each configuration item in the estimation structure, see Section 5.4.5. Then the durations for each configuration item are overlaid by their planned start/end dates. Then the builds or releases are overlaid by their planned start/end dates. This can be accomplished using a spreadsheet. Commercial models do this task automatically.

The overlaid effort of the configuration items and the builds or releases creates an overall effort profile across the software development (similar to the cover page of this manual). A cost estimate can now be created from this.
7.6.3 Estimate Cost Analysis

Convert the effort estimate for each month in the effort profile into dollars using the average labor rate from the data collection step. Be sure to express costs in base-year dollars. A base-year is a selected year to which all costs are deflated or inflated (using an approved index) to normalize the dollar values so comparisons can be made across years. Past projects and historical costs would be inflated. Future costs would be deflated if they were adjusted for inflation in another estimate.

Compare the effort, adjusted base-year cost and duration estimate against analogous programs. Examine where and why there are differences. Look for errors like double counting and omitted costs.

After comparisons have been made, the estimate should be adjusted for then-year dollars. Then-year dollars are the costs for future years adjusted for inflation. The inflation rates are obtained from an approved index such as from the Office of Management and Budget.

Use a cost-estimating checklist, such as the Software Engineering Institute’s A Manager’s Checklist for Validating Software Cost and Schedule Estimates in [48], to review the estimate. At a high level, there are seven parts to this checklist:

1. Are the objectives of the estimate clear and correct?
2. Has the task been appropriately sized?
3. Is the estimated cost and schedule consistent with demonstrated accomplishments on other projects?
4. Have the factors that affect the estimate been identified and explained?
5. Have steps been taken to ensure the integrity of the estimating process?
6. Is the organization’s historical evidence capable of supporting a reliable estimate?
7. Has the situation changed since the estimate was prepared?

Each of these parts are discuss in more detail in the SEI report.

7.6.4 Alternate Estimate

It is considered a best practice to create a second independent estimate using a different estimation methodology. The value in creating a second independent estimate comes from comparing the two results. Estimation methodology inputs and assumptions will be highlighted and errors/omissions will be discovered. This exercise will improve the overall quality of the estimate. The goal should be to get both estimates to within 10% of each other. As an example of comparing different methodologies, Appendix A contains a comparison of several cost model parameters and assumptions.

Software cost estimation should not be a one-and-done activity. The estimation methodologies should be updated as more data becomes available or if conditions change on the program. The applicable estimate steps should then be repeated.
7.7 Sensitivity Analysis

A simple approach to associating cost to risk is to create different development scenarios based on the highest probable set of risks occurring. This consists of changing the estimated software size (based on a growth factor and/or uncertainty) and other model inputs. Identify the effects to the overall estimate. Determine which risks have the highest impact on effort and duration and which estimation structure elements are affected most by changes.

7.8 Risk and Uncertainty Analysis

A software cost estimate is uncertain. Risk analysis captures the uncertainty about the point estimate (discussed in the previous step) and the anticipated growth in cost and duration. Risk is expressed as a distribution around the estimate.

There are several approaches to quantifying risk and creating a distribution around a cost estimate. Two simple approaches are discussed here. Additional approaches and more detailed discussion on the topic can be found in the GAO Guidebook [18], Air Force Cost Risk and Uncertainty Analysis Handbook [2], and NASAs Probabilistic Risk Assessment Procedures Guide [52].

One way to determine whether a program is realistically budgeted is to perform an uncertainty analysis, so that the probability associated with achieving its budget point estimate can be determined. A cumulative probability distribution (CDF), also known as an S-curve, is particularly useful in portraying the uncertainty implications of cost estimates. It shows the probability of occurrence, or the percent confidence of achieving particular costs.

The CDF is usually derived from a simulation technique such as Monte Carlo analysis using random samples from input probability distributions. Monte Carlo simulation produces a CDF per Figure 7.3. This example shows a cost estimate mapped to probability levels. The point estimate created in the previous step can be found on the distribution curve with its probability of occurrence in Figure 7.3.

For instance, if a point estimate was 235 Person-Months of effort, Figure 7.3 shows a 25% probability of occurrence. A 300 Person-Month estimate has 50% probability, and there is a 70% probability of achieving a cost of less than or equal to 350 Person-Months.

The basic tasks for carrying out a risk analysis are:

1. Determine program risks. This is a subjective task based on interviews with technical experts familiar with the program. Possible categories of risk for software are
Stakeholder contention over software functionality, funding and schedule
- Requirements growth and volatility
- Subcontractor performance
- Legacy software function shortfall
- Technology immaturity
- Unplanned rework
- Inadequate staffing or the wrong skill set
- Not enough allocated schedule, i.e., the estimated build duration is longer than the allocated schedule.
- Inefficient production processes.

Identified risks should be given a score as to their impact on cost, e.g., 1 = low impact, 5 = medium impact, and 10 = high impact.

2. Select the probability distributions to model uncertainty in the cost inputs. Some simple, useful common probability distributions are described next.

7.8.1 Common Probability Distributions

Some common probability distributions used in cost estimation are shown in Figure 7.4. These include the uniform, triangular, normal and lognormal distributions to model uncertainty. Each of these is described
next in terms of their characteristics and applications.

**Uniform**

A uniform distribution represents a range of equally likely outcomes with an equal number of measures in each interval. As shown in Figure 7.4, all values in between the range endpoints of $a$ and $b$ are equally likely. It can be used as an initial model for a quantity that is felt to be randomly varying between $a$ and $b$ but about which little else is known. It can be used:

- in the absence of distribution data
- to represent equally likely outcomes (e.g., decision choices)
- for many software cost drivers.

The uniform is often used with engineering data or analogy estimates [18].

![Uniform Distribution](image)

**Triangular**

The triangular distribution is used when the minimum, maximum and
most likely values of a random variable are known but no other information is available. It is characterized by three points, can be skewed or symmetric and is easy to understand because it is intuitive. A triangular distribution is often used as a rough model in the absence of data. It accounts for a tapered distribution with the values being centered around the \( c \) location parameter, the most likely value. The limiting cases where \( c = b \) and \( c = a \) are called right triangular and left triangular distributions respectively.

The triangular distribution is commonly used to express technical uncertainty in software cost estimation because of its use of minimum, most likely and maximum inputs. It is often used as an initial rough model when the minimum, most likely and maximum are known but no other distribution shape data is known. Software cost drivers, particularly size, are frequently represented with triangular distributions.

**Normal**

A normal distribution is symmetric around the mean and bell-shaped (also called a bell curve) in which most measures occur in the closest to the center of the distribution and fewer in the intervals farther from the center. Figure 7.4 shows a normal distribution centered around its mean \( \mu \).

The symmetric normal distribution is used for outcomes likely to occur on either side of the average value. For software these may include:

- personnel factors representing a normal spread of capability
- size, effort, complexity, other project characteristics

Besides technical uncertainty it is used to assess the uncertainty of cost estimating relationships, as presented in Chapter 5.

**Lognormal**

The lognormal is a continuous distribution positively skewed with a limitless upper bound and known lower bound. It is skewed to the right to reflect the limitless upper bound.

In the skewed distribution the measurements cluster around a certain value, but that value is not in the center of the distribution. In the lognormal distribution \( \ln(x) \) is normally distributed. The distribution mean is the mean of \( \ln(x) \), and the standard deviation is the standard deviation of \( \ln(x) \). Figure 7.4 shows a sample lognormal distribution. The dispersion around the mean is characterized by the standard deviation.

The lognormal distribution is often a good one to model the overall size or complexity of a software project, as one tends to underestimate the maximum possible values. It can be used:

- for uncertainty in system size, effort, complexity
• accounting for typical underestimation of scope and system-wide effects

One drawback of a lognormal function is its extremely long tail. For this a truncated lognormal function bounds the distribution to more realistic end values. The truncated lognormal prevents non-positive values or other inconsistent values from a standard lognormal (e.g. when the mean is close enough to zero that the spread goes negative).

The lognormal function can take on shapes similar to the gamma function or the Weibull distribution (the Rayleigh curve is actually a special case of the Weibull distribution). The lognormal is also used characterize uncertainty in nonlinear cost estimating relationships.

See [18] for additional probability distributions, their descriptions and typical applications.

3. Continue the risk analysis by accounting for correlation between cost elements. This identifies correlations between different elements in the estimation structure, e.g. when one element increases in cost the correlated element also increases in cost. Care must be taken when sampling that when one element has a high cost value the correlated element does not have a low cost value producing inconsistency.

4. Perform uncertainty analysis with a Monte Carlo simulation. CER parameters are modeled as distributions according to Step 2 resulting in a cost cumulative distribution function per Figure 7.3. This technique is illustrated for a CER in Section 7.8.2.

5. Identify the probability associated with the point estimate. The output from the Monte Carlo simulation will produce a cost cumulative probability distribution, see Figure 7.3. Locate the point estimate on the cumulative curve.

6. Recommend sufficient contingency reserves. Identify the amount of contingency funding and add this to the point estimate to determine the risk-adjusted cost estimate.

7. Allocate risk-adjusted cost across development builds or releases.

7.8.2 Monte Carlo Analysis with a CER

An illustration of the Monte Carlo simulation technique will be applied to the RTE CER, equation 5.4 derived in Chapter 5:

\[ PM = 13.16 \cdot KESLOC^{0.84} \]  
(7.1)

We choose a normal distribution of size represented in Figure 7.5 with a mean value of 100 and standard deviation of 20 EKSLOC.
The RTE CER is computed many times at each random value of KESLOC drawn from the normal distribution in Figure 7.5. The Monte Carlo simulation results are based on 1000 runs. The probability distribution function shows the simulated effort intervals in Figure 7.6, and the frequencies are summed for increasing size in the cumulative distribution function in Figure 7.7.

The smoothed line in Figure 7.7 is the continuous CDF corresponding to earlier Figure 7.3 where the probability of achieving a given estimate can be determined. For example, a safe 80% confidence level estimate corresponds to about 700 Person-Months.
This Monte Carlo simulation example covered the technical uncertainty associated with the size. A more thorough simulation would also account for the CER error distribution as shown in Figure 5.9. Both sources of uncertainty would be reflected in the simulation output.

### 7.9 Estimate Documentation and Packaging

The documentation step provides a record of the software estimate. There should be enough detail that a cost analyst unfamiliar with the program can recreate the estimate and produce the same result. This will require the documentation to include background information as well as information on the estimation methodology and results. Background documentation should include:

- The purpose of the estimate, the team that prepared it, and who approved the estimate and on what date
- Description of the program, performance characteristics and system configurations
- Program acquisition strategy including deployment and maintenance
- Program funding profile and schedule
- Program technical details such as key functional requirements, number of platforms, development contractor, and a high-level architecture diagram
- Programs relationship to other existing systems
Software architecture for different configurations.

Estimation methodology documentation should include:

- Ground rules and assumptions
- Estimation structure with a description and classification of each cost element
- Data sources for each cost element
- How the data were normalized
- Past program technical data: software size, effort, build or release durations with an assessment of the data quality and reliability
- Program office feedback and insights on the historical reconstruction of previous builds or releases.

Estimation results documentation should include:

- Estimating methodology and rationale used to derive each estimation structure element’s cost
- Programs time-phased life-cycle cost containing the effort profile and duration across all builds or releases.
- Results of the risk, uncertainty, and sensitivity analyses
- Comparison of how the estimate compares to the funding profile
- Differences in how this estimate compares to any previous estimates.

The software estimation process described in this chapter is based on GAO guidance [18]. Where applicable, references to other sections of this manual are used to provide examples of employing metrics data to make cost estimates.

A software cost estimate is only part of the system level estimate. This chapter focused on estimating the cost of software activities associated with development, i.e., requirement analysis through acceptance activities. As pointed out there may be other activities requiring a cost estimate, e.g., training, data, support for system test and evaluation. The estimation structure can be expanded to handle these other sources of cost.
Appendix A

Cost Model Descriptions

A.1 Introduction

This appendix summarizes and compares the parametric software cost models most frequently used in the DoD. Transformations between the respective models are provided so projects can be represented in all models in a consistent manner and to help understand why estimates may vary. These descriptions are subject to change, and the reader should check for updates and more complete details as necessary.

The cost models used in sea, space, ground, and air platforms by the services are generally based on the common effort formula shown in Equation A.1. Size of the software is provided in a number of available units, cost factors describe the overall environment and calibrations may take the form of coefficients adjusted for actual data or other types of factors that account for domain-specific attributes [31], [32]. The total effort is calculated and then decomposed by phases or activities according to different schemes in the models.

\[ \text{Effort} = A \cdot \text{Size}^B \cdot \text{EM} \]  

(A.1)

where

- \( \text{Effort} \) is in Person Months
- \( A \) is a constant derived from historical project data
- \( \text{Size} \) is software size in source lines of code or other size measures
- \( B \) is a scale factor
- \( \text{EM} \) is an effort multiplier from cost factors.

The models can use size expressed as lines of code, function points, object-oriented metrics and other measures. Each model has its own respective cost factors for the linear effort multiplier term, and each model specifies the \( B \) scale factor in slightly different ways (either directly or through other factors). Some models use project type or application domain to improve estimating accuracy. Others use alternative mathematical formulas to compute their estimates. Sizing for the models is summarized in Table A.1. A comparative analysis of cost model factors is listed in Table A.2. The model WBS phases and activities are in Tables A.3 and A.4 respectively.
A.2 Cost Models

The models covered include COCOMO II, SEER-SEM, SLIM, and True Planning. They were selected because they are the most frequently used models for estimating DoD software effort, cost and schedule. A comparison of the COCOMO II, SEER-SEM and True Planning models for NASA projects is described in [32]. A previous study analyzed the same three models with respect to flight and ground projects [31], and another comparative survey of software cost models can also be found in [33].

COCOMO II is a public domain model and is implemented in several commercial tools. True Planning and SEER-SEM are both proprietary commercial tools with unique features but also share some aspects with COCOMO. SLIM is another parametric tool that uses a different approach to effort and schedule estimation.

Any of the models can be used effectively if properly calibrated. Each model has strengths and weaknesses, thus the studies recommend using at least two models to estimate costs whenever possible to provide added assurance that you are within an acceptable range of variation.

A.2.1 COCOMO II

The COCOMO (COnstructive COst MOdel) cost and schedule estimation model was originally published in [5] and the model continues to be updated at USC. The COCOMO II model defined in [8] has three submodels: Applications Composition, Early Design and Post-Architecture. They can be combined in various ways to deal with different software environments. The Application Composition model is used to estimate effort and schedule on projects typically done as rapid application development. The Early Design model involves the exploration of alternative system architectures and concepts of operation. Typically, not enough is known to make a detailed fine-grained estimate. This model is based on function points (or lines of code when available) and a set of five scale factors and seven effort multipliers.

The most frequently used is the Post-Architecture model for when top level design is complete and detailed information about the project is available and the software architecture is well defined. It uses Source Lines of Code and/or Function Points for the sizing parameter, adjusted for reuse and breakage; a set of 17 effort multipliers and a set of five scale factors that determine the economies/diseconomies of scale of the software under development. The effort formula is:

\[
Effort = A \cdot Size^B \cdot \prod_{i=1}^{N} EM_i
\]  
(A.2)
Where

- **Effort** is Person Months
- **A** is a constant derived from historical project data
- **Size** is in KSLOC (thousand source lines of code), or converted from other size measures
- **B** is an exponent for the diseconomy of scale dependent on additive scale drivers
- **$EM_i$** is an effort multiplier for the *i*th cost driver. The geometric product of *N* multipliers is an overall effort adjustment factor to the nominal effort.

The COCOMO II effort is decomposed by lifecycle phase and activity as detailed in Section A.4.

### A.2.2 True Planning

True Planning is offered by PRICE Systems, and initially was called PRICE S. It fits into a composite modeling system and can be used to estimate more than just software costs. Many of the models central algorithms were published in [45]. For more details on the model and the modeling system see the PRICE Systems website at http://www.pricesystems.com.

The True Planning model consists of three submodels that enable estimating costs and schedules for the development and support of computer systems. The model covers business systems, communications, command and control, avionics, and space systems. It includes features for reengineering, code generation, spiral development, rapid development, rapid prototyping, object-oriented development, and software productivity measurement. Size inputs include SLOC, function points and/or Predictive Object Points (POPs). The True Planning system also provides a COCOMO II capability. Some of the descriptions herein may retain the old PRICE S terminology for the cost factors, such as the Rosetta Stone mapping between models.

### A.2.3 SEER-SEM

SEER-SEM is a product offered by Galorath, Inc. This model is based on the original Jensen model [26], which derives from COCOMO and other models in its mathematical formulation. However, its parametric modeling equations are proprietary. SEER-SEM estimates can be used as part of a composite modeling system for hardware/software systems. Descriptive material about the model can be found in [17].

The scope of the model covers all phases of the project life-cycle, from early specification through design, development, delivery and maintenance. It handles a variety of environmental and application configurations, and models different development methods and languages. Development modes covered
include object oriented, reuse, COTS, spiral, waterfall, prototype and incremental development. Languages covered are 3rd and 4th generation languages (C++, FORTRAN, COBOL, Ada, etc.), as well as application generators.

The SEER-SEM cost model allows probability levels of estimates, constraints on staffing, effort or schedule, and it builds estimates upon a knowledge base of existing projects. Estimate outputs include effort, cost, schedule, staffing, and defects. Sensitivity analysis is also provided as is a risk analysis capability. Many sizing methods are available including lines of code and function points. For more information, see the Galorath Inc. website at http://www.galorath.com.

A.2.4 SLIM

The Software Lifecycle Management (SLIM) model is offered by Quantitative Software Management. It was developed by Putnam based on a Norden/Rayleigh manpower distribution and his finding in analyzing many completed projects [46]. The central part of Putnam’s model called the software equation is:

\[ S = C_k \cdot Effort^{1/3} \cdot t_d^{4/3} \]  \hspace{1cm} (A.3)

Where

- \( S \) is Size of the software
- \( Effort \) is Person Months
- \( t_d \) is the software delivery time
- \( C_k \) is a productivity environment factor

The productivity environment factor reflects the development capability derived from historical data using the software equation.

Another relation found by Putnam is

\[ Effort = D_0 \cdot t_d^3 \]  \hspace{1cm} (A.4)

where \( D_0 \) is the manpower build-up parameter which ranges from 8 (entirely new software with many interfaces) to 27 (rebuilt software). The SLIM software tool is based on this model for cost estimation and manpower scheduling [34].
Appendix A - Cost Model Descriptions

A.3 Cost Model Input Factors

A.3.1 Software Size

A.3.1.1 Overview and Sizing Units

This section describes software sizing inputs in the models. The primary unit of software size is Thousands of Source Lines of Code (KSLOC). The models also support Function Point-based sizing, conversions from from other size measures to KSLOC, or additional size units can be used directly in the models. User-defined proxy sizes can also be developed for any of the models.

All models support size inputs for new and adapted software, and some support automatically translated or generated code. The models differ with respect to their detailed parameters for the developed categories of software. The size inputs for the models are described in subsequent sections. The respective software adaptation models for determining equivalent size are also elaborated. Table A.1 summarizes the size factors and maps them between the models.

COCOMO II

The COCOMO II size model is based on SLOC or function points converted to SLOC, and can be calibrated and used with other software size units. Examples include use cases, use case points, object points, physical lines, and others. Alternative size measures can be converted to lines of code and used directly in the model or it can be independently calibrated directly to different measures.

COCOMO II uses Unadjusted Function Points for sizing, and applies its reuse factors, cost drivers, and scale drivers to this sizing quantity to account for the effects of reuse, distribution, etc. on project effort. The COCOMO II procedure for determining Unadjusted Function Points follows the definitions in [22] in both the Early Design and the Post-Architecture models. The Unadjusted Function Points are converted to ESLOC for the effort and schedule formulas using default or user-supplied conversion ratios [8].

SEER-SEM

In SEER-SEM several sizing units can be used alone or in combination. It can use SLOC, function points and custom proxies. SEER provides several variations of function point counts. Its Function-Based Sizing is consistent with IFPUG counting rules, with the addition of Internal Functions for algorithmic processes. It also has proxy sizing for Mark II Function Points and Function Points for direct input of a total function point count.

SEER allows custom proxies as a flexible way to estimate software size. Any countable artifact can be established as a measure. Custom proxies can be used with other size measures in a project.
SEER converts all size data into internal size units, also called effort units, Users can combine or select a single metric for any project element or for the entire project.

The SEER function-based sizing uses traditional function point types described in Section 3.3.1:

- External Inputs (EI)
- External Outputs (EO)
- Internal Logical Files (ILF)
- External Interface Files (EIF)
- External Inquiries (EQ)
- Internal Functions (IF) any functions that are neither data nor transactions.

Additional pre-defined proxy measures for size include:

- Web Site Development
- Mark II Function Points
- Function Points (for direct IFPUG-standard function points)
- Object-Oriented Sizing.

COTS sizing elements available in SEER are in section A.3.1.6.

True Planning

The True Planning software cost model size measures may be expressed in different size units including source lines of code (SLOC), function points, predictive object points (POPs) or Use Case Conversion Points (UCCPs). True Planning also differentiates executable from non-executable software sizes. Functional Size describes software size in terms of the functional requirements that you expect a Software COTS component to satisfy. The True Planning software cost model size definitions for all of the size units are listed below.

SLIM

SLIM uses effective system size composed of new, modified and reused code. Deleted code is not considered in the model. If there is reused code than the Productivity Index ($PI$) factor may be adjusted to add in time and effort for regression testing and integration of the reused code.

SLIM provides different sizing techniques including:

- Sizing by history
- Total system mapping
- Sizing by decomposition
- Sizing by module
- Function point sizing.

Alternative sizes to SLOC such as use cases or requirements can be used in Total System Mapping. The user defines the method and quantitative mapping factor.
### TABLE A.1: Cost Model Size Inputs

<table>
<thead>
<tr>
<th></th>
<th>COCOMO II</th>
<th>SEER-SEM</th>
<th>True Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>New Software</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>New Size</td>
<td>New Size</td>
<td>New Size</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>New Size</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Non-executable</td>
<td></td>
</tr>
<tr>
<td><strong>Modified Software</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adapted Size</td>
<td>Pre-exists Size</td>
<td>Adapted Size</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Adapted Size</td>
<td>Non-executable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Amount of Modification %</td>
<td></td>
</tr>
<tr>
<td>% Design Modified</td>
<td>Redesign Required %</td>
<td>% of Design Adapted</td>
<td></td>
</tr>
<tr>
<td>% Code Modified</td>
<td>Reimplementation Required %</td>
<td>% of Code Adapted</td>
<td></td>
</tr>
<tr>
<td>% Integration Required</td>
<td>Retest Required %</td>
<td>% of Test Adapted</td>
<td></td>
</tr>
<tr>
<td>Assessment and Assimilation</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Software Understanding</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Programmer Unfamiliarity</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted Size</td>
<td>Deleted Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Code Removal Complexity</td>
</tr>
<tr>
<td><strong>Reused Software</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reused Size</td>
<td>Pre-exists Size</td>
<td>Reused Size</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reused Size</td>
<td>Non-executable</td>
</tr>
<tr>
<td></td>
<td>Redesign Required %</td>
<td>% of Design Adapted</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Reimplementation Required %</td>
<td>% of Code Adapted</td>
<td></td>
</tr>
</tbody>
</table>

(continued)
TABLE A.1: Cost Model Size Inputs (continued)

<table>
<thead>
<tr>
<th>COCOMO II</th>
<th>SEER-SEM</th>
<th>True Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>% Integration Required</td>
<td>Retest Required %</td>
<td>% of Test Adapted</td>
</tr>
<tr>
<td>Assessment and Assimilation</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-</td>
<td>Deleted Size</td>
<td>Deleted Size</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Code Removal Complexity</td>
</tr>
</tbody>
</table>

**Generated Software**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>Auto Generated Code Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Auto Generated Size Non-executable</td>
</tr>
</tbody>
</table>

**Automatically Translated Software**

<table>
<thead>
<tr>
<th>Adapted SLOC</th>
<th>-</th>
<th>Auto Translated Code Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Automatic Translation Productivity</td>
<td>-</td>
<td>Auto Translated Size Non-executable</td>
</tr>
<tr>
<td>% of Code Reengineered</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

### A.3.1.2 New Software

New software is interpreted the same in all the cost models. The *New Size* parameter is used in COCOMO II to designate new software. New software in SEER is represented by *New Lines of Code* or *New Functions*. In True Planning, *New Code Size* is the amount of new code that does not reuse any design, code, or test artifacts. *New Size Non-executable* is the percentage of the *New Code Size* that is non-executable (such as data statements, type declarations, and other non-procedural statements). *New software* is also used in SLIM to designate software created for the first time.

### A.3.1.3 Adapted Software (Modified and Reused)

All the software adaptation models involve estimating *AAF* as previously described for modified and reused code. The linear approximation is used to convert reused software into equivalent lines of code in the models. However,
the models have other adjustment factors or parameterizations for $AAF$ as described next.

**COCOMO II**

The COCOMO II treatment of software adaptation includes a nonlinear estimation model with the following parameters [8]:

- *Software Understanding (SU)* is an effort increment as a percentage based on the software structure, application clarity and self-descriptiveness.

- *Unfamiliarity (UNFM)* is the programmer’s relative unfamiliarity with the software which is applied multiplicatively to the *Software Understanding* increment.

- *Assessment and Assimilation (AA)* measures how much relative effort is needed to determine whether a reused software module is appropriate to the application, and to integrate its description into the overall product description. AA is expressed as an effort percentage increment.

The overall *Adaptation Adjustment Multiplier (AAM)* in COCOMO II uses the factors for $AAF$, $SU$, $UNFM$ and $AA$ as:

\[
AAM = \frac{AA + AAF(1 + 0.02 \cdot SU \cdot UNFM)}{100}, \text{ for } AAF \leq 50%
\]

\[
AAM = \frac{AA + AAF + SU \cdot UNFM}{100}, \text{ for } AAF > 50%
\]

**SEER**

The software adaption parameters in SEER are applied to the category *Pre-Existing* software which is modified to fit into a new system. There are two categories of pre-existing software:

- *Pre-existing, Designed for Reuse*
- *Pre-existing, Not Designed for Reuse.*

Each category is then composed of:

- *Pre-existing lines of code* which is the number of lines from a previous system
- *Lines to be Deleted* are those lines deleted from a previous system.

With these additional inputs SEER will automatically assign lower rework percentages to software that is designed for reuse. The adaption parameters applied to the pre-existing software to determine equivalent size are:

- *Redesign Required* is the percentage of existing code that must be redesigned to meet new system requirements.
Reimplementation Required is the percentage of existing code that must be re-implemented, physically recoded, or reentered into the system, such as code that will be translated into another language.

Retest Required is the percentage of existing code that must be retested to ensure that it is functioning properly in the new system.

SEER then uses different proportional weights with these parameters in their AAF equation according to

\[
\text{Pre-existing Effective Size} = 0.4 \cdot A + 0.25 \cdot B + 0.35 \cdot C.
\]

Where \( A \), \( B \) and \( C \) are the respective percentages of code redesign, code reimplementation, and code retest required. In the standard AAF formula, \( DM \) is from Redesign Required, \( CM \) is Reimplementation Required and \( IM \) is Retest Required.

True Planning

The True Planning software adapted size parameters are detailed below.

- **Adapted Code Size** - The amount of existing code that must be changed, deleted, or adapted for use in the new software project. When the value is zero (0.00), the value for New Code Size or Reused Code Size must be greater than zero.

- **Adapted Size Non-executable** - The percentage of the adapted code size that is non-executable (such as data statements, type declarations, and other non-procedural statements). Typical values for fourth generation languages range from 5% to 30%. When a value cannot be obtained by any other means, the suggested nominal value for non-executable code is 15%.

- **Amount for Modification** - The percent of the component functionality that you plan to modify, if any. The Amount for Modification value (like Glue Code Size) affects the effort calculated for the Software Design, Code and Unit Test, Perform Software Integration and Test, and Perform Software Qualification Test activities.

- **Percent of Code Adapted** - The percentage of the adapted code that must change to enable the adapted code to function and meet the software project requirements.

- **Percent of Design Adapted** - The percentage of the existing (adapted code) design that must change to enable the adapted code to function and meet the software project requirements. This value describes the planned redesign of adapted code. Redesign includes architectural design changes, detailed design changes, and any necessary reverse engineering.
Appendix A - Cost Model Descriptions

- **Percent of Test Adapted** - The percentage of the adapted code test artifacts that must change. Test plans and other artifacts must change to ensure that software that contains adapted code meets the performance specifications of the Software Component cost object.

- **Reused Code Size** - The amount of pre-existing, functional code that requires no design or implementation changes to function in the new software project. When the value is zero (0.00), the value must be greater than zero for New Code Size or Adapted Code Size.

- **Reused Size Non-executable** - The percentage of the Reused Code Size that is non-executable (such as, data statements, type declarations, and other non-procedural statements). Typical values for fourth generation languages range from 5% to 30%. If a value cannot be obtained by any other means, the suggested nominal value for non-executable code is 15%.

- **Equivalent Source Lines of Code** - The ESLOC (Equivalent Source Lines of Code) value describes the magnitude of a selected cost object in equivalent Source Lines of Code size units. True Planning does not use ESLOC in routine model calculations, but provides an ESLOC value for any selected cost object. Different organizations use different formulas to calculate ESLOC. The True Planning calculation for ESLOC is

\[
ESLOC = \text{New Code} + 0.7 \cdot \text{Adapted Code} + 0.1 \cdot \text{Reused Code}.
\]

- **Code Removal Complexity** - This value describes the difficulty of deleting code from the adapted code. Two things need to be considered when deleting code from an application or component: the amount of functionality being removed and how tightly or loosely this functionality is coupled with the rest of the system. Even if a large amount of functionality is being removed, if it accessed through a single point rather than from many points, the complexity of the integration will be reduced.

- **Deleted Code Size** - This describes the amount of pre-existing code that you plan to remove from the adapted code during the software project. The Deleted Code Size value represents code that is included in Adapted Code Size, therefore, it must be less than, or equal to, the Adapted Code Size value.

**SLIM**

SLIM uses effective system size composed of new, modified and reused code. *Modified* software designates changed software and *Reused* is unchanged. The calculations for effective size used in SLIM involve the PI factor to adjust for the relative adaptation effort. If there is reused code than the Productivity Index (PI) factor may be adjusted to add in time and effort for regression testing and integration of the reused code.
A.3.1.4 Generated Software

COCOMO II

Generated software is not designated as a separate entity in COCOMO II. The generator statements, or lines developed as input to the generators, are counted as new lines of code.

SEER

Generated software is not designated as a separate entity in SEER. The generator statements are counted as new lines of code. In maintenance, the generated code may be used instead according to the rules in Section Reference source not found. for maintenance sizing. Function-Based Sizing is also available in SEER to handle generated software.

True Planning

In True Planning, the parameter Auto Gen Size Non-executable represents the percentage of the Auto Generated Code Size that is non-executable (such as, data statements, type declarations, and other non-procedural statements). Typical values for fourth generation languages range from 5% to 30%. If a value cannot be obtained by any other means, the suggested nominal value for non-executable code is 15%. Auto Generated Code Size describes the amount of code generated by an automated design tool for inclusion in a component.

SLIM

Generated code is not treated as a separate category in SLIM. It could be improvised through a user-defined size measure and gearing factor.

A.3.1.5 Automatically Translated Software

COCOMO II

The COCOMO II re-engineering and conversion estimation approach involves estimation of an additional factor, \( AT \), the percentage of the code that is re-engineered by automatic translation. The productivity for automated translation is designated as another factor called \( ATPROD \). Automated translation is considered to be a separate activity from development. Thus, its Adapted SLOC are not included in ESLOC, and its automated translation effort is not included in estimating the projects schedule in COCOMO II.

SEER

Translated code is not covered in SEER.

True Planning

In True Planning, Auto Trans Size Non-executable is the percentage of the Auto Translated Code Size that is non-executable (such as, data statements, type declarations, and other non-procedural statements). Typical values for fourth generation languages range from 5% to 30%. If a value cannot be ob-
Appendix A - Cost Model Descriptions

157

the suggested nominal value for non-executable code is 15%. Auto Translated Code Size describes the amount of code translated from one programming language to another by using an automated translation tool (for inclusion in a component). Auto Translation Tool Efficiency is the percentage of code translation that is actually accomplished by the tool. More efficient auto translation tools require more time to configure the tool to translate. Less efficient tools require more time for code and unit test on code that is not translated.

SLIM
Translated code is not covered in SLIM.

A.3.1.6 Commercial Off-The-Shelf Software (COTS)
The sizing of COTS software sizing is not addressed fully here since COTS source code is not available to be modified nor measured. However, there are instances when size proxies are used for effort related to COTS in some models. COTS can sometimes be handled as reused software or be estimated with other COTS-specific models. Others have more extensive COTS models built in such as True Planning.

COCOMO II
COTS is not directly represented in the COCOMO II model. However, COTS can be treated as reused software or be used in conjunction with the COCOTS model. See further guidance for COTS estimation in [8].

SEER-SEM
COTS Elements available for sizing in SEER include:

- Quick Size
- Application Type Parameter
- Functionality Required Parameter
- Features
- Number of Features Used
- Unique Functions
- Data Tables Referenced
- Data Tables Configured

True Planning
The COTS estimation model in True Planning uses the following:

- Functional Size - Software size in terms of the functional requirements that you expect a Software COTS component to satisfy. When you select Functional Size as the unit of measure (Size Units value) to describe a Software COTS component, the Functional Size value represents a conceptual level size that is based on the functional categories of the
software (such as Mathematical, Data Processing, or Operating System).
A measure of Functional Size can also be specified using Source Lines of
Code, Function Points, Predictive Object Points or Use Case Conversion
Points if one of these is the Size Unit selected.

- **Glue Code Size** - The amount of glue code that will be written. Glue
  Code holds the system together, provides interfaces between Software
  COTS components, interprets return codes, and translates data into
  the proper format. Also, Glue Code may be required to compensate
  for inadequacies or errors in the COTS component selected to deliver
  desired functionality.

To calculate ESLOC for a Software COTS, True Planning first converts
*Functional Size* and *Glue Code Size* inputs to SLOC using a default set of con-
version rates. *New Code* includes *Glue Code Size* and *Functional Size* when
the value of *Amount for Modification* is greater than or equal to 25%. *Adapted
Code* includes *Functional Size* when the value of *Amount for Modification* is
less than 25% and greater than zero. *Reused Code* includes *Functional Size*
when the value of *Amount for Modification* equals zero.

**SLIM**

COTS is not explicitly modeled in SLIM.

### A.3.2 Software Cost Drivers

This section summarizes the mappings, or transformations between cost
factors in different models. With this information estimate inputs can be con-
verted between the models. It also illustrates differences in the models to help
understand why estimates may vary. A top-level Rosetta Stone for common
factors in COCOMO II, SEER-SEM and True Planning factors is shown in
Table A.2.

Most of the mappings between factors are one to one, but some are one to
many (e.g. SEER-SEM has platform factor ratings split into target and host).
In the case of True Planning, many of the COCOMO II factors have direct
corollaries to sub-factors in aggregate True Planning factors. For example the
COCOMO personnel factors are represented as sub-factors under the aggre-
gate True Planning factor for Development Team Complexity. Note there are
additional factors in SEER-SEM and True Planning for which there are no
analogs in COCOMO II.

See [32] for additional details and comparisons of the models. Also refer
to current documentation of the respective cost models for full details and
updates of the cost drivers.
### TABLE A.2: Cost Model Factors

<table>
<thead>
<tr>
<th>Scale Drivers</th>
<th>COCOMO II</th>
<th>SEER-SEM</th>
<th>True Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precedencedness</td>
<td>none</td>
<td>none</td>
<td>none</td>
</tr>
<tr>
<td>Development Flexibility</td>
<td>none</td>
<td>Operating Specification</td>
<td></td>
</tr>
<tr>
<td>Architecture/Risk Resolution</td>
<td>none</td>
<td>none</td>
<td>Development Team Complexity</td>
</tr>
<tr>
<td>Team Cohesion</td>
<td>none</td>
<td>Development Team Complexity</td>
<td></td>
</tr>
<tr>
<td>Process Maturity</td>
<td>none 1</td>
<td>Organization Productivity - CMM Level</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Product Attributes</th>
<th>COCOMO II</th>
<th>SEER-SEM</th>
<th>True Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Required Software Reliability</td>
<td>Specification Level - Reliability</td>
<td>Operating Specification</td>
<td></td>
</tr>
<tr>
<td>Data Base Size</td>
<td>none</td>
<td>Code Size non Executable</td>
<td></td>
</tr>
<tr>
<td>Product Complexity</td>
<td>- Complexity (Staffing)</td>
<td>Functional Complexity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Application Class Complexity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Required Reusability</td>
<td>- Reusability Level Required</td>
<td>Design for Reuse</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Software Impacted by Reuse</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation Match to Lifecycle Needs</td>
<td>none</td>
<td>Operating Specification</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform Attributes</th>
<th>COCOMO II</th>
<th>SEER-SEM</th>
<th>True Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time Constraint</td>
<td>Time Constraints</td>
<td>Project Constraints - Communications and Timing</td>
<td></td>
</tr>
<tr>
<td>Main Storage Constraint</td>
<td>Memory Constraints</td>
<td>Project Constraints - Memory and Performance</td>
<td></td>
</tr>
<tr>
<td>Platform Volatility</td>
<td>- Target System Volatility</td>
<td>Hardware Platform Availability</td>
<td></td>
</tr>
</tbody>
</table>

(continued)
TABLE A.2: Cost Model Factors (Continued)

<table>
<thead>
<tr>
<th>COCOMO II</th>
<th>SEER-SEM</th>
<th>True Planning</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Host System Volatility</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Personnel Attributes

<table>
<thead>
<tr>
<th>Analyst Capability</th>
<th>Analyst Capability</th>
<th>Development Team Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Capability of Analysts and Designers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Programmer Capability</th>
<th>Programmer Capability</th>
<th>Development Team Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Capability of Programmers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Personnel Continuity</th>
<th>none</th>
<th>Development Team Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Team Continuity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application Experience</th>
<th>Application Experience</th>
<th>Development Team Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Familiarity with Platform</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform Experience</th>
<th>- Development System Experience</th>
<th>Development Team Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>- Target System Experience</td>
<td>- Familiarity with Product</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language and Toolset Experience</th>
<th>Programmer’s Language Experience</th>
<th>Development Team Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>- Experience with Language</td>
</tr>
</tbody>
</table>

Project Attributes

<table>
<thead>
<tr>
<th>Use of Software Tools</th>
<th>Software Tool Use</th>
<th>Design Code and Test Tools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multi-site Development</td>
<td>Multiple Site Development</td>
<td>Multi Site Development</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Required Development Schedule</th>
<th>none</th>
<th>Start and End Date</th>
</tr>
</thead>
</table>
A.4 Cost Model Lifecycles and Work Breakdown Structures

The phases covered in the models are summarized in Table 1, activities in Table 2 and their categories of cost sources in Table 3. Note that these are the primary defaults and the models provide customization.

COCOMO II allows effort and schedule to be allocated to either a waterfall or MBASE lifecycle which is an iterative and incremental lifecycle model like the Rational Unified Process (RUP) or the Incremental Commitment Spiral Model (ICSM). In True Planning and SEER-SEM the standard lifecycle activities may be defined differently across development organizations and mapped to other designations. The phase names, activity descriptions and deliverables can also be changed in SLIM.

The tools provide utilities to import or define alternative work breakdown structures and activities that are decomposed by effort and schedule percentages. The SEER-SEM and True Planning tool suites cover other discipline such as systems and hardware engineering, and COCOMO II has a suite of other models covering systems engineering and more.

Future work will update the cost model descriptions and expand them to more comprehensive.
### TABLE A.3: Model Lifecycle Phases

<table>
<thead>
<tr>
<th>Model</th>
<th>Phases</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO II</td>
<td>Inception</td>
</tr>
<tr>
<td></td>
<td>Elaboration</td>
</tr>
<tr>
<td></td>
<td>Construction</td>
</tr>
<tr>
<td></td>
<td>Transition</td>
</tr>
<tr>
<td></td>
<td>Maintenance</td>
</tr>
<tr>
<td>SEER-SEM</td>
<td>System Requirements Design</td>
</tr>
<tr>
<td></td>
<td>Software Requirements Analysis</td>
</tr>
<tr>
<td></td>
<td>Preliminary Design</td>
</tr>
<tr>
<td></td>
<td>Detailed Design</td>
</tr>
<tr>
<td></td>
<td>Code / Unit Test</td>
</tr>
<tr>
<td></td>
<td>Component Integrate and Test</td>
</tr>
<tr>
<td></td>
<td>Program Test</td>
</tr>
<tr>
<td></td>
<td>System Integration Through OT&amp;E</td>
</tr>
<tr>
<td>True Planning</td>
<td>Concept</td>
</tr>
<tr>
<td></td>
<td>System Requirements</td>
</tr>
<tr>
<td></td>
<td>Software Requirements</td>
</tr>
<tr>
<td></td>
<td>Preliminary Design</td>
</tr>
<tr>
<td></td>
<td>Detailed Design</td>
</tr>
<tr>
<td></td>
<td>Code / Unit Test</td>
</tr>
<tr>
<td></td>
<td>Integration and Test</td>
</tr>
<tr>
<td></td>
<td>Hardware / Software Integration</td>
</tr>
<tr>
<td></td>
<td>Field Test</td>
</tr>
<tr>
<td></td>
<td>System Integration and Test</td>
</tr>
<tr>
<td></td>
<td>Maintenance</td>
</tr>
<tr>
<td>SLIM</td>
<td>Concept Definition</td>
</tr>
<tr>
<td></td>
<td>Requirements and Design</td>
</tr>
<tr>
<td></td>
<td>Construction and Testing</td>
</tr>
<tr>
<td></td>
<td>Maintenance</td>
</tr>
</tbody>
</table>
### TABLE A.4: Model Cost Activities

<table>
<thead>
<tr>
<th>Model</th>
<th>Activities</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO II</td>
<td>Management, Environment / CM, Requirements, Design, Implementation, Assessment, Deployment</td>
</tr>
<tr>
<td>SEER-SEM</td>
<td>Management, Software Requirements, Design, Code, Data Programming, Test, CM, QA</td>
</tr>
<tr>
<td>True Planning</td>
<td>Design, Programming, Data, SEPGM, QA, CFM</td>
</tr>
<tr>
<td>SLIM</td>
<td>Concept Definition, Requirements and Design, Construct and Test, Perfective Maintenance</td>
</tr>
</tbody>
</table>
### TABLE A.5: Model Cost Categories

<table>
<thead>
<tr>
<th>Model</th>
<th>Categories</th>
</tr>
</thead>
<tbody>
<tr>
<td>COCOMO II</td>
<td>Software Engineering Labor</td>
</tr>
<tr>
<td>SEER-SEM</td>
<td>Software Engineering Labor</td>
</tr>
<tr>
<td></td>
<td>Purchases</td>
</tr>
<tr>
<td>True Planning</td>
<td>Software Engineering Labor</td>
</tr>
<tr>
<td></td>
<td>Purchased Good</td>
</tr>
<tr>
<td></td>
<td>Purchased Service</td>
</tr>
<tr>
<td></td>
<td>Other Cost</td>
</tr>
<tr>
<td>SLIM</td>
<td>Software Engineering Labor</td>
</tr>
</tbody>
</table>
Appendix B

MIL-STD-881C WBS Mapping to Application Types

B.1 Overview

The Work Breakdown Structures were adapted from MIL-STD-881C to assist in identifying the appropriate Application Type (AT). Each System from 881C is listed with the associated one of more Metrics Manual Operating Environments. Within the environments, look through the Subsystems to find one that matches the component being estimated. Each Subsystem or Sub-Subsystem has a matching AT.

B.n Environment (WBS Level 1) (this will appear as a section heading)

- SubSystem (WBS Level 2)
  - Sub-Subsystem (WBS Level 3)
    * Domain (WBS Level 4) → AT

Use the AT to lookup the associated Model-based CER or Productivity Benchmark for that AT.
B.2 Aerial Vehicle Manned (AVM)

Source: MIL-STD-881C Appendix-A: Aircraft Systems

- Air Vehicle
  - Flight Control Subsystem → VC
  - Auxiliary Power Subsystem → VC
  - Hydraulic Subsystem → VC
  - Electrical Subsystem → VC
  - Crew Station Subsystem → VC
  - Environmental Control Subsystem → VC
  - Fuel Subsystem → VC
  - Landing Gear → VC
  - Rotor Group → VC
  - Drive System → VC

- Avionics
  - Communication / Identification
    * Intercoms → RTE
    * Radio System(S) → RTE
    * Identification Equipment (IFF) → RTE
    * Data Links → RTE
  - Navigation / Guidance
    * Radar → SCP or RTE depending on decomposition
    * Radio → SCP or RTE depending on decomposition
    * Other Essential Nav Equipment → RTE
    * Radar Altimeter → SCP
    * Direction Finding Set → RTE
    * Doppler Compass → RTE
  - Mission Computer / Processing → MP
  - Fire Control
    * Search, Target, Tracking Sensors → SSP
    * Self-Contained Navigation → RTE
    * Self-Contained Air Data Systems → RTE
    * Displays, Scopes, Or Sights → RTE
    * Bombing Computer → MP
    * Safety Devices → RTE
  - Data Display and Controls
    * Multi-Function Displays → RTE
Appendix B - MIL-STD-881C WBS Mapping to Application Types

- Control Display Units → RTE
- Display Processors → MP
- On-Board Mission Planning → RTE if it is embedded

- Survivability
  - Ferret And Search Receivers → RTE
  - Warning Devices → RTE
  - Electronic Countermeasures → RTE
  - Jamming Transmitters → RTE
  - Chaff → RTE
  - Infra-Red Jammers → RTE
  - Terrain-Following Radar → SCP or RTE depending on decomposition

- Reconnaissance
  - Photographic Sensors → SCP
  - Electronic Sensors → SCP
  - Infrared Sensors → SCP
  - Search Receivers → RTE
  - Recorders → RTE
  - Warning Devices → RTE
  - Magazines → RTE
  - Data Link → RTE

- Automatic Flight Control
  - Flight Control Computers → MP
  - Signal Processors → SCP
  - Data Formatting → MP
  - Interfaces To Other Systems → VC
  - Pressure Transducers → RTE
  - Rate Gyros → RTE
  - Accelerometers → RTE
  - Motion Sensors → SCP

- Health Monitoring System → SYS
- Stores Management → VP

B.3 Aerial Vehicle Unmanned (AVU) & Ground Site Fixed (GSF)


- Air Vehicle
- Vehicle Subsystems
  * Propulsion $\rightarrow$ VC
  * Flight Control Subsystem $\rightarrow$ VC
  * Auxiliary Power Subsystem $\rightarrow$ VC
  * Hydraulic Subsystem $\rightarrow$ VC
  * Electrical Subsystem $\rightarrow$ VC
  * Environmental Control $\rightarrow$ VC
  * Subsystem Fuel Subsystem $\rightarrow$ VC
  * Landing Gear $\rightarrow$ VC
  * Rotor Group $\rightarrow$ VC
  * Drive System $\rightarrow$ VC

- Avionics
  * Communication / Identification $\rightarrow$ RTE
  * Navigation / Guidance $\rightarrow$ RTE
  * Automatic Flight Control $\rightarrow$ VC
  * Health Monitoring System $\rightarrow$ SYS
  * Stores Management $\rightarrow$ VP
  * Mission Processing $\rightarrow$ MP
  * Fire Control $\rightarrow$ RTE

* Payload
  - Survivability Payload $\rightarrow$ VP
  - Reconnaissance Payload $\rightarrow$ VP
  - Electronic Warfare Payload $\rightarrow$ VP
  - Armament / Weapons Delivery $\rightarrow$ VP

* Ground / Host Segment (GSF)
  - Ground Control Systems $\rightarrow$ RTE
  - Command and Control Subsystem $\rightarrow$ MP
  - Launch and Recovery Equipment $\rightarrow$ RTE

---

**B.4 Ordnance Vehicle Unmanned (OVU)-Ordnance**

Source: MIL-STD-881C Appendix-D: Ordnance Systems

* Munition
  - Guidance
    * Seeker Assembly $\rightarrow$ SCP
* Guidance Software $\rightarrow$ RTE

- Navigation
  * Sensor Assembly $\rightarrow$ SCP
  * Navigation Software $\rightarrow$ RTE

- Payload
  * Target Defeat Mechanism $\rightarrow$ RTE
  * Target Detection Device $\rightarrow$ RTE
  * Fuze $\rightarrow$ SCP
  * Payload software $\rightarrow$ VP

- Power and Distribution
  * Primary Power $\rightarrow$ VC
  * Power Conditioning Electronics $\rightarrow$ VC
  * Power and distribution software $\rightarrow$ VC

- Communications
  * Antenna Assembly $\rightarrow$ SCP
  * Communications software $\rightarrow$ RTE

- Propulsion Subsystem
  * Motor Engine $\rightarrow$ VC
  * Fuel / Oxidizer Liquid Management $\rightarrow$ VC
  * Arm / Fire Device $\rightarrow$ VC
  * Thrust Vector Actuation $\rightarrow$ VC
  * Flight Termination/Mission Termination $\rightarrow$ RTE
  * Propulsion software $\rightarrow$ VC

- Controls
  * Controls software $\rightarrow$ VC

- Launch System
  - Fire Control $\rightarrow$ RTE

---

**B.5 Ordnance Vehicle Unmanned (OVU)-Missile**

Source: MIL-STD-881C Appendix-C: Missile Systems

- Air Vehicle
– Guidance
  * Seeker Assembly \(\rightarrow\) SCP
  * Guidance Software \(\rightarrow\) RTE
– Navigation
  * Sensor Assembly \(\rightarrow\) SCP
  * Navigation Software \(\rightarrow\) RTE
– Payload
  * Target Defeat Mechanism \(\rightarrow\) RTE
  * Target Detection Device \(\rightarrow\) SCP
  * Fuze \(\rightarrow\) SCP
  * Payload-specific software \(\rightarrow\) VP
– Power and Distribution
  * Primary Power \(\rightarrow\) VC
  * Power Conditioning Electronics \(\rightarrow\) VC
  * Power and distribution software \(\rightarrow\) VC
– Communications
  * Antenna Assembly \(\rightarrow\) SCP
  * Communications software \(\rightarrow\) RTE
– Propulsion Subsystem
  * Motor Engine \(\rightarrow\) VC
  * Thrust Vector Actuation \(\rightarrow\) VC
  * Attitude Control System \(\rightarrow\) VC
  * Fuel / Oxidizer Liquid Management \(\rightarrow\) VC
  * Arm / Fire Device \(\rightarrow\) VC
  * Flight Termination/Mission Termination \(\rightarrow\) RTE
  * Propulsion software \(\rightarrow\) VC
– Controls
  * Controls software \(\rightarrow\) VC
– Reentry System \(\rightarrow\) VC
– Post boost System \(\rightarrow\) VC
– On Board Test Equipment \(\rightarrow\) TST
– On Board Training Equipment \(\rightarrow\) TRN
– Auxiliary Equipment \(\rightarrow\) SYS
– Air Vehicle Software \(\rightarrow\) VC or MP depending on decomposition

• Encasement Device
  – Encasement Device Software \(\rightarrow\) RTE

• Command & Launch
  – Surveillance, Identification, and Tracking Sensors \(\rightarrow\) SCP
Appendix B - MIL-STD-881C WBS Mapping to Application Types

- Launch & Guidance Control →RTE
- Communications →RTE
- Launcher Equipment →RTE
- Auxiliary Equipment →SYS

B.6 Ordnance Vehicle Unmanned (OVU)-Launch Vehicles

Source: MIL-STD-881C Appendix-J: Launch Vehicles

- Launch Vehicle
  - Stage(s)
    * Propulsions System →VC
    * Reaction Control System →VC
    * Recovery System →VC
    * Environmental Control System →VC
    * Stage Peculiar Avionics →RTE
  - Avionics
    * Guidance Navigation and Control →RTE
    * Power →VC
    * Data Acquisition and Telemetry →RTE
    * Range Tracking & Safety (Airborne) →RTE
    * Flight Software →VC

- Flight Operations
  - Real-time mission control
    * Telemetry processing →RTE
    * Communications →RTE
    * Data reduction and analysis →IIS

B.7 Maritime Vessel Manned (MVM)

Source: MIL-STD-881C Appendix-E: Sea Systems

- Ship
  - Command, Communication & Surveillance
B.8 Maritime Vessel Unmanned (MVU) and Maritime Vessel Manned (MVM)

Source: MIL-STD-881C Appendix-I: Unmanned Maritime Vessel Systems

- Maritime Vehicle
  - Energy Storage / Conversion
    * Energy Storage And Conversion Monitoring And Control System → VC
  - Electrical Power
    * Electric Power Monitoring And Control System → VC
  - Vehicle Command and Control
    * Mission Control → RTE
    * Navigation → RTE
    * Guidance And Control → RTE
    * Health Status Monitoring → SYS
    * Rendezvous, Homing And Docking Systems → SYS
    * Fire Control → RTE
  - Surveillance → RTE
Appendix B - MIL-STD-881C WBS Mapping to Application Types

- Communications / Identification → RTE
- Ship Control Systems
  * Hovering And Depth Control → VC
  * Ballast And Trim → VC
  * Maneuvering System → VC
- Auxiliary Systems
  * Emergency Systems → MP
  * Launch And Recovery System → MP
  * Environmental Control System → MP
  * Anchoring, Mooring And Towing → MP
  * Miscellaneous Fluid Systems → MP

- Payload
  - Survivability Payload → VP
  - Intelligence, Surveillance Reconnaissance Payload → VP
  - Armament / Weapons Delivery Payload → VP
  - Mission Payload → VP

- Shipboard Segment (MVM)
  - Shipboard UM Command and Control Subsystem → RTE
  - Shipboard Communication Subsystem → TEL
  - Shipboard Power Subsystem → VC
  - Launch and Recovery Equipment → RTE

B.9 Space Vehicle Manned / Unmanned (SVM/U) & Ground Site Fixed (GSF)

Source: MIL-STD-881C Appendix-F: Space Systems

- Bus
  - Structures & Mechanisms (SMS) → VC
  - Thermal Control (TCS) → VC
  - Electrical Power (EPS) → VC
  - Attitude Control (ACS) → VC
  - Propulsion → VC
  - Telemetry, Tracking, & Command (TT&C) → RTE
  - Bus Flight Software → VC

- Payload
- Thermal Control $\rightarrow$ VP
- Electrical Power $\rightarrow$ VP
- Pointing, Command, & Control Interface $\rightarrow$ VP
- Payload Antenna $\rightarrow$ SCP
- Payload Signal Electronics $\rightarrow$ SCP
- Optical Assembly $\rightarrow$ SCP
- Sensor $\rightarrow$ SCP
- Payload Flight Software $\rightarrow$ VP

- **Ground Operations & Processing Center (GSF)**
  - Mission Management $\rightarrow$ MP
  - Command and Control $\rightarrow$ MP
  - Mission Data Processing $\rightarrow$ MP
  - Mission Data Analysis $\rightarrow$ MP
  - Collection Management $\rightarrow$ MP
  - Infrastructure & Framework $\rightarrow$ SYS

- **Ground Terminal / Gateway (GSF)**
  - Ground Terminal Software
    - Application Specific Integrated Circuit $\rightarrow$ RTE
    - Field Programmable Gate Array $\rightarrow$ RTE

---

### B.10 Ground Site Fixed (GSF)


- **Custom Application Software**
  - Subsystem Software CSCI $\rightarrow$ depends on decomposition

- **Enterprise Service Element**
  - Software COTS / GOTS
    - Component identification $\rightarrow$ IIS
    - Assessment and Selection $\rightarrow$ IIS
    - Prototyping $\rightarrow$ IIS
    - Glue code development $\rightarrow$ IIS
    - Tailoring and configuration $\rightarrow$ IIS

- **Enterprise Information System**
B.11 Ground Vehicle Manned & Unmanned (GVM/U)

Source: MIL-STD-881C Appendix-G: Surface Vehicle Systems

- Primary Vehicle
  - System Survivability → RTE
  - Turret Assembly → RTE
  - Suspension / Steering → RTE
  - Vehicle Electronics
    - Computers And Other Devices For Command And Control → RTE
    - Data Control And Distribution → MP
    - Controls And Displays → RTE
    - Power Distribution And Management → VC
    - Health Management Systems → VC
  - Power Package / Drive Train
    - Controls And Instrumentation → VC
    - Power Transmission, Final Drivers, And Power Takeoffs → VC
    - Brakes And Steering When Integral To Power Transmission → VC
    - Hybrid Electric Drive Systems → VC
    - Energy Storage Systems → VC
  - Fire Control
    - Radars And Other Sensors → SCP
    - Controls And Displays → RTE
    - Sights Or Scopes → RTE
    - Range Finders, Gun Drives And Stabilization Systems → VP
  - Armament
    - Main Gun And Secondary Guns → VP
    - Missile Launchers → VP
    - Non-Lethal Weapons → VP
* Other Offensive Weapon Systems → VP
  - Automatic Ammunition Handling → MP
  - Navigation and Remote Piloting → RTE
  - Communications → RTE

- Remote Control System (GVU specific)
  - Ground Control Systems → RTE
  - Command and Control Subsystem → MP
  - Remote Control System Software → RTE

B.12 Applies to ALL Environments

Source: MIL-STD-881C Appendix-L: Common Elements

- System Integration Lab (SIL)
  - SIL Software - SIL Operations → TST
  - SIL Software - Simulation → SCI

- Test and Evaluation Support
  - Test Software → TST

- Automated Test Equipment
  - Equipment Software → TST

- Training
  - Equipment → TRN
  - Simulators → SCI
  - Computer Based-Application → IIS
  - Computer Based-Web → IIS

- Support Equipment
  - Software → SYS

- Test and Measurement Equipment
  - Equipment Software → TST

- Data Migration
  - Software Utilities → SYS
Appendix C

CER Nomograms

C.1 Overview

Nomograms can be used for rapid, graphical computation of cost models without computers or manual calculations. They are the most concise representation to visualize and quantify CER model relationships. They provide instant results when approximate answers are appropriate and useful. The graphs can be also used as crosschecks of other estimation methods and tools, or help flag their erroneous inputs. A couple examples for the types shown in this appendix are illustrated.

An example of using a simple nomogram is demonstrated in Figure C.1 for the Real Time Embedded - Aerial Vehicle CER. The corresponding effort for any size on the left axis is adjacent on the right side.

\[ PM = 15.09 \cdot KESLOC^{0.81} \]

FIGURE C.1: Simple CER Nomogram Sensitivity Analysis
An estimate for 100 KESLOC is 630 Person-Months per Figure C.1. A sensitivity analysis that varies size is displayed as the colored range. E.g., for the range of size between 90-130 KESLOC the corresponding effort range is 530-780 Person-Months. This is a form of uncertainty analysis.

A more complex CER is shown in Figure C.2 for the COCOMO cost model that incorporates the Effort Adjustment Factor (EAF) (see Appendix A). With this type of nomogram an isopleth line is drawn connecting values on any two axes to compute the third variable. The example shown indicates the COCOMO estimate for 100 KSLOC and EAF of 1 corresponds to 465 Person-Months. Sensitivity analysis and tradeoffs can be visualized by moving the calculation line between values.

FIGURE C.2: COCOMO Nomogram Estimate

C.2 Operating Environment CERs
$PM = 9.41 \cdot KESLOC^{0.93}$

FIGURE C.3: Aerial Vehicle - CER Nomogram
$PM = 12.6 \cdot KESLOC^{0.79}$

FIGURE C.4: Ground Site - CER Nomogram
\[ PM = 15.1 \cdot KESLOC^{0.82} \]

**FIGURE C.5: Ground Vehicle - CER Nomogram**
\[ PM = 5.44 \cdot KESLOC^{1.12} \]

**FIGURE C.6: Maritime Vessel - CER Nomogram**
$PM = 27.45 \cdot KESLOC^{0.71}$

FIGURE C.7: Ordnance Vehicle - CER Nomogram
C.3 CERs by Application Type within Operating Environments

C.3.1 Ground Site Operating Environment
$PM = 6.66 \cdot KESLOC^{1.01}$

**FIGURE C.8**: Ground Site - Command and Control (C&C) CER Nomogram
\[ PM = 2.57 \cdot KESLOC^{1.03} \]

FIGURE C.9: Ground Site - Custom AIS Software (CAS) CER Nomogram
PM = 10.80 \cdot KESLOC^{0.83}

FIGURE C.10: Ground Site - Communications (COM) CER Nomogram
\[ PM = 6.14 \cdot KESLOC^{0.86} \]

FIGURE C.11: Ground Site - Mission Planning (MP) CER Nomogram
$PM = 8.71 \cdot KESLOC^{0.92}$

FIGURE C.12: Ground Site - Real Time Embedded (RTE) CER Nomogram
**PM = 31.88 · KESLOC^{0.79}**

**FIGURE C.13**: Ground Site - Sensor Control and Signal Processing (SCP) CER Nomogram
\[ PM = 7.93 \cdot KESLOC^{0.89} \]

FIGURE C.14: Ground Site - System Software (SYS) CER Nomogram
C.3.2 Ground Vehicle Operating Environment
$$PM = 7.81 \cdot KESLOC^{1.1}$$

FIGURE C.15: Ground Vehicle - Vehicle Control (VC) CER Nomogram
C.3.3 Aerial Vehicle Operating Environment
$$PM = 5.61 \cdot KESLOC^{1.16}$$

**FIGURE C.16**: Aerial Vehicle - Real Time Embedded (RTE) CER Nomogram
$PM = 28.26 \cdot KESLOC^{0.86}$

**FIGURE C.17:** Aerial Vehicle - Sensor Control and Signal Processing (SCP) CER Nomogram
Appendix C  - CER Nomograms

**PM = 8.09 \cdot KESLOC^{1.05}**

FIGURE C.18: Aerial Vehicle - Vehicle Control (VC) CER Nomogram
\[ PM = 13.38 \cdot KESLOC^{0.91} \]

FIGURE C.19: Aerial Vehicle - Vehicle Payload (VP) CER Nomogram
C.3.4 Maritime Vessel Operating Environment
$PM = 5.77 \cdot KESLOC^{0.96}$

FIGURE C.20: Maritime Vessel - Communications (COM) CER Nomogram
C.4 CERs by AppType Across All Environments
\[ PM = 6.6 \cdot KESLOC^{1.05} \]

FIGURE C.21: Command and Control (C&C) CER Nomogram
$PM = 7.3 \cdot KESLOC^{0.91}$

FIGURE C.22: Communications (COM) CER Nomogram
PM = 2.64 · KESLOC^{1.02}

FIGURE C.23: Custom AIS Software (CAS) CER Nomogram
\[ PM = 26.5 \cdot KESLOC^{0.87} \]

**FIGURE C.24**: Sensor Control and Signal Processing (SCP) CER Nomogram
PM = 7.43 \cdot KESLOC^{0.91}

FIGURE C.25: Scientific and Simulation (S&S) CER Nomogram
PM = 6.14 \cdot KESLOC^{0.91}

FIGURE C.26: Mission Processing (MP) CER Nomogram
PM = $13.1 \cdot KESLOC^{0.84}$

FIGURE C.27: Real Time Embedded (RTE) CER Nomogram
PM = 5.06 \cdot KESLOC^{0.98}

FIGURE C.28: System Software (SYS) CER Nomogram
PM = 7.42 \cdot KESLOC^{1.0}

**FIGURE C.29**: Test, Measurement, and Diagnostic Equipment (TMDE) CER Nomogram
Appendix C - CER Nomograms

$PM = 9.05 \cdot KESLOC^{1.02}$

FIGURE C.30: Vehicle Control (VC) CER Nomogram
\[ PM = 22.27 \cdot KESLOC^{0.81} \]

FIGURE C.31: Vehicle Payload (VP) CER Nomogram
C.5 Cost Model Nomograms

C.5.1 COCOMO
\[ PM = 2.94 \cdot KESLOC^{1.0997} \]

FIGURE C.32: COCOMO Nominal Nomogram
\[ PM = 2.94 \times KSLOC^{1.0997} \times EAF \]

**FIGURE C.33: COCOMO with EAF Nomogram**
Appendix D

Unified CodeCount

D.1 Overview

Unified CodeCount (UCC) is a source code counting and differencing tool [41]. It allows the user to count, compare, and collect both physical and logical differentials between two versions of the source code of a software product. The differencing capabilities allow users to count the number of added/new, deleted, modified, and unmodified physical and logical source lines of code (SLOC) of the current version in comparison with the previous version. With the counting capabilities, users can generate the physical, logical SLOC counts, and other sizing information, such as comment and keyword counts, of the target program.

The tool can be compiled using a C/C++ supported compiler. It is run by providing files of filenames to be counted or providing the directories where the files reside. It can be downloaded from http://csse.usc.edu/research/CODECOUNT/.

Sample output is shown below in Figure D.34.

<table>
<thead>
<tr>
<th>RESULTS SUMMARY</th>
<th>Total</th>
<th>Blank</th>
<th>Comments</th>
<th>Compiler</th>
<th>Data</th>
<th>Exec.</th>
<th>Number of Files</th>
<th>SLOC</th>
<th>Type</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines</td>
<td>Lines</td>
<td>Whole</td>
<td>Embedded</td>
<td>Direct.</td>
<td>Decl.</td>
<td>Instr.</td>
<td>of Files</td>
<td>SLOC</td>
<td>Type</td>
<td>Definition</td>
</tr>
<tr>
<td>11833</td>
<td>1140</td>
<td>1620</td>
<td>328</td>
<td>60</td>
<td>496</td>
<td>8517</td>
<td>22</td>
<td>9073</td>
<td>CODE</td>
<td>Physical</td>
</tr>
<tr>
<td>11833</td>
<td>1140</td>
<td>1620</td>
<td>328</td>
<td>60</td>
<td>496</td>
<td>6434</td>
<td>22</td>
<td>6983</td>
<td>CODE</td>
<td>Logical</td>
</tr>
</tbody>
</table>

FIGURE D.34: Unified CodeCount Example Summary Output

The SLOC count in this summary is the total of 6983 Logical SLOC consisting of 496 data declarations and 6434 executable instructions. This is the software size to be used for estimation, measurement of actuals, and model calibration.
## Appendix E

### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GL</td>
<td>Fourth Generation Language</td>
</tr>
<tr>
<td>AAF</td>
<td>Adaptation Adjustment Factor</td>
</tr>
<tr>
<td>AAM</td>
<td>Adaptation Adjustment Multiplier</td>
</tr>
<tr>
<td>ACAT</td>
<td>Acquisition Category</td>
</tr>
<tr>
<td>ASP</td>
<td>Acquisition Support Plan</td>
</tr>
<tr>
<td>AV</td>
<td>Aerial Vehicle</td>
</tr>
<tr>
<td>AVM</td>
<td>Aerial Vehicle Manned</td>
</tr>
<tr>
<td>AVU</td>
<td>Aerial Vehicle Unmanned</td>
</tr>
<tr>
<td>CARD</td>
<td>Cost Analysis Requirements Document</td>
</tr>
<tr>
<td>CER</td>
<td>Cost Estimating Relationship</td>
</tr>
<tr>
<td>CM</td>
<td>Code Modified Percentage</td>
</tr>
<tr>
<td>CMM</td>
<td>Capability Maturity Model</td>
</tr>
<tr>
<td>COCOMO</td>
<td>COnstructive COst MOdel</td>
</tr>
<tr>
<td>COCOTS</td>
<td>COnstructive Commercial-off-the Shelf</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-off-the Shelf</td>
</tr>
<tr>
<td>CSCI</td>
<td>Computer Software Configuration Item</td>
</tr>
<tr>
<td>CSDR</td>
<td>Cost and Software Data Report</td>
</tr>
<tr>
<td>DACIMS</td>
<td>Defense Automated Cost Information Management System</td>
</tr>
<tr>
<td>DAMS</td>
<td>Defense Acquisition Management System</td>
</tr>
<tr>
<td>DCARC</td>
<td>Defense Cost and Resource Center</td>
</tr>
<tr>
<td>DM</td>
<td>Design Modified Percentage</td>
</tr>
<tr>
<td>DoD</td>
<td>Department of Defense</td>
</tr>
</tbody>
</table>
EI  External Inputs
EIF  External Interfaces
EO  External Outputs
EQ  External Inquiries
EKSLOC  Equivalent Thousands of Source Lines of Code
ESLOC  Equivalent Source Lines of Code
FPA  Function Point Analysis
FPC  Function Point Count
GAO  U.S. General Accounting Office
GS  Ground Site
GSF  Ground Site Fixed
GSM  Ground Site Mobile
GUI  Graphical User Interface
GV  Ground Vehicle
GVM  Ground Vehicle Manned
GVU  Ground Vehicle Unmanned, e.g., Robot vehicles
HOL  Higher Order Language
HWCI  Hardware Configuration item
IDPD  Incremental Development Productivity Decline
IFPUG  International Function Point User’s Group
IIS  Intelligence and Information Software
ILF  Internal Files
IM  Integration Modified Percentage
IRS  Interface Requirement Specification
IS  Information System
KESLOC  Thousands of Equivalent Source Lines of Code
KSLOC  Thousands of Source Lines of Code
LCC  Life Cycle Cost
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDAP</td>
<td>Major Defense Acquisition Program</td>
</tr>
<tr>
<td>MP</td>
<td>Mission Processing</td>
</tr>
<tr>
<td>MSLOC</td>
<td>Millions of source lines of code</td>
</tr>
<tr>
<td>MV</td>
<td>Maritime Vessel</td>
</tr>
<tr>
<td>MVM</td>
<td>Maritime Vessel Manned</td>
</tr>
<tr>
<td>MVU</td>
<td>Maritime Vessel Unmanned</td>
</tr>
<tr>
<td>NASA</td>
<td>National Aeronautics and Space Administration</td>
</tr>
<tr>
<td>NDI</td>
<td>Non-Development Item</td>
</tr>
<tr>
<td>NLM</td>
<td>Non-Linear Model</td>
</tr>
<tr>
<td>OLS</td>
<td>Ordinary Least Squares regression</td>
</tr>
<tr>
<td>OO</td>
<td>Object Oriented</td>
</tr>
<tr>
<td>OpEnv</td>
<td>Operating Environment</td>
</tr>
<tr>
<td>OSD</td>
<td>Office of the Secretary of Defense</td>
</tr>
<tr>
<td>OV</td>
<td>Ordnance Vehicle</td>
</tr>
<tr>
<td>OVU</td>
<td>Ordnance Vehicle Unmanned</td>
</tr>
<tr>
<td>PC</td>
<td>Process Control</td>
</tr>
<tr>
<td>PLN</td>
<td>Planning software</td>
</tr>
<tr>
<td>Pr</td>
<td>Productivity</td>
</tr>
<tr>
<td>RTE</td>
<td>Real-Time Embedded</td>
</tr>
<tr>
<td>RUP</td>
<td>Rational Unified Process</td>
</tr>
<tr>
<td>SCI</td>
<td>Scientific Software</td>
</tr>
<tr>
<td>SCP</td>
<td>Sensor Control and Signal Processing (SCP)</td>
</tr>
<tr>
<td>SDD</td>
<td>Software Design Document</td>
</tr>
<tr>
<td>SDP</td>
<td>Software Development Plan</td>
</tr>
<tr>
<td>SDR</td>
<td>Software Design Review</td>
</tr>
<tr>
<td>SEER-SEM</td>
<td>System Evaluation and Estimation of Resources Software Estimating Model (software cost estimation tool)</td>
</tr>
<tr>
<td>SEI</td>
<td>Software Engineering Institute</td>
</tr>
</tbody>
</table>

SER  Schedule Estimating Relationship
SLIM  Software Lifecycle Management (software cost estimation tool)
SLOC  Source Lines of Code
SRDR  Software Resource Data Report
SRR  Systems Requirements Review
SRS  Software Requirements Specification
SSR  Software Specification Review
SSS  System Segment Specification
SU  Software Understanding
SV  Space Vehicle
SVM  Space Vehicle Manned
SVU  Space Vehicle Unmanned
SYS  System Software
TEL  Telecommunications software
TOOL  Software Tools
TST  Test software
TRN  Training Software
UCC  Universal Code Counter
UNFM  Programmer Unfamiliarity
USC  University of Southern California
VC  Vehicle Control
VP  Vehicle Payload
WBS  Work Breakdown Structure
Bibliography


## Index

<table>
<thead>
<tr>
<th>Term</th>
<th>Page Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAF, 25, 26, 41–42</td>
<td></td>
</tr>
<tr>
<td>Activities, 16, 17, 29–30, 134, 144</td>
<td></td>
</tr>
<tr>
<td>Adaptation Adjustment Factor, see AAF</td>
<td></td>
</tr>
<tr>
<td>Adapted software, 20, 23, 25–26, see also Modified software and Reused software</td>
<td></td>
</tr>
<tr>
<td>Aerial Vehicle (AV), 47, 56, 71, 184, 194</td>
<td></td>
</tr>
<tr>
<td>Application Type, 4, 5, 7, 13, 48–56, 128</td>
<td></td>
</tr>
<tr>
<td>Auto-generated software, 15, 35, 36, 42</td>
<td></td>
</tr>
<tr>
<td>Blank lines, 22</td>
<td></td>
</tr>
<tr>
<td>Carryover code, 15</td>
<td></td>
</tr>
<tr>
<td>CER, vii, 1–7, 33, 44–109, 135</td>
<td></td>
</tr>
<tr>
<td>COCOMO, 116, 117, 119, 146, 147, 149, 152, 153, 156–158, 161–164, 213</td>
<td></td>
</tr>
<tr>
<td>Code counting, 7, 15, 111</td>
<td></td>
</tr>
<tr>
<td>Code modified (CM), 25, 41–42, 112</td>
<td></td>
</tr>
<tr>
<td>Command and Control (C&amp;C), 50, 201</td>
<td></td>
</tr>
<tr>
<td>Command and Control (CC), 77</td>
<td></td>
</tr>
<tr>
<td>Comments, 22</td>
<td></td>
</tr>
<tr>
<td>Communication (COM), 79</td>
<td></td>
</tr>
<tr>
<td>Communications (COM), 50, 201</td>
<td></td>
</tr>
<tr>
<td>Compiler directives, 22, 23</td>
<td></td>
</tr>
<tr>
<td>Converted software, 20, 22, 25</td>
<td></td>
</tr>
<tr>
<td>Cost, 1, 4, 5</td>
<td></td>
</tr>
<tr>
<td>Cost Estimating Relationships, see CER</td>
<td></td>
</tr>
<tr>
<td>Cost estimation process, 3, 7</td>
<td></td>
</tr>
<tr>
<td>Cost model, 3, 5, 7, 145–161, 213</td>
<td></td>
</tr>
<tr>
<td>COTS, 13, 20, 23, 25, 114, 115, 123, 131, 132</td>
<td></td>
</tr>
<tr>
<td>CSDR, 9, 17</td>
<td></td>
</tr>
<tr>
<td>Custom AIS Software (CAS), 53, 78, 201</td>
<td></td>
</tr>
<tr>
<td>DACIMS, 2, 4, 10</td>
<td></td>
</tr>
<tr>
<td>DAMS, 2, 3</td>
<td></td>
</tr>
<tr>
<td>Data assessment, 6, 32–43</td>
<td></td>
</tr>
<tr>
<td>Data Dictionary, 11, 16, 34, 38</td>
<td></td>
</tr>
<tr>
<td>Data normalization, 1, 5, 134, 136, 144</td>
<td></td>
</tr>
<tr>
<td>Data quality, 33, 36–38</td>
<td></td>
</tr>
<tr>
<td>Data segmentation, 56</td>
<td></td>
</tr>
<tr>
<td>DCARC, 9, 17</td>
<td></td>
</tr>
<tr>
<td>Declarations, 21, 22</td>
<td></td>
</tr>
<tr>
<td>Defects, 16</td>
<td></td>
</tr>
<tr>
<td>Deleted software, 15</td>
<td></td>
</tr>
<tr>
<td>Design modified (DM), 25, 41–42, 112</td>
<td></td>
</tr>
<tr>
<td>DoD, vii, 1–4, 10–11, 17, 126</td>
<td></td>
</tr>
<tr>
<td>Duration, 31, 33, 34, 134–136, 144, see also Schedule</td>
<td></td>
</tr>
<tr>
<td>Effort, 1, 4, 6, 15, 29, 33, 35, 36, 38, 39, 134–136, 144–150, 153–157, 161</td>
<td></td>
</tr>
<tr>
<td>Enterprise Information Systems (EIS), 54</td>
<td></td>
</tr>
<tr>
<td>Enterprise Service Systems (ESS), 53</td>
<td></td>
</tr>
<tr>
<td>Equivalent size, 4, 13, 20, 23–26, 33, 35, 37, 39, 40, 42, 112, 117, 149, 152, 153, 155, see also ESLOC</td>
<td></td>
</tr>
<tr>
<td>Equivalent Source Lines of Code, see ESLOC</td>
<td></td>
</tr>
<tr>
<td>ESLOC, 24, 36, 112, 113</td>
<td></td>
</tr>
<tr>
<td>Executables, 21, 22</td>
<td></td>
</tr>
</tbody>
</table>
Experience, 16
External interface requirements, 14

Function Points, 26–28
GAO, 7, 125, 141
Generated software, 20, 22, 25
GFS, 20, 23, see also COTS
GOTS, 13, 132
Ground Site (GS), 47, 56, 72, 184
Ground Vehicle (GV), 47, 73, 184, 192
Integration required (IM), 25, 41–42, 112
KESLOC, 4

Labor categories, 30, 161
Labor hours, 31, 35, 134
Labor rate, 1, 134, 136
Lifecycle phases, see Phases
Linear regression, 58, 60, 63–69
Logical source statements, 19, 22–23, 37–39, 134, see also SLOC
Maritime Vessel (MV), 47, 56, 74, 184, 199
MDAP, 9
Metrics definitions, 5, 6, 19–31
Microcode and Firmware (M&F), 48
MIL-STD-881C, 7, 54–56
Mission Planning (MP), 53, 80
Mission Processing (MP), 201
Modified software, 20, 22, 25, 33, 35, 36, 41
NCSS, 21–23, 37, 39
New software, 13, 20, 22, 23, 25, 26, 41
Non-Commented Source Statements, see NCSS
Nonexecutables, 22
Normalization, see Data normalization, 33, 38, 134
Operating Environment, 1, 2, 7, 56, 70, 128
Ordnance Vehicle (OV), 47, 56, 75, 184
Person-Months, 1, 4, see also Effort, 113
Personnel experience, 14
Personnel experience, 150
Phases, 29–30, 132
Physical lines, 21–23, 37
Process Control (PC), 51
Product quality, see Quality
Productivity, 36, 102–108
Productivity benchmarks, vii, 1, 5, 6, 33, 102–108, 135
Programming language, 13, 15
Quality, 16, 17, 33
Real Time Embedded (RTE), 50, 81, 201
Requirements, 14, 17
Requirements stability, see Requirements volatility
Requirements volatility, 14, 112, 121, 134
Response variable, 45, 56, 60, 63–66
Reused software, 13, 15, 20, 22, 25, 35, 36, 42, 112
Schedule, 15, 31, 33, 36, see also Duration, 132, 138
Scientific and Simulation (S&S), 51, 82, 201
SEER-SEM, 116, 146–149, 157, 158, 161–164
Sensor Control and Signal Processing (SCP), 49, 83, 201
Size, 4–6, 14, 16, 19, 33, 35, 36, 38, 134, 137, 144
adapted, 19
equivalent, 4, 13, 20, 23–26, 33, 35, 37, 39, 40, 42, 112, 117, 149, 152, 153, 155
SLIM, 146, 148, 150, 152, 155–158, 161–164
Index

SLOC, 4, 19–23, 33, 34, 38, 111, 113, 115, 117, 134
Software activities, see Activities
Software cost, see Cost
Software process maturity, 13
Software requirements, see Requirements
Software Resources Data Report, see SRDR
Software reuse, see Reused software
Software size, see Size
Software Tools (TOOL), 52
Software volatility, 25, 26
Source Lines of Code, see SLOC
Space Vehicle (SV), 47, 56
SRDR, vii, 1–7, 9–17, 29, 34, 38, 112, 134
Staffing, 13, 16, 33
System Software (SYS), 51, 84, 201
Test, Measurement, and Diagnostic Equipment (TMDE), 52, 85, 201
Total equivalent size, 26
Total lines, 21–23, 37–39
Training (TRN), 52
True Planning, 146, 147, 150, 152, 154–158, 161–164
Unified Code Count, 25, 39
Vehicle Control (VC), 49, 86, 201
Vehicle Payload (VP), 49, 87, 201
WBS, 7, 12, 15, 132
Work Breakdown Structure, see WBS