SYM-AM-18-058

PROCEEDINGS
OF THE
FIFTEENTH ANNUAL
ACQUISITION RESEARCH
SYMPOSIUM

WEDNESDAY SESSIONS
VOLUME I

Acquisition Research:
Creating Synergy for Informed Change

May 9–10, 2018

Published April 30, 2018

ACQUISITION RESEARCH PROGRAM
GRADUATE SCHOOL OF BUSINESS & PUBLIC POLICY
NAVAL POSTGRADUATE SCHOOL

Exploring the Department of Defense Software Factbook

Christopher Miller—is a Senior Researcher at the Software Engineering Institute (SEI). Dr. Miller's expertise is in software metrics, measurement, and estimation of software intensive systems. His quantitative analysis background is focused on life cycle cost estimation, evaluating project feasibility measures, establishing performance measurements, and providing analysis for the optimization of systems. Dr. Miller served as the Chair of the International Council on Systems Engineering (INCOSE) Measurement Working Group (MWG) for seven years and is a certified trainer for Practical Software and Systems Measurement (PSM). Dr. Miller earned a Masters of Engineering Management and PhD in Systems Engineering at the George Washington University and is currently a member of the adjunct faculty in the School of Engineering and Applied Science (SEAS). [clmiller@sei.cmu.edu]

Forrest Shull—is Assistant Director for Empirical Research at Carnegie Mellon University's Software Engineering Institute. His role is to lead work with the U.S. Department of Defense, other government agencies, national labs, industry, and academic institutions to advance the use of empirically grounded information in software engineering, cybersecurity, and emerging technologies. He has been a lead researcher on projects for the U.S. Department of Defense, NASA's Office of Safety and Mission Assurance, the Defense Advanced Research Projects Agency (DARPA), the National Science Foundation, and commercial companies. He serves on the IEEE Computer Society Board of Governors and Executive Committee. [fjshull@sei.cmu.edu]

David Zubrow—is an Associate Director for the Software Solutions Division and Manager of the Software Engineering Measurement and Analysis Initiative. He has been employed at the Software Engineering Institute (SEI) at Carnegie Mellon University since 1992. For much of his 26 years at the SEI, he has developed and transitioned quality and process improvement principles and practices directly through the CMMI and applications of statistical analysis. More recently, his research and transition projects have involved using machine learning methods in conjunction with software engineering data to model program status assessments and risks as well as predicting defects. [dz@sei.cmu.edu]

Abstract

The Carnegie Mellon Software Engineering Institute (SEI) conducted an analysis of software engineering data owned and maintained by the Department of Defense (DoD) to produce high-level, DoD-wide heuristics and domain-specific benchmark data. This work yielded basic facts about software projects, such as averages, ranges, and heuristics for requirements, size, effort, and duration. Factual, quantitatively-derived statements were reported to provide users with easily digestible benchmarks.

Findings were also presented by system type, or super domain. The analysis in this area focused on identifying the most and least expensive projects and the best and worst projects within three super domains: real time, engineering, and automated information systems. It also provided insight into the differences between system domains and contained domain-specific heuristics.

Finally, correlations were explored among requirements, size, duration, and effort and the strongest models for predicting change were described. The goal of this work was to determine how well the data could be used to answer common questions related to planning or replanning software projects. The paper provides a high-level overview of the SEI's research and primary findings.

Introduction

In 2015 the SEI undertook an analysis of the most extensive collection of software engineering data owned and maintained by its primary sponsor, the Department of Defense (DoD). The resulting *Department of Defense Software Factbook* provides an analysis of the software resources data report (SRDR), a contract data deliverable that applies to major contracts and subcontracts for programs with software development elements that include a projected software effort greater than $20 million.[1]

The SRDR is a contract data deliverable that formalized the reporting of software metrics data and is the primary source of data on software projects and their performance. The SRDR reports are provided at the project level or subsystem level, not at the DoD Acquisition Program level. **It is important to note that when the analysis refer to a "project" in this report, a project is synonymous with a software build, increment, or release**. In many cases, several projects (i.e., data points) would contribute to the overall scope and make up of Acquisition Program (i.e., an entire weapon system).

This work builds on a field of research begun in the 1970s into how to estimate the cost of software development. An entire industry focused on parametric software estimation has grown since that time, and at the core of this industry is a fundamental assumption that the cost of developing software can be estimated based on an accurate estimate of the size of the software product to be developed. This concept might be more accurately described as an assumed empirical relationship between cost and software size.

A new SRDR Data Item Description (DID), DI-MGMT-82035A, with updated formats for software development and maintenance was approved for release in November 2017. This new DID replaces the 2016 version of the DID which superseded the 2011 Initial and Final SRDR DIDs. The SRDR DID remains at $20 million or over for all new contracts, and over $1 million per year maintenance efforts. Key parameters related to software cost include functional size (in requirements), physical size (in equivalent source lines of code), effort hours, and duration of software projects. In most DoD environments, size is measured by requirements and the final physical size of the software product, commonly measured in source lines of code (SLOC). The amount of effort required to deliver the software can be estimated if you know the size. Similarly, duration (or schedule) can be derived from the effort.

The majority of the SRDR data used in this analysis is based on the final report that contains data about actual results. Projects used for this analysis had to include the following information: size data (functional and product), effort data, and schedule data. Our analysis included 287 projects from the product-event final report data and 181 pairs of initial and final case data (to compare estimated versus actual performance).

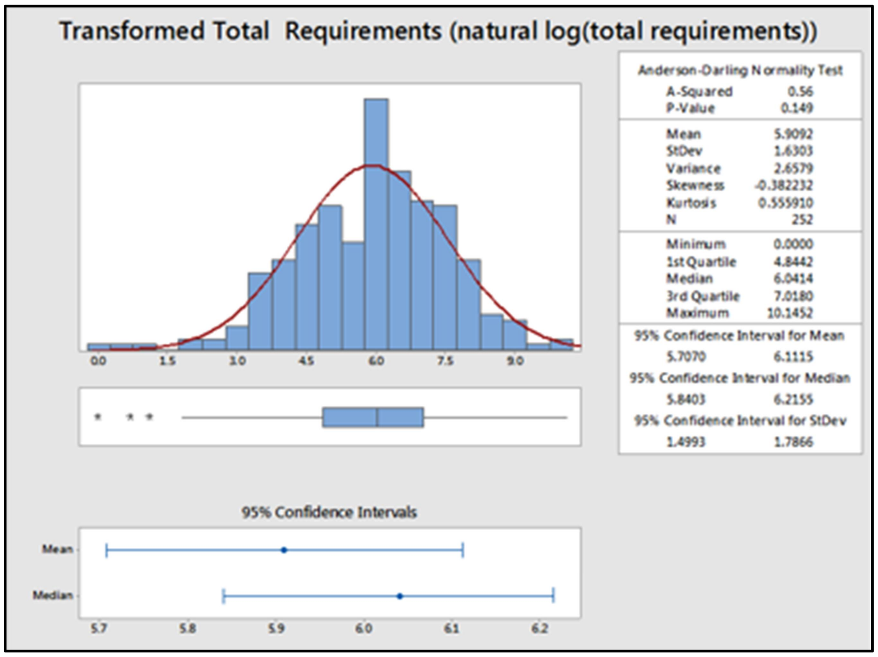
[1] For a more detailed description of programs types that require the use of the SRDR, see the *Department of Defense Software Factbook*, or CSDR Requirements, OSD Defense Cost and Resource Center, http://dcarc.cape.osd.mil/CSDR/CSDROverview.aspx#Introduction.

Functional Size (Requirements)

Functional size represents the overall magnitude of the software capabilities without regard to the final solution. The benefit of using functional measures is their availability early in the software development lifecycle. In the DoD acquisition community, requirements are rigorously derived and used as the contractual basis for acquiring systems. Therefore requirements and requirements documents are produced as part of the system acquisition life cycle and are readily available for the extraction of the number of requirements.

The drawback of using functional measures is that the requirement does not consistently correlate to a unit of effort (i.e., not all requirements take the same amount of effort to satisfy). Using the total number of requirements to represent size is useful, but trying to attach a unit cost (i.e., the cost per requirement) is not advised.

In general, software project data tends to be skewed. So making a transformation to get it into a normal (Gaussian) distribution is usually necessary. This was necessary for the SRDR requirements data. Since it was quite skewed, with the bulk of the data between 102 (~100) and 1110 (~1100) requirements, it needed to be transformed. Once the data was normalized using a natural log transformation, the median is e6.04, or 420 requirements with a mean of 368 requirements. Both are much closer to the raw data median of 399 than the raw data mean of 1118 requirements (see Figure 1).



Anderson-Darling Normality Test	
A-Squared	0.56
P-Value	0.149
Mean	5.9092
StDev	1.6303
Variance	2.6579
Skewness	-0.382232
Kurtosis	0.555910
N	252
Minimum	0.0000
1st Quartile	4.8442
Median	6.0414
3rd Quartile	7.0180
Maximum	10.1452
95% Confidence Interval for Mean	
5.7070	6.1115
95% Confidence Interval for Median	
5.8403	6.2155
95% Confidence Interval for StDev	
1.4993	1.7866

Figure 1. Functional Size, Normalized

Requirements data analyzed by super domain are presented in Figure 2. As is in shown on the top of the figure, to the left of the line is the 25th percentile value. This indicates that 25% of the projects have less than 100 requirements. Similarly, on the right the 75th percentile value indicates that 25% of the projects have more than 1100 requirements. Note that 50% of the projects have between 100 and 1100 requirements, with relatively more toward the lower end and a median or typical view of 400. The additional lines in the figure can be similarly interpreted. Similar figures are provided throughout this paper showing the 25th percentile, median, and 75th percentiles. An easy heuristic for the average functional size of a DoD software project is **400 requirements**.

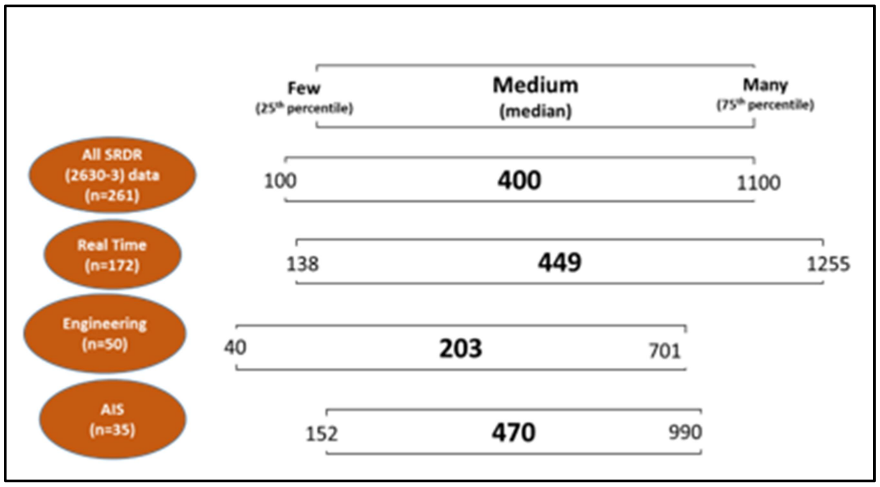


Figure 2. Requirements Data by Super Domain

Product Size (ESLOC)

Another common measure of interest is product size, which is often measured in source lines of code (SLOC). A key issue in using SLOC as a measure of work effort and duration is the difference in work required to incorporate software from different sources, including new code, modified code (changed in some way to make it suitable), reused code (used without changes), and auto-generated code (created from a tool and used without change).

Each of these sources requires a different amount of work effort to incorporate into a software product. The challenge is in coming up with a single measure that includes all of the code sources. The approach taken here is to normalize all code sources to the *equivalent* of a new line of code. This is done by taking a portion of the measures for modified, reused, and auto-generated code. The portioning is based on the percentage of modification to the code based on changes to the design, code and unit test, and integration and test documents. This approach is adopted from the COCOMO II Software Cost Estimation Model (Boehm et al., 2000, p. 22).

Equivalent source lines of code (ESLOC), then, is the homogeneous sum of the different code sources. The portion of each code source is determined using a formula called an Adaptation Adjustment Factor (AAF):

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

Where

%DM: Percentage Design Modified

%CM: Percentage Code and Unit Test Modified

%IM: Percentage Integration and Test Modified

Using a different set of percentages for the different code sources, ESLOC is expressed as

ESLOC = New SLOC +

(AAFM x Modified SLOC) +

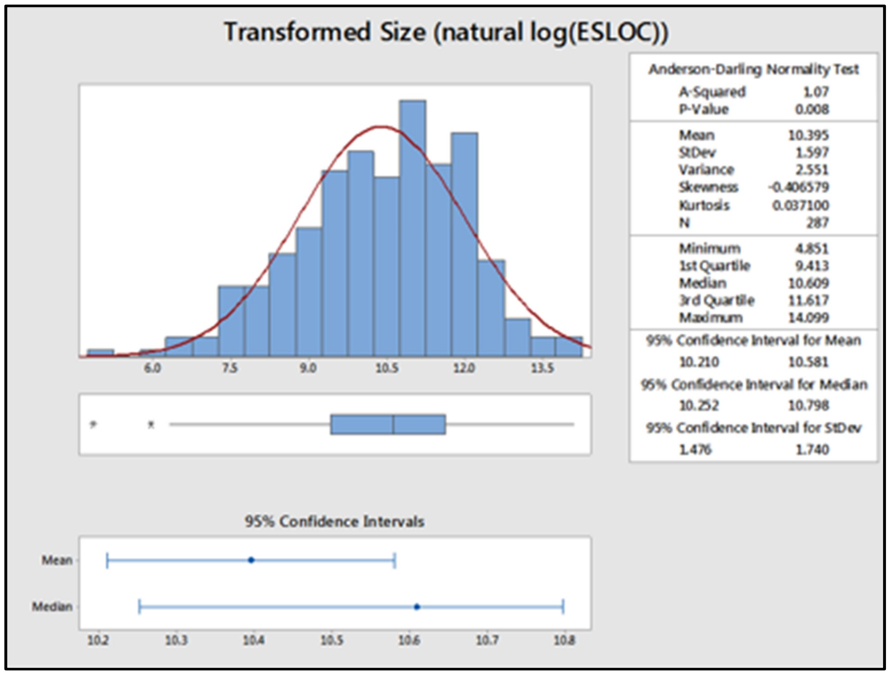
(AAFR x Reused SLOC) +

(AAFAG x Auto-Generated SLOC)

New code does not require any adaptation parameters, since nothing has been modified.

Auto-generated code does not require the DM or CM adaptation parameters. 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.

Equivalent source lines of code (ESLOC) normalize all code sources to the equivalent of a new line of code by computing a portion of the physical measures for modified, reused, and auto-generated code. Figure 3 shows the ESLOC data normalized using a natural log transformation. ESLOC by super domain is presented in Figure 4. An easy heuristic to use for average project size is **around 40,000 ESLOC** for all projects.



Anderson-Darling Normality Test	
A-Squared	1.07
P-Value	0.008
Mean	10.395
StDev	1.597
Variance	2.551
Skewness	-0.406579
Kurtosis	0.037100
N	287
Minimum	4.851
1st Quartile	9.413
Median	10.609
3rd Quartile	11.617
Maximum	14.099
95% Confidence Interval for Mean	
10.210	10.581
95% Confidence Interval for Median	
10.252	10.798
95% Confidence Interval for StDev	
1.476	1.740

Figure 3. Product Size in ESLOC, Normalized

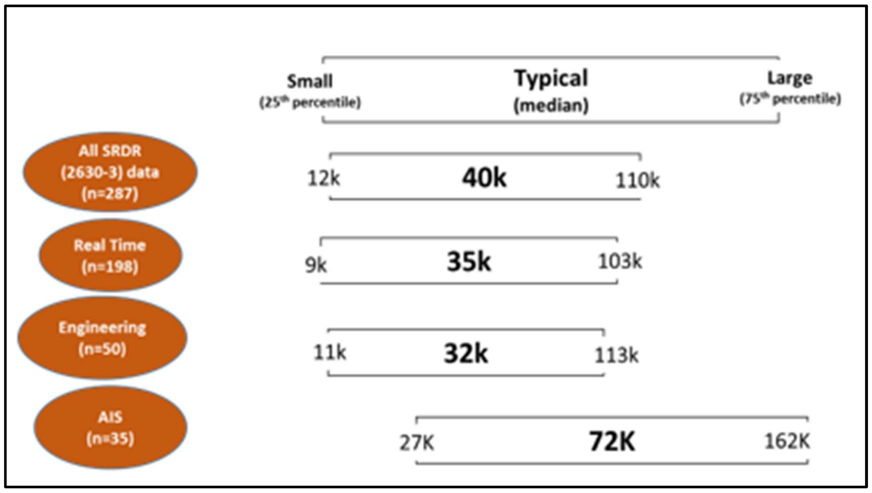
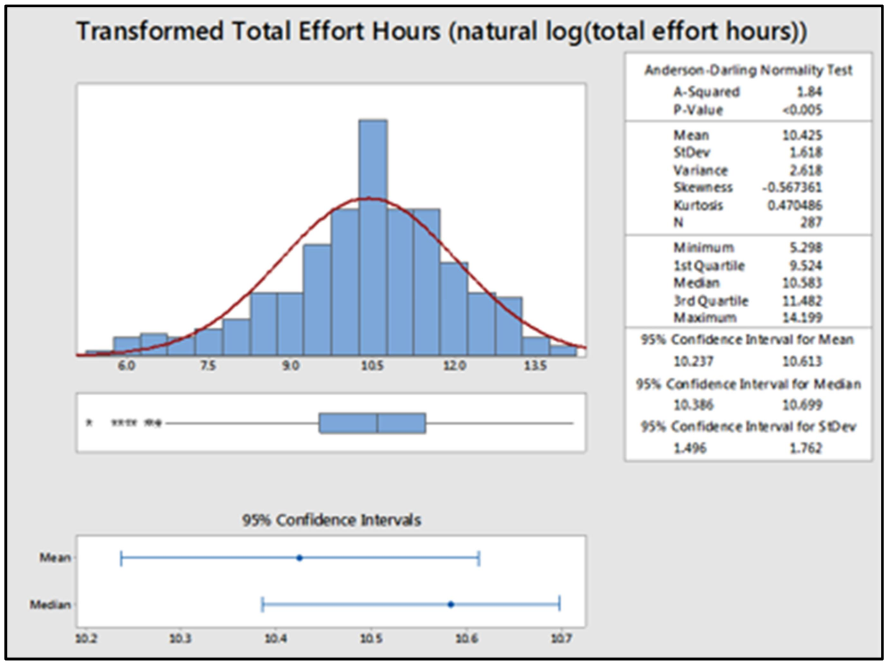


Figure 4. ESLOC by Super Domain

Effort

The amount of effort used to create software is the major driver of the cost of the development; the effort estimate in dollars provides the largest element in the cost estimate for software. Effort is usually collected in hours. For simplification purposes many estimation tools and equations use person months. When comparing effort data, ensure that the same conversion rate is used across the data set (i.e., the number of hours in a person month and/or number of hours in a full time equivalent). As detailed in Appendix G: Burden Labor Rate, it is assumed here that there are 152 hours in a labor month and 1824 hours per full-time equivalent (FTE), based on an annual labor rate of $150,000.

Figure 5 shows the effort data normalized. The effort hour data analyzed by super domain are presented in Figure 6. An easy heuristic to use for average project effort is around **40,000 hours, 263 person months, or 22 FTEs** for a DoD software project.



Anderson-Darling Normality Test	
A-Squared	1.84
P-Value	<0.005
Mean	10.425
StDev	1.618
Variance	2.618
Skewness	-0.567361
Kurtosis	0.470486
N	287
Minimum	5.298
1st Quartile	9.524
Median	10.583
3rd Quartile	11.482
Maximum	14.199
95% Confidence Interval for Mean	
10.237	10.613
95% Confidence Interval for Median	
10.386	10.699
95% Confidence Interval for StDev	
1.496	1.762

Figure 5. Effort, Normalized

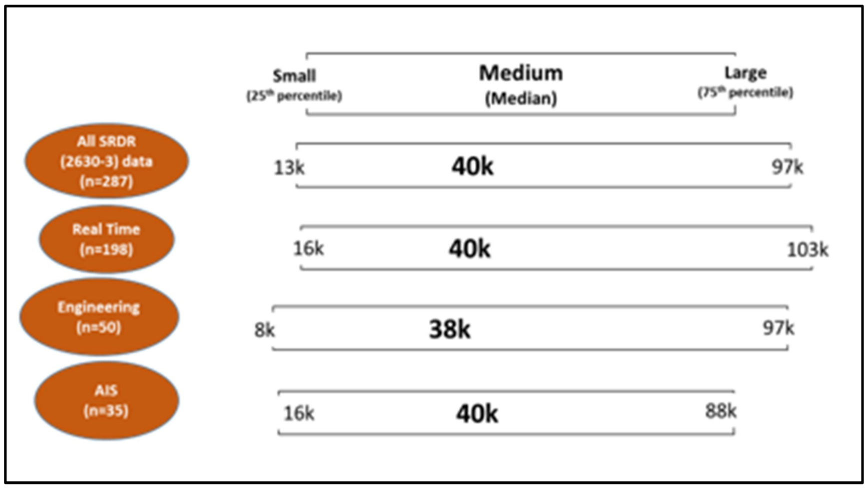


Figure 6. Effort Hours by Super Domain

Duration

Duration is a measure of the calendar time it takes to complete the software project. Many factors affect duration, including staffing profile, schedule constraints, and release plan. No adjustments are made for these factors in the data reported in this section.

Figure 7 shows the duration data normalized. The data indicate that the majority of projects take between 2 ½ to 3 years. An easy heuristic to use for the duration of an average DoD software project is **approximately 3 years**. Duration data analyzed by super domain is presented in Figure 8.

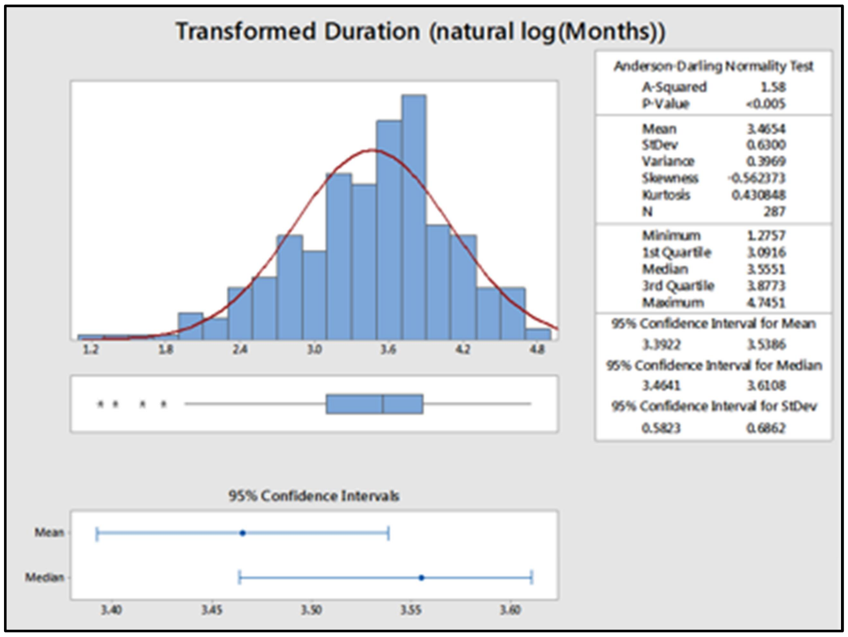


Figure 7. Duration, Normalized

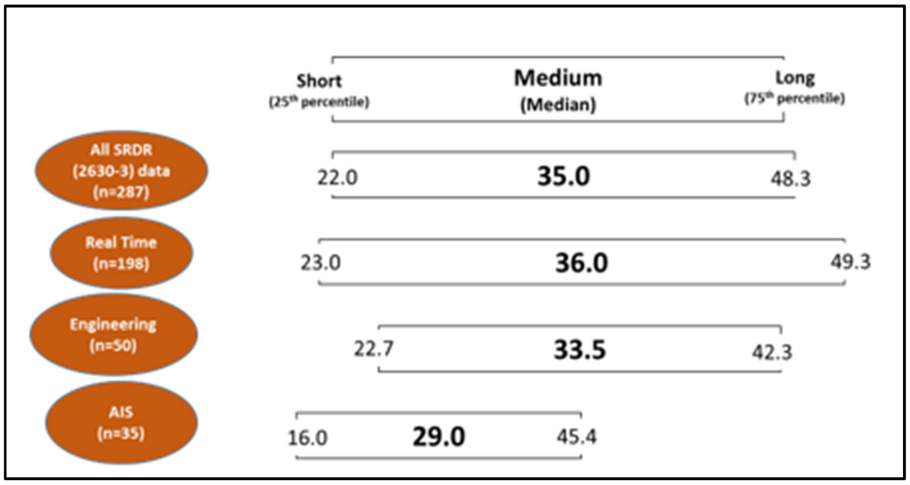
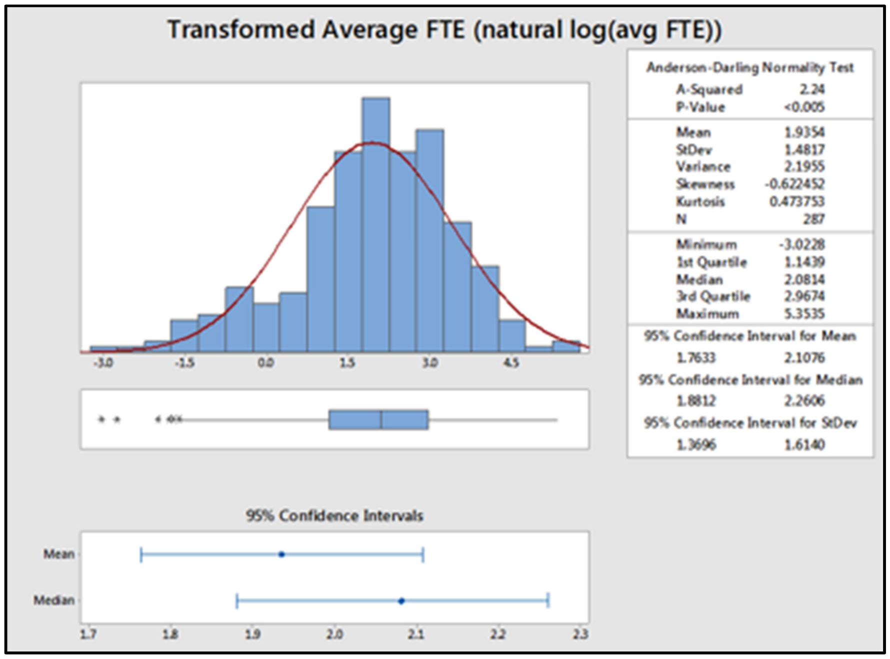


Figure 8. Duration Data by Super Domain

Team Size (People)

The size of the development team reported here is based on measures of project effort and duration. The effort for a project is reported in labor hours. Labor hours are converted to person months of effort (based on 152 hours/month) and divided by months of project duration. This derives the average level of project staffing or full time equivalent (FTE).

Figure 9 shows a histogram of the data in natural log scale. It shows that most teams have 20 or fewer people. Recall that SRDRs are required for contracts over $20 million. These contracts have multiple product events resulting in seemingly small team sizes which, in fact, are due to low levels of effort over relatively long durations.



Anderson-Darling Normality Test	
A-Squared	2.24
P-Value	<0.005
Mean	1.9354
StDev	1.4817
Variance	2.1955
Skewness	-0.622452
Kurtosis	0.473753
N	287
Minimum	-3.0228
1st Quartile	1.1439
Median	2.0814
3rd Quartile	2.9674
Maximum	5.3535
95% Confidence Interval for Mean	
1.7633	2.1076
95% Confidence Interval for Median	
1.8812	2.2606
95% Confidence Interval for StDev	
1.3696	1.6140

Figure 9. Time Size, Normalized

Figure 10 shows the data divided into three groups: small-, medium-, and large-team-size projects. The groups are based on a cumulative percentage divided into thirds. Small teams make up the lower third, medium size teams are in the middle third, and large teams make up the upper third. Based on the groupings the team sizes are as follows:

- small-size teams: < 5 average staff
- medium-size teams: 5–14 average staff
- large-size teams: > 14 average staff

Medium and large team sizes are used in the effort/schedule tradeoff analysis.

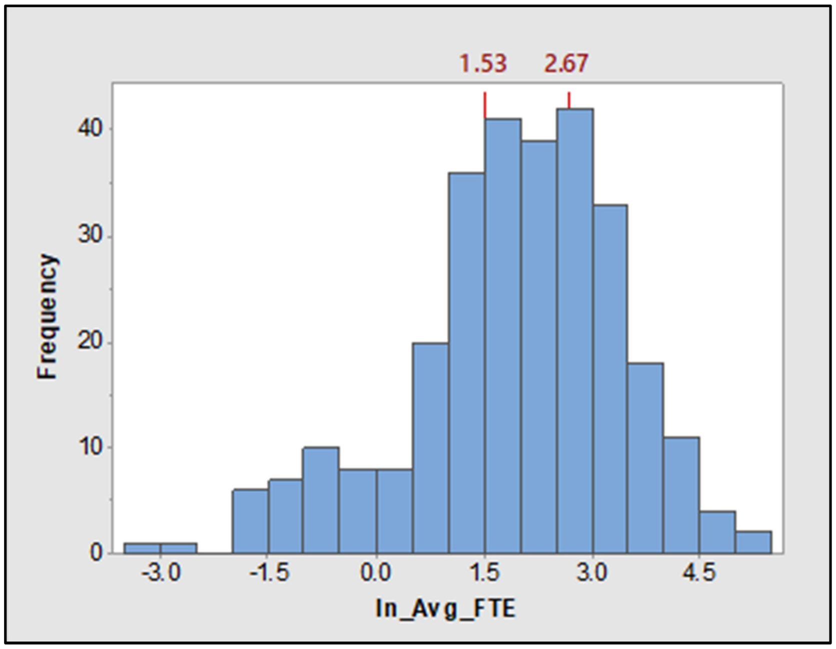


Figure 10. Team Size Distribution

Team size data analyzed by super domain is presented in Figure 11.

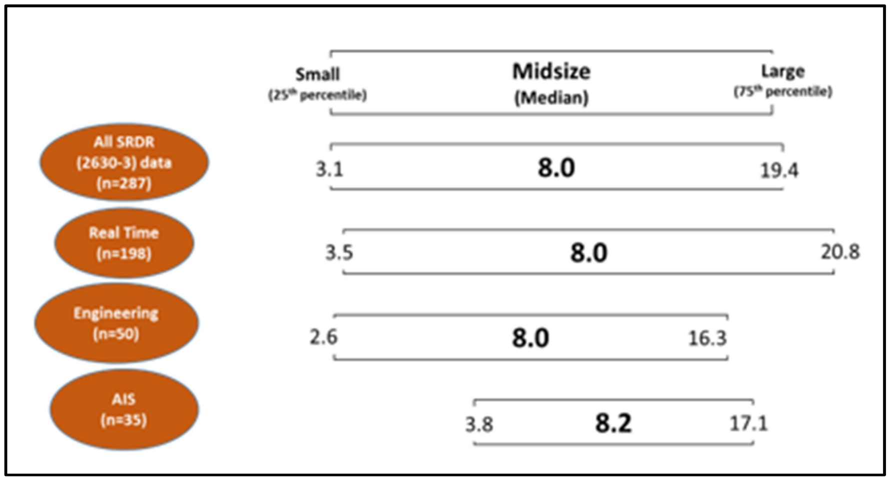


Figure 11. Team Size Data by Super Domain

Productivity

Productivity (also referred to as efficiency) is the amount of product produced for an amount of resource. For software, productivity is commonly measured by size (ESLOC) divided by effort hours.

Productivity in general is considered very competition sensitive and therefore rarely shared publicly by the private sector. Since the SRDR data set is owned and maintained by the government and the individual data provider's productivity is protected, this compilation of data provides a rarely available insight into software productivity across the industrial base.

Figure 12 shows the productivity data after normalization. For practical purposes, the data shows a 1:1 ESLOC: hour ratio. Productivity data analyzed by super domain is presented in Figure 13.

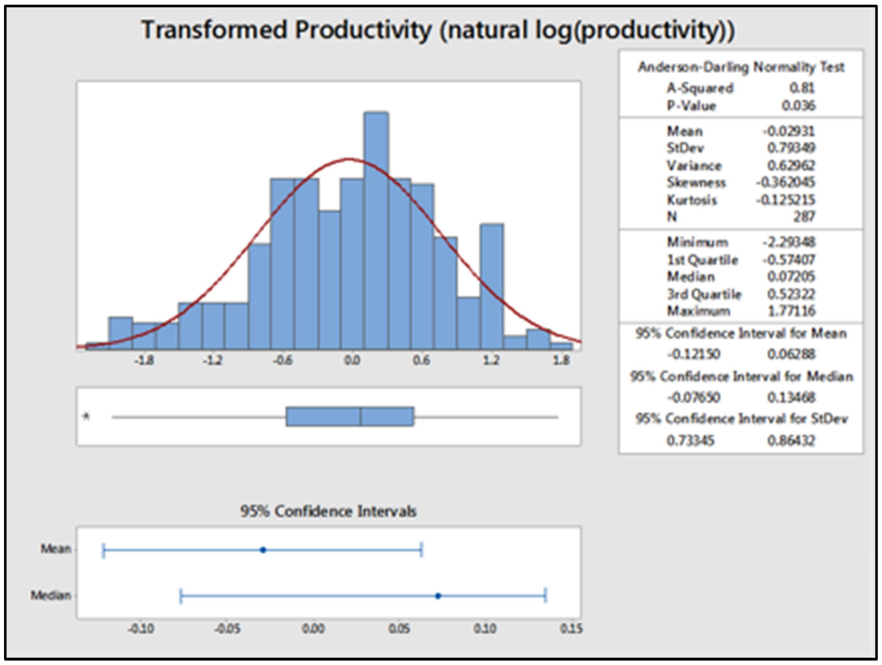


Figure 12. Productivity, Normalized

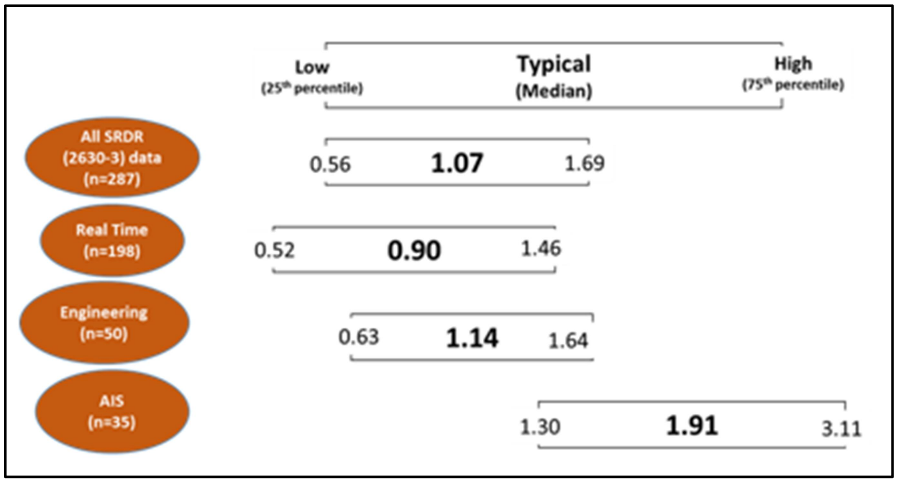


Figure 13. Productivity by Super Domain

Profiles of Typical Projects

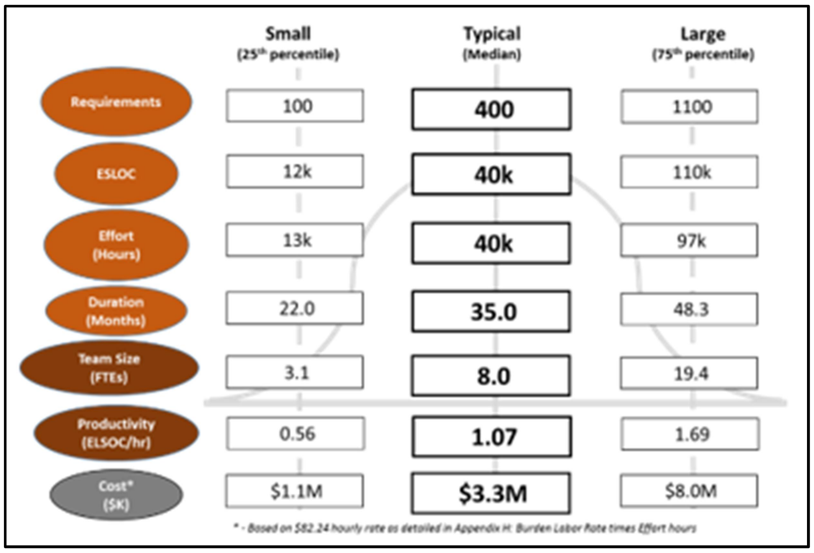
Integrating the analysis results of the individual parameters provides a general software project profile. This section contains the profiles for a generic DoD software project, as well as profiles for RT, ENG, and AIS projects.

As a reminder, the SRDR reports are provided at the project level or subsystem level, not at the DoD Acquisition Program level. **It is important to note that when the analysis refer to a "project" in this report, a project is synonymous with a software**

build, increment, or release. In many cases, several projects (i.e., data points) would contribute to the overall scope and make up of Acquisition Program (i.e., an entire weapon system).

Snapshot of a Typical DoD Software Project

Figure 14 provides a snapshot of the overall dataset, showing the size and scope of a typical DoD software project. Keep in mind SRDR data points are typically submitted by subsystem or potential increment; these numbers do not represent an entire DoD program of record.



	Small (25th percentile)	Typical (Median)	Large (75th percentile)
Requirements	100	**400**	1100
ESLOC	12k	**40k**	110k
Effort (Hours)	13k	**40k**	97k
Duration (Months)	22.0	**35.0**	48.3
Team Size (FTEs)	3.1	**8.0**	19.4
Productivity (ELSOC/hr)	0.56	**1.07**	1.69
Cost* ($K)	$1.1M	**$3.3M**	$8.0M

* - Based on $82.24 hourly rate as detailed in Appendix H: Burden Labor Rate times Effort hours

Figure 14. Parameters of DoD Software Projects

This data can be used to answer general questions about DoD software projects. For example,

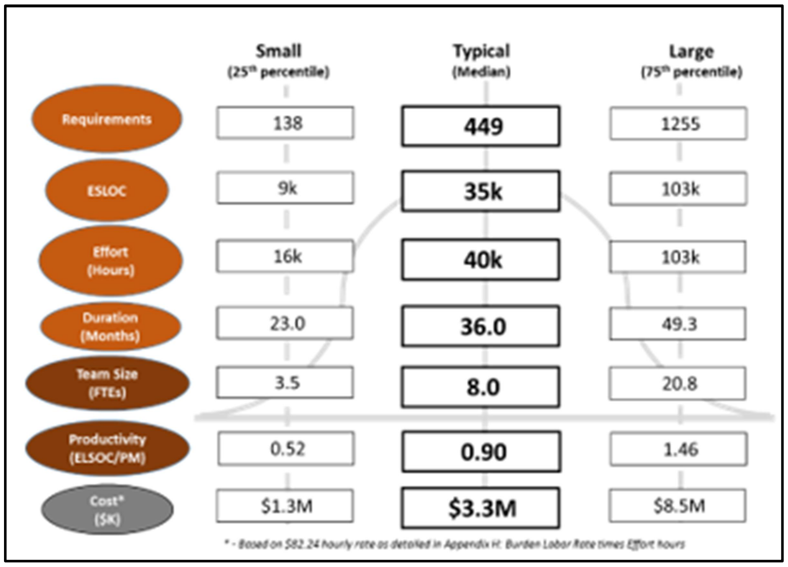
- Question: What is the typical (average) size of a software delivery?
  Answer: 40 KESLOC
- Question: How long does an increment take?
  Answer: 35 months (~3 years)
- Question: How many FTEs does a typical software project require?
  Answer: 8 FTEs; some large projects may require upwards of 20 FTEs.
- Question: In general how much does a software project cost?
  Answer: Software projects tend to range between $1 million and $8 million; without knowing any details about what type of software or its composition, a generic DoD project costs a little over $3 million.

The percentile numbers help convey the variation in the data. These data can be utilized by oversight offices when assessing overall program feasibility. A project plan that contains parameter values outside the 25th and 75th percentile range signifies a situation that is not common and might require additional scrutiny. In this case, the oversight office would want to ask for more information about the engineering and technical rationale to justify this plan.

Given the mix of system domains, language types, environments, platforms, functionality, and associated quality/performance parameters, these rules of thumb may not provide a lot of value to project managers estimating their software efforts. To get the information useful to them, they would need to isolate like projects in the dataset and generate a parameter profile that best represents the system they are developing. In this vein, the following sections provide heuristics by super domains.

Snapshot of Real-Time Software Projects

RT software is typically the most complex and intricate type of software. It tends to be embedded in the system architecture and contributes to the success or failure of key performance parameters of the system. Given the level of rigor this type of software requires, the variations between the RT super domain parameters in Figure 15 are not surprising. Of the 287 data points analyzed, 198 were classified as real time.



	Small (25th percentile)	Typical (Median)	Large (75th percentile)
Requirements	138	**449**	1255
ESLOC	9k	**35k**	103k
Effort (Hours)	16k	**40k**	103k
Duration (Months)	23.0	**36.0**	49.3
Team Size (FTEs)	3.5	**8.0**	20.8
Productivity (ELSOC/PM)	0.52	**0.90**	1.46
Cost* ($K)	$1.3M	**$3.3M**	$8.5M

* - Based on $82.24 hourly rate as detailed in Appendix H: Burden Labor Rate times Effort hours

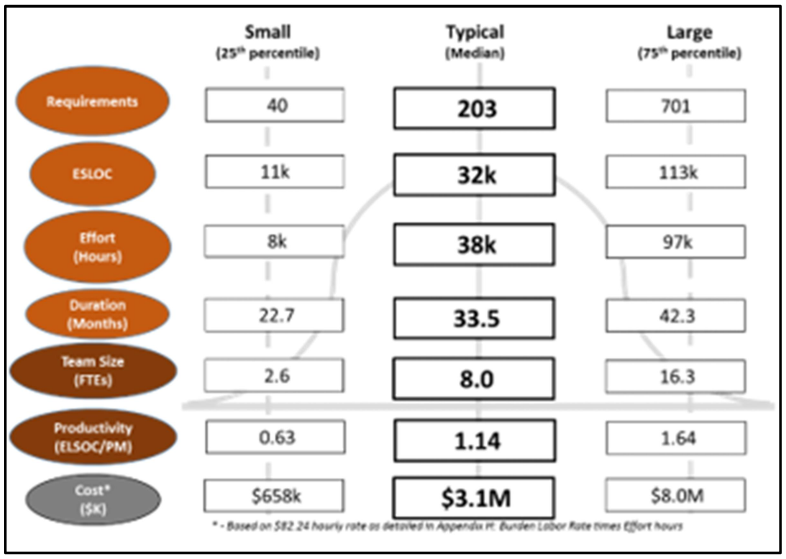
Figure 15. Parameters of Real-Time Software Projects

It is logical that increased system complexity would require a more detailed articulation of the requirements, resulting in a higher requirements count and lower productivity in comparison to the overall data set. This can also be seen in the slightly higher effort hour percentile values.

Snapshot of Engineering Software Projects

ENG super domain software is of medium complexity. It requires engineering external system interfaces, high reliability (but not life-critical) requirements, and often involves coupling of modified software. Examples of software domains in this super-domain are: mission processing, executive, automation and process control, scientific systems, and telecommunications.

Figure 16 shows the key software parameters for the 50 ENG super domain data points in the 287 data set.



	Small (25th percentile)	Typical (Median)	Large (75th percentile)
Requirements	40	**203**	701
ESLOC	11k	**32k**	113k
Effort (Hours)	8k	**38k**	97k
Duration (Months)	22.7	**33.5**	42.3
Team Size (FTEs)	2.6	**8.0**	16.3
Productivity (ELSOC/PM)	0.63	**1.14**	1.64
Cost* ($K)	$658k	**$3.1M**	$8.0M

* - Based on $82.24 hourly rate as detailed in Appendix H: Burden Labor Rate times Effort hours

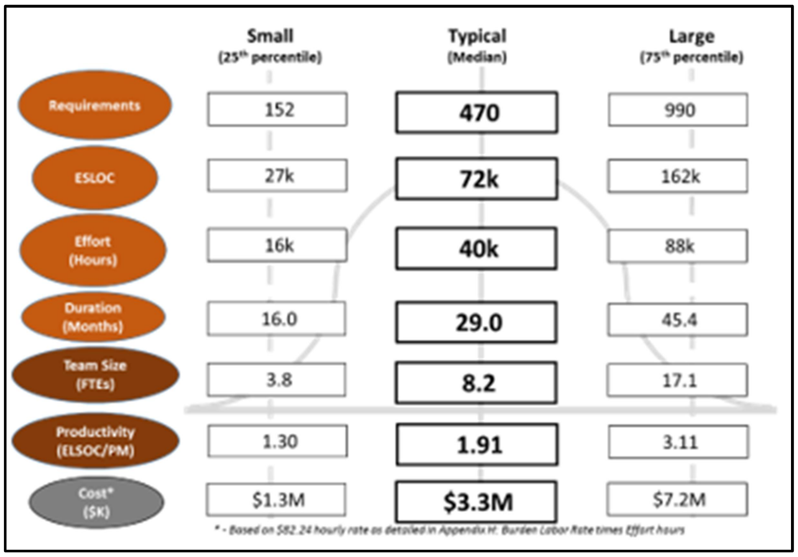
Figure 16. Parameters of Engineering Software Projects

In comparison to RT systems, ENG systems tend to state their requirements at a slightly higher level. For example, a typical requirement may be, "System X shall interface with System Y." In this case there are several nuances to meeting this requirement. This can be seen by comparing the requirements parameters, ESLOC, and effort parameters of the RT data to the ENG data.

Snapshot of Automated Information System Software Projects

AIS software automates information processing. These applications allow the designated authority to exercise control over the accomplishment of the mission. Humans manage a dynamic situation and respond to user input in real time to facilitate coordination and cooperation. Examples of software domains in this super-domain include intelligence and information systems, software services, and software applications.

Figure 17 shows the key software parameters for the 35 AIS super domain data points in the 287 data set.



	Small (25th percentile)	Typical (Median)	Large (75th percentile)
Requirements	152	470	990
ESLOC	27k	72k	162k
Effort (Hours)	16k	40k	88k
Duration (Months)	16.0	29.0	45.4
Team Size (FTEs)	3.8	8.2	17.1
Productivity (ELSOC/PM)	1.30	1.91	3.11
Cost* ($K)	$1.3M	$3.3M	$7.2M

* - Based on $82.24 hourly rate as detailed in Appendix H: Burden Labor Rate times Effort hours

Figure 17. Parameters of Automated Information System Software Projects

The size and productivity parameters vary the most from the overall super domain parameters. Based on the way AIS are developed (i.e., adaptation of existing COTS/GOTS applications), the increase in comparison to the other super domains is not surprising.

Portfolio Performance: Common Questions

This section explores the findings by super domain to answer some common questions about software types.

Most and Least Expensive Software

What are the most and least expensive software types to develop?

Our analysis is based on the rationale that some types of software are more difficult to develop than other types and therefore require more effort to develop. The level of difficulty can be caused by factors such as execution timing constraints, interoperability requirements, commercial-off-the-shelf (COTS) software product incorporation, algorithmic complexity, communication complexity, data-bandwidth requirements, and security requirements. To account for the dissimilarities in project difficulty, projects are segregated into three super domains.

The analysis proceeds by introducing two concepts: unit cost and production rate.

- Unit cost is the cost of producing a unit of software with some amount of effort. In this case, the unit of software is thousands of equivalent source lines of code (KESLOC). The effort is reported in labor hours, which can be translated into cost using an average labor rate.
- Production rate is the rate at which a unit of software is delivered over a period of time. The unit of software is a KESLOC and the time is days of project duration.
- Cost is derived by applying a burdened labor rate to the number of labor hours worked in a day. Hours per day are determined by dividing total hours

by the duration (total days). For example, if a real-time project required 1,007 total hours and 25 days, the labor hours expended in a day is 40.3 (implying several people were working on the project).

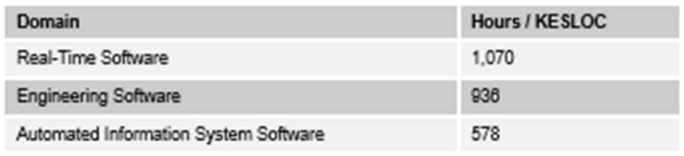
The analysis then normalizes the unit cost with the production rate, creating a high-level comparison. This is done because some projects may choose to employ more staff to increase their production rate and deliver the software sooner or vice versa. The resulting effort per day is then multiplied by an average burden labor rate to derive cost.

Unit Cost

With an average project size of 40,000 ESLOC, each of the three groups are analyzed separately. Trends for each group were created based on a natural log-transformation of the data. This transformation made it easier to see the relationships between the three groups for an average project size of 40,000 ESLOC.

The difference in unit costs between the three groups is shown in Table 1. Real-time software shows that for small amounts of size, a large amount of effort is required. Automated information system software data shows the opposite: for large amounts of size, a small amount of effort is required.

Table 1. Unit Costs for Different Domains



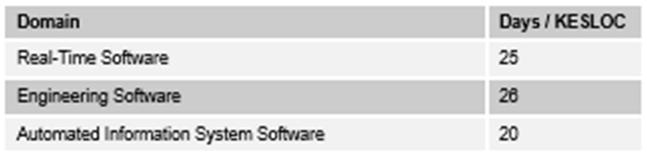
Domain	Hours / KESLOC
Real-Time Software	1,070
Engineering Software	936
Automated Information System Software	578

Production Rate

The production rate data analysis focused on the relationships between size and duration for the three super domains. The analysis revealed much greater variability than the unit cost plot. This indicates a very weak systematic relationship between size and duration. The dispersion of the data is attributed to other factors that influence the size-duration relationship (e.g., different levels of staffing on similar size projects can impact duration). This is an area for further research.

For an average-size project, the production rate (how long it takes to deliver a unit of software) is shown in Table 2.

Table 2. Production Rate for Different Domains



Domain	Days / KESLOC
Real-Time Software	25
Engineering Software	26
Automated Information System Software	20

Cost Comparison

When unit cost is divided by production rate, the average number of hours each month is determined. Using an average burden labor rate, the normalized monthly cost for each group is shown in Table 3. The hours/day indicate that more than one person is working per day.

Table 3. Costs for Different Domains

Domain	Hrs / Day	Cost / Day
Real-Time Software	40.4	$3,324
Engineering Software	35.4	$2,912
Automated Information System Software	29.1	$2,393

Real-time software is the most expensive to develop and automated information system software is the least expensive. RT software costs 14% more to develop than ENG software and 39% more than AIS software.

Cost Heuristics

Units for cost vary based on the office reporting them and the types of decisions that are being made. Engineering organizations often prefer to discuss things in technical units (e.g., requirement and SLOC) and effort (e.g., hours or person months, months). Cost offices tend to communicate in terms of dollars and fiscal years. Table 4 is a translation table that shows the same unit cost, production rate, and cost data expressed in different units.

Table 4. Unit Cost and Productivity

Project Size (40 KESLOC)	Unit Cost	Production Rate		
Domain	Hours / KESLOC	Days / KESLOC	Hrs / Day	Cost / Day
Real-Time Software	1,007	25	40.4	$3,324
Engineering Software	936	26	35.4	$2,912
AIS Software	578	20	29.1	$2,393

Project Size (40 KESLOC)	Productivity				
Domain	ESLOC / Hour	ESLOC / Day	People (FTEs)	Cost Month	Cost per Year
Real-Time Software	0.99	40	5.1	$99,720	$1,196,640
Engineering Software	1.07	38	4.4	$87,360	$1,048,320
AIS Software	1.73	50	3.6	$71,790	$861,480

Table 4 provides the unit cost (hours/KESLOC) and its inverse, productivity (ESLOC/hour). Depending on the type of information needed, one of the metrics may be preferred over the other. Alternatively, production rate is a metric that can be expressed in terms of units of product produced in a period of time (days/KESLOC) or units of time to produce a single product (ESLOC/day). It also provides monthly and annual costs by domain. The cost by year represents the annual costs for an average project for a full calendar year. This number doesn’t help an engineering organization determine the total cost of a particular project, but it is a useful metric to technical managers when they are required to submit an annual budget.

Best-in-Class/Worst-in-Class

What differences are there between best-in-class and worst-in-class software projects?

This analysis examines the best- and worst-in-class projects within each of the three super-domains discussed in the previous section. To assess differences between projects, the three derived metrics explained in the previous section are used: unit cost, production rate, and cost.

Analysis Approach

An average size project within each super domain is used to derive unit cost, production rate, and cost. A ±1 standard error (SE) about the unit cost and production rate trend lines were used to identify best- and worst-in-class projects.

The definition of best-in-class and worst-in-class projects were developed as follows:

- Best-in-class projects: at or below the −1 SE value are projects that used less effort or less time to finish than an average project. This boundary represents the worst of the best-in-class projects—performance may actually be better.
- Worst-in-class projects: at or above the +1 SE value are projects that used more effort or more time to finish than an average project. This boundary represents the best of the worst-in-class projects—performance may actually be worse.

Real-Time (RT) Software

Unit Cost

The average-size RT project (34,000 ESLOC for the RT domain) expends 39,664 labor hours of effort. Best-in-class projects expend 18,361 labor hours and worst-in-class projects expend 85,687 labor hours, a 10-fold increase. The difference between a best- or worst-in-class project from the average project is 21,304 labor hours. It is important to understand the context of the labor-hour differences in conjunction with project duration.

Production Rate

The average-size project delivers a product in 997 days (32.8 months). A best-in-class project delivers a product in 538 days (17.7 months). A worst-in-class project delivers a product in 1,848 days (60.8 months).

Cost

Table 5 summarizes the differences in unit cost and production rate between best-, average-, and worst-in-class RT projects. An average RT size project of 34,000 ESLOC was used to determine effort and schedule. Best-in-class RT projects are 2 times more efficient than average projects and 4.7 times more efficient than worst-in-class projects. Best-in-class projects are 1.8 times faster than an average projects and 3.4 times faster than a worst-in-class project. As mentioned earlier, the noted results for the best-in-class are the lowest reported numbers in the best-in-class bracket. Conversely, the reported results for worst-in-class are the highest reported numbers in the worst-in-class bracket.

Table 5. Real-Time Software Best and Worst Summary

Metric	Best-in-Class	Average	Worst-in-Class
Effort (Labor Hours)	18,361	39,664	85,687
Schedule (Days)	538	997	1,848
Cost (per Day)	$2,805	$3,271	$3,813
Total Cost ($M)	$1.510	$3.262	$7.047

Using a burden labor rate of $150,000 per year, the best-in-class project saves $1.752 million dollars over an average project and $5.537 million over a worst-in-class project.

Engineering (ENG) Software

Unit Cost

There are 50 projects in the ENG super-domain. The average-size project (32,000 ESLOC for the ENG domain) expends 30,780 labor hours of effort. The best-in-class expends 14,468 labor hours and the worst-in-class expends 65,485, a 4.5 increase times the amount of best in class. The difference between a best- and worst-in-class project from the average project is 16,312 hours.

Production Rate

The best-in-class project delivers a software product in 640 days (21 months), an average project in 1,031 days (33.9 months), and a worst-in-class project in 1,659 days (54.6 months).

Cost

Table 6 summarizes the differences in unit cost and production rate between best, average, and worst-in-class ENG projects. An average ENG size project of 32,000 ESLOC was used to determine effort and schedule. The best-in-class ENG projects are 2.3 times more efficient than average projects and 5.3 times more efficient than worst-in-class projects. The best-in-class project is 1.6 times faster than an average project and 2.6 times faster than a worst-in-class project.

Table 6. Best and Worst Summary of Engineering Software

Metric	Best-in-Class	Average	Worst-in-Class
Effort (Labor Hours)	14,468	30,780	65,485
Schedule (Days)	640	1,031	1659
Cost (per Day)	$1,859	$2,456	$3,246
Total Cost ($M)	$1.190	$2.531	$5.385

Best-in-class projects save $1.341 million dollars over average projects and $4.195 million dollars over a worst-in-class project.

Automated Information System (AIS)

Unit Cost

Using an average-size project of 72,000 ESLOC, best-in-class, average, and worst-in-class projects expended an average of 22,400, 39,114, and 68,297 labor hours of effort, respectively. There is a three-fold increase in effort between best and worst-in-class. The difference between a best or worst-in-class project and the average project is 16,713 labor hours.

Production Rate

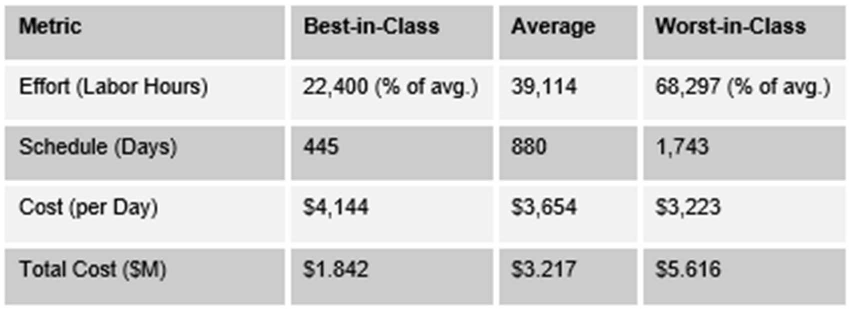
The best-in-class average-size project delivers a product in 445 days (14.6 months). The average project delivers a product in 880 days (29 months). The worst-in-class a project delivers product in 1,743 days (57.3 months).

Cost

Table 7 summarizes the differences in unit cost and production rate between best, average, and worst-in-class projects. An average AIS size project of 72,000 ESLOC was used to determine effort and schedule. That makes best-in-class projects 1.7 times more efficient than average projects and 3 times more efficient than a worst-in-class projects. Best-in-class projects are 2 times faster than average projects and 4 times faster than worst-in-class projects.

Best-in-class projects save $1.375 million over average projects and $3.774 million over worst-in-class projects.

Table 7. Best and Worst Summary of AIS Software



Metric	Best-in-Class	Average	Worst-in-Class
Effort (Labor Hours)	22,400 (% of avg.)	39,114	68,297 (% of avg.)
Schedule (Days)	445	880	1,743
Cost (per Day)	$4,144	$3,654	$3,223
Total Cost ($M)	$1.842	$3.217	$5.616

Project Planning, Trade-Offs and Risk

As part of our analysis we also explored correlations among requirements, size, duration, and effort. The goal of this work was to determine how well the data could be used to answer common questions related to planning or replanning software projects, such as "How much growth should we plan for?" and "How well can initial estimates be used to predict final outcomes?"

The Department of Defense Software Factbook provides a more extensive description of our work in this area, while this paper provides a brief overview of the strongest models we found to predict growth in requirements, ESLOC, schedule, and effort from the initial estimates. Each of the models can be used to construct predicted growth intervals for any given initial estimate, although we caution against using the model outside the bounds indicated by the 5th and 95th percentiles for each variable.

Estimation Relationships

Among the many factors and models for estimating effort, the SRDR data allows us to investigate the relationship between requirements and the size of the effort and then the relationship between the estimated size and the estimated effort as well as the final effort. A simple look at the correlations among requirements, size, duration, and effort found that the only actionable correlation was between size and effort.

Predicting Actual Total Effort by Estimated ESLOC

The following model shows that an initial estimate of ESLOC can also be used to predict the total actual effort. Although the model is only moderately strong, it is presented here in case an initial estimate of effort is not available, but an estimate of size (ESLOC) is available.

n = 163

Regression Equation:

$$\ln \text{Total Hours\_Actual} = 2.031 + 0.8259 \ln \text{ESLOC\_Estimated}$$

which translates to:

$$Actual\ Total\ Hours = 7.614 * (Estimated\ ESLOC)^{.83}$$

The table shows the predictions have a "sweet spot" that is +/− 10% in the range from 75KESLOL to 200 KESLOC. The model accounts for over 67% of the variance. Below are the predicted (forecast) values and prediction ranges for a set of new given inputs, followed by a graphic showing the actual data fitted to the model along with the associated prediction intervals. Predicted values show an underestimate of the initial by 158% at the low end (500 ESLOC) but an overestimate of −22% at the high end (500K ESLOC). This indicated that the model is reasonably good fit to the data.

Table 8. Prediction Values for Actual Total Hours (Effort) Using ESLOC

Initial ESLOC Estimate	Forecast Total Hours	Percent Difference From Estimate	Prediction Interval—Total Hours	
			Lower 95%	Upper 95%
500	1,291	158%	264	6,305
750	1,805	141%	372	8,747
1,000	2,289	129%	475	11,040
2,500	4,879	95%	1,024	23,235
5,000	8,648	73%	1,828	40,911
7,500	12,088	61%	2,562	57,025
10,000	15,330	53%	3,255	72,213
25,000	32,675	31%	6,949	153,635
50,000	57,921	16%	12,300	272,755
75,000	80,961	8%	17,158	382,026
100,000	102,674	3%	21,717	485,437
150,000	143,515	−4%	30,249	680,898
200,000	182,006	−9%	38,248	866,094
300,000	254,403	−15%	53,200	1,216,562
400,000	322,634	−19%	67,199	1,549,009
500,000	387,926	−22%	80,526	1,868,786

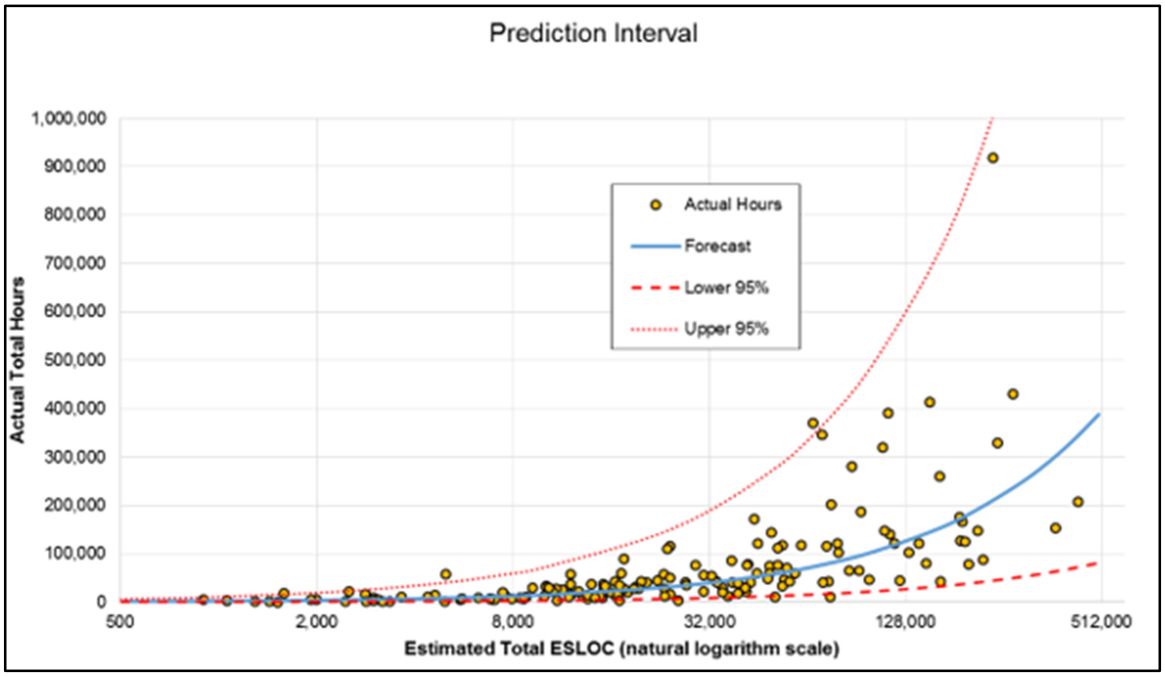


Figure 18. +/−10% is the Range for 75,000 to 200,000 Initial ESLOC Estimates With +/−10%

Software Growth—Predicting Outcomes

To determine if final outcomes can be predicted from initial estimates we examined the project performance as represented by 181 paired initial and final contractor submissions. As such, we measured the difference between the initial estimates and the actual outcomes.

Based on historical SRDR data transformed to natural logarithms, we determined that we can predict (with a known degree of certainty) the expected outcomes for software size, schedule, and effort. The models presented enable predictions of final outcomes based on initial estimates. Each of the models can be used to construct outcome prediction intervals for any given initial value, although we caution against using the model outside the bounds indicated by the 5th and 95th percentiles for each variable.

While the full report describes the data and statistical analyses in more detail, we provide here an overview of the strongest models to emerge from this analysis:

Requirements	$(r^2 = .936)\ Actual\ Total\ Reqts = 1.2838 * (Estimated\ Total\ Reqts)^{.9456}$
ESLOC	$(r^2 = .849)\ Actual\ Total\ ESLOC = 2.0157 * (Estimated\ ESLOC)^{.964}$
Schedule	$(r^2 = .776)\ Actual\ Total\ Duration = 2.3054 * (Estimated\ Total\ Duration)^{.7878}$
Effort	$(r^2 = .898)\ Actual\ Total\ Hours = 3.3128(Estimated\ Total\ Hours)^{.9097}$

Predicting productivity is less strong unless we separate the underestimated cases from the overestimated cases, which then yield very strong models (r^2 equals .886 and .758, respectively). This indicates that if the productivity could be assessed during the development effort, then the actual outcome could be more accurately predicted. If we also account for the type of super domain, these models increase in strength.

Schedule duration can also be separately predicted for the three services. We show that total effort hours can also be predicted by using the initial estimate for ESLOC, although the fit is not as strong ($r^2 = .674$) as using the initial estimate for hours. We also show how the prediction interval becomes tighter when the confidence level for the prediction is reduced.

Perhaps the most useful takeaway from this analysis are the prediction tables. The tables provide the predicted value along with the prediction interval at a 95% confidence level. These can be used in the absence of any available estimates, or as a sanity check against estimates coming from other sources. New values can easily produce a ballpark forecast by interpolation or the actual equation can be used for calculation. The data sets we used are also available for distribution which enable users to reproduce the models with their own statistical software.

As mentioned earlier, no further adjustments were made in case selection once the data were trimmed. Undoubtedly, the models could be improved (and the predictive intervals narrowed) with substantive knowledge concerning the behavior of outliers which could provide meaningful reasons for their exclusion from a model. Also, any additional data supplied during the interim of the project—which is under consideration by the DoD—could further calibrate and improve a model's fit. This would be especially useful in the productivity models where the best fits were determined by whether the original submission over- or underestimated the productivity. A midcourse determination of productivity would then indicate which sub-model was appropriate to estimate the final productivity for the project.

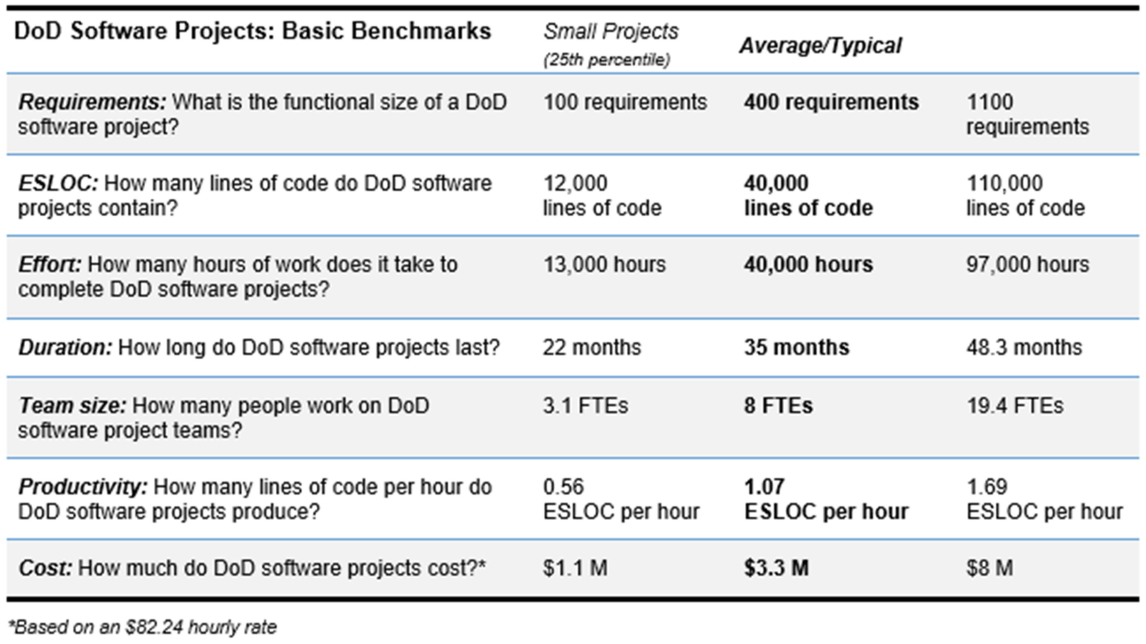
Conclusion

The analyses conducted by the SEI shows that the cost of software development varies depending on several factors. The class or super-domain of software makes a difference in the cost of software. Different super domains have different levels of difficulty that cause more effort to be expended on more difficult software. On an average-size project, AIS software costs $31,350 a month and RT software costs $101,250 a month—more than three times as much.

The time to develop software also drives cost. Based on an average-size project, shorter duration projects cost disproportionately more than longer duration projects. It was shown that team size is clearly NOT determined solely by the size of the software to be built. The performance of a project also drives cost. The analysis looked at best-, average-, and worst-performing projects within each super-domain.

Perhaps the most valuable contribution of this study is the ability to provide, for the first time, guidance to decision makers about projects that is based on empirical analysis. Table 9 summarizes the basic benchmarks that can now be used throughout the DoD.

Table 9. Basic Benchmarks for DoD Software Projects



DoD Software Projects: Basic Benchmarks	*Small Projects* *(25th percentile)*	***Average/Typical***	
Requirements: What is the functional size of a DoD software project?	100 requirements	**400 requirements**	1100 requirements
ESLOC: How many lines of code do DoD software projects contain?	12,000 lines of code	**40,000 lines of code**	110,000 lines of code
Effort: How many hours of work does it take to complete DoD software projects?	13,000 hours	**40,000 hours**	97,000 hours
Duration: How long do DoD software projects last?	22 months	**35 months**	48.3 months
Team size: How many people work on DoD software project teams?	3.1 FTEs	**8 FTEs**	19.4 FTEs
Productivity: How many lines of code per hour do DoD software projects produce?	0.56 ESLOC per hour	**1.07 ESLOC per hour**	1.69 ESLOC per hour
Cost: How much do DoD software projects cost?*	$1.1 M	**$3.3 M**	$8 M

**Based on an $82.24 hourly rate*

While this information is valuable, there was not enough background data on projects to investigate some important issues, such as why best and worst projects perform differently. This leads to the next steps. There is an effort to link the project data back to source documents and other data to provide the capability to investigate the data more fully. There is a lot of data and source material, and some progress has been made to date with a lot more to do. While more analysis will be done, we would like to hear from you. What are the important questions that need answers? For comments and suggestions, please contact: fact-book@sei.cmu.edu.

Reference

Boehm, B., Abts, C., Brown, W., Chulani, S., Clark, B., Horowitz, E., Madachy, R., Reifer, R., & Steece, B. (2000). *Software cost estimation with COCOMO II.* Prentice Hall.

Disclaimer and Distribution Statement