Technomics
Better Decisions Faster

Dynamic Software Effort Estimation

How SWEET It Is!

Will Gellatly-wgellatly@technomics.net

Lindsey Jones-ljones@technomics.net

Alex Wekluk-awekluk@technomics.net

Dave Brown-dbrown@technomics.net

Peter Braxton-pbraxton@technomics.net

International Cost Estimating and Analysis Association (ICEAA)

2/24/2022

Abstract

Software estimation is a complex and diverse field that must accommodate technical inputs that vary by life-cycle phase and estimating approach. In response to this challenge, our team developed the Software Effort Estimating Tool (SWEET), an Excel-based model designed for both flexibility and transparency. Practitioners are often faced with both a dearth of software data and opaque “black-box” models with a plethora of dubious inputs. Our approach circumvents these issues by building on the widely-accepted International Software Benchmarking Standards Group (ISBSG) database and providing the analyst direct insight into the effect of potential cost drivers. SWEET dynamically builds an effort estimating relationship (EER), reflecting in real time the analyst’s selection of data fields and source data. Our paper explores how we combined statistical analysis with practical knowledge of software development and the acquisition cycle to drive the development of this “clear-box” model. We balanced filtering out data fields of limited utility with providing the analyst the ability to generate an essentially limitless number of relationships.

Keywords: *Software, Agile, Tools, CERs*

Table of Contents

Introduction

As practicing software cost analysts, the authors of this paper are constantly looking for ways to improve estimating capability for projects that include a significant software development activity. The Software Effort Estimation Tool (SWEET) was created to address a specific need: what is the best way to build an estimate of software development, based on a known functional size? We believe that the best methods are highly defensible and transparent, in both the source data and underlying calculations.

As shown in the figure below, estimating method selection is influenced by when the estimate is being generated, and is also dependent upon the data available for a software development estimate.

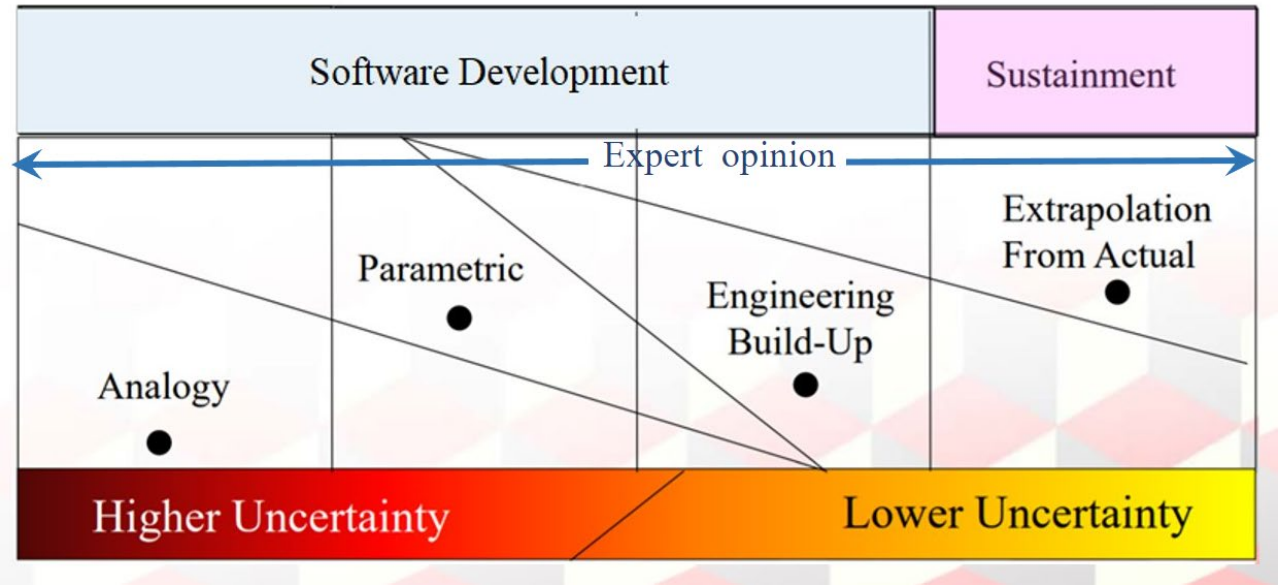


Figure 1: Software Estimating Methods

In the above graphic, the most defensible estimates are produced later in the life cycle, using an extrapolation from actuals method (Cost Estimating Body of Knowledge (CEBoK), 2013). If analysts have an actual cost history available, then a method such as the one described in ***Are We Agile Enough to Estimate Agile Software***

Development Costs? is most appropriate. (Kosmakos & Brown, 2022) If an actual cost history is not available and requirements are mature enough to estimate functional size, then a parametric method and tool such as SWEET is an ideal choice. It should be acknowledged, however, that an estimate of functional size is not always available, especially for programs/projects early in the life cycle. In these situations, an alternative method is preferred, such as the one described in ***Uncertainty of Expert Judgment in Agile Software Sizing***. (Braxton, 2022)

This paper presents the data and analysis used to develop SWEET. By using dynamic CER generation, transparent data, and traceable calculations, we offer a highly defensible solution for software cost estimating. It is best employed when the analyst needs a parametric solution based on known functional size.

Technomics Research Competition

The development of SWEET was funded by the Technomics Research Competition (TRC). TRC was created to encourage analyst growth and the advancement of methods, tools, and processes in the wide range of fields that apply to day-to-day project work. The competition allows employee-owners to explore topics of interest to them with the goal of producing a practical work product that can improve the organizations Technomics supports. In addition, the TRC provides the opportunity for employee-owners to work with colleagues from outside of their specific projects or areas of expertise, thus allowing for sharing of knowledge and diverse persectives as they work toward the development of their work product. The work discussed in this paper was conducted as part of the annual TRC during calendar year 2021.

Problem Statement

Software Development Estimation

Estimation of software development effort and cost is a long-standing challenge in both the IT and cost estimating communities. History has shown that for many IT programs and projects, software development represents not only a large portion of total cost, but also carries a disproportionately high level of cost risk. Although software development is often a focused area of cost estimation, analysts generally have a poor track record[1]. This historical record is shown in Table 1.

Table 1: Software/IT Historical Cost Growth

	Olympics	Software/ IT	Dams	NASA/ DoD	Rail	Bridges/ Tunnels	Roads
Average Cost Growth	156%	43-56%	24-96%	52%	45%	34%	20%
Frequency of Occurrence	10/10	8/10	8/10	8/10	9/10	9/10	9/10
Frequency of Doubling	1 in 2	1 in 4	1 in 5	1 in 6	1 in 12	1 in 12	1 in 50
Average Schedule Delay	0%	63-84%	27-44%	27-52%	45%	23%	38%
Frequency of Schedule Delay	0/10	9/10	7/10	9/10	8/10	7/10	7/10

As noted in the table and source material (Smart, 2021), software/IT cost over-runs are:

- Common: multiple industries experienced significant cost overruns for a long time
- Frequent: 70-80% of projects experience cost and schedule growth
- High: average cost growth of 50% or more
- Extreme: cost growth in excess of 100% is a common occurrence (1 in 4)

Further evidence shown in Figure 2 suggests that within the realm of IT, software is particularly prone to cost overruns. (McKinsey, n.d.)

[1] Actually, cost estimators too often get blamed for a poor track record. When actuals deviate widely from estimates, it could be due to: (1) poor estimates; (2) ignored estimates (i.e., the baseline from which growth is measured was not set to the estimate); or (3) poor performance. It very well could be all three!

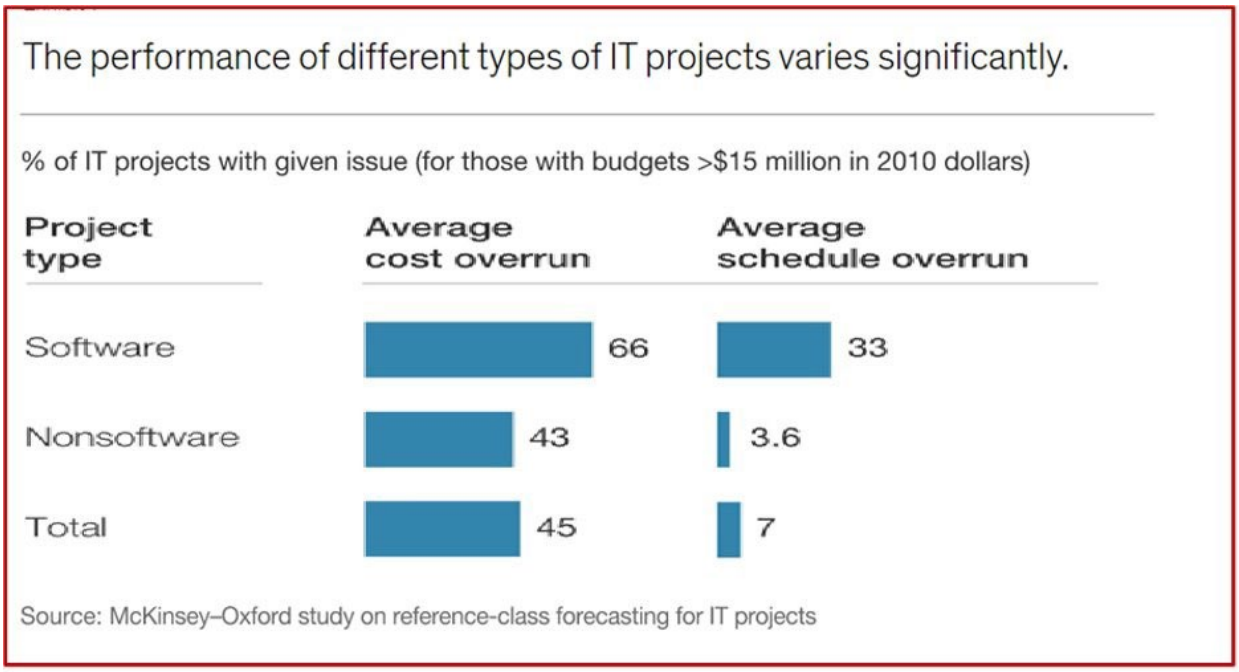


Figure 2: Software Cost Within the IT Industry

Limitations of Existing Estimating Tools

In response to the need for better software estimating capability, a variety of approaches have been developed over the years, including parametric models (e.g., COCOMO), commercial estimating tools, and sizing frameworks (e.g., function point counting). Based on our review of these approaches, as well as professional estimating experience, we have observed the following limitations within the software estimating community:

- Over-reliance on a single sizing metric (i.e., SLOC)
- Parametric models do not expose underlying data (most commercial models)
- Estimating methods do not include a data source (function point counting)
- Cost estimating relationships (CERs) often do not accommodate missing input data
- Methods are based on limited or no historical data

We believe that many of these limitations can be addressed through the use of an appropriate data set, methods that don't rely on any single input value or sizing metric, and methods that are transparent in both the source data as well as underlying equations.

ISBSG as a Data Source

The dataset that forms the basis of SWEET is from the International Software Benchmarking and Standards Group (ISBSG) data repository (https://www.isbsg.org). The repository contains software project data across the IT industry. Data are submitted to ISBSG by trusted international IT organizations, including both private sector and government sponsored projects. ISBSG was selected for SWEET because it contains:

- *A large volume of projects*. ISBSG's software development repository contains 10,600 projects, dating from 1989 to 2020.
- *A variety of descriptive variables (fields)* on each project. A total of 252 fields are available. Of these, 105 are quantitative, and 147 are qualitative.
- A variety of fields that describe *software size*. Although not all metrics are available for each project, ISBSG allows for the reporting of software size based on IFPUG function points, COSMIC points, SLOC, and a variety of less prevalent metrics.
- Multiple *development methodologies*. Fields are included which indicate whether each project was developed according to agile, incremental, waterfall, or a number of other possible development methodologies.
- Fields that report on the *work effort*, measured in person-hours, of the development team.
- *A data quality rating*, which contains an ISBSG-assigned letter grade, A through D. For purposes of this analysis, we excluded data that was rated D.
- It is *available*, through purchase of a subscription, to the general public. Details are available at https://www.isbsg.org/isbsg-subscriptions/.

It is important to note one limitation of the ISBSG dataset, specifically that not all of the available fields are populated for all projects. For example, adjusted function points are reported for approximately 71% of projects and number of users are reported for only 8% of projects. As a result, the amount of data available for analysis is reduced, possibly significantly, depending on the type and number of fields chosen for analysis.

Analytical Approach

ISBSG Dataset

The large amount of data presented in ISBSG required considerable preliminary analysis to focus our effort on the most suitable variables for software effort estimation.

Data covered by the database include new development, enhancement and re-factoring or re-development effort. The data are dominated by enhancements, which constitute about seventy percent of the dataset. Most of the remaining data points are new development efforts. About 16.5% of the data are from the 1990s, 43.5% from the 2000s, and the remaining 40% from the 2010s. Table 2 shows the allocation of data by decade and category.

Table 2: ISBSG Data Points by Decade and Project Type

Decade	Number of Data Points
1990s	1576
2000s	4145
2010s*	3804

Development Type	Number of Data Points
Enhancement	6620
New Development	2806
Re-Development	104

Figure 3 provides a more detailed characterization by year and project type.

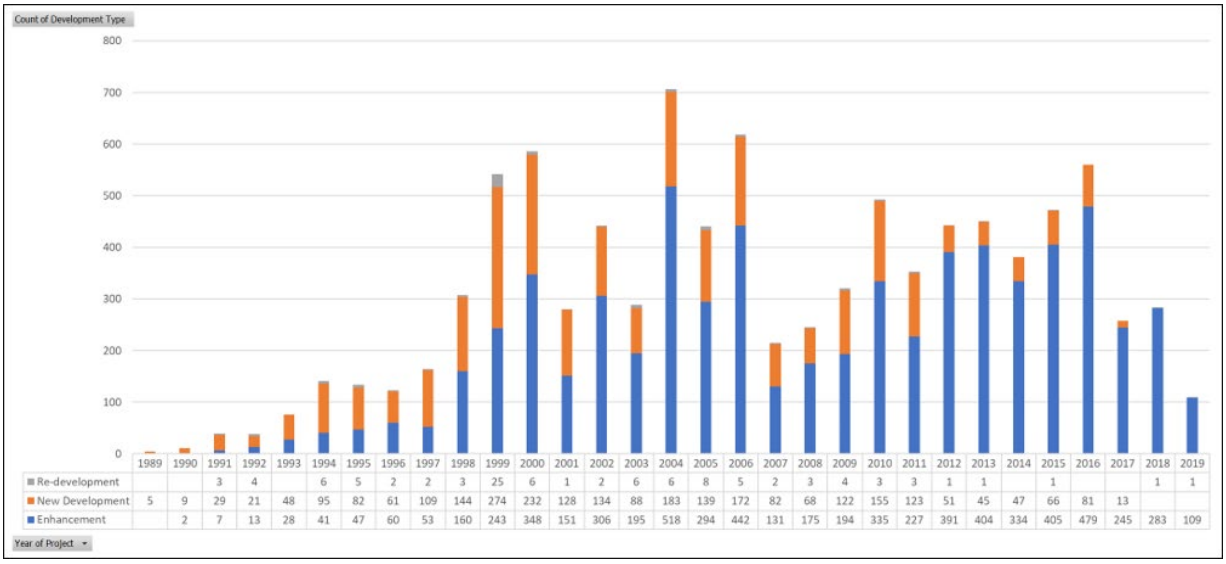


Figure 3: ISBSG Data Points by Year and Project Type

The ISBSG data includes a variety of function point counting methods, where IFPUG 4+, NESMA, and COSMIC represent 98.6% of the data. The spread of that data over time is shown in Figure 4 and Figure 5 below.

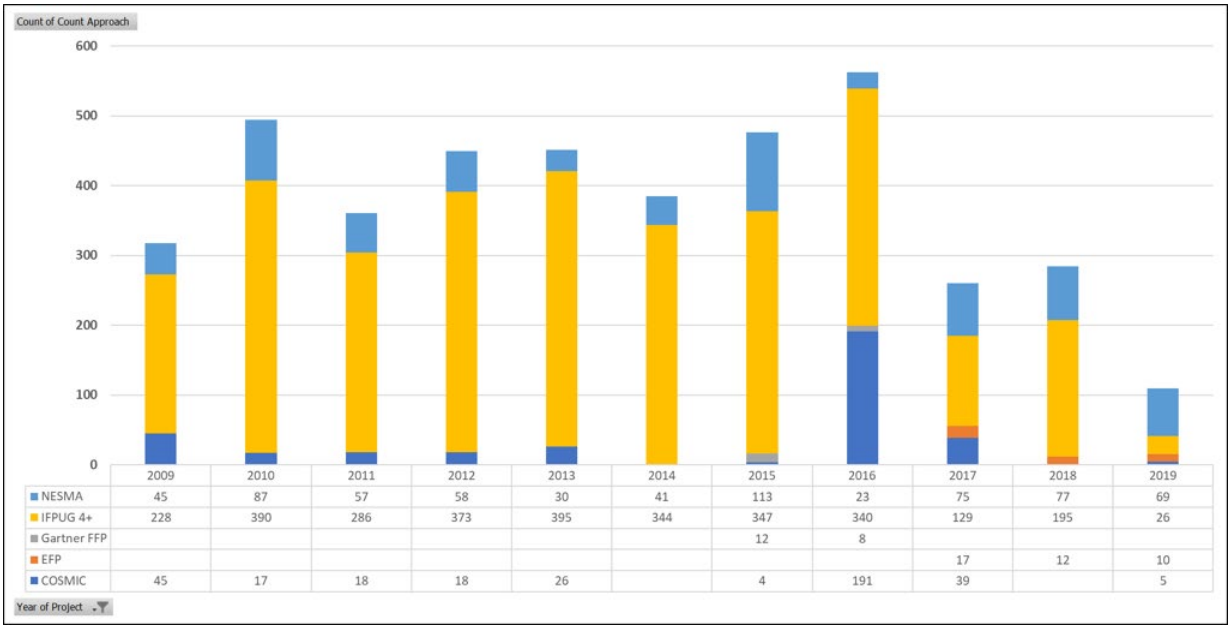


Figure 4: ISBSG Data Points by Function Point Sizing Methods (2009-2019)

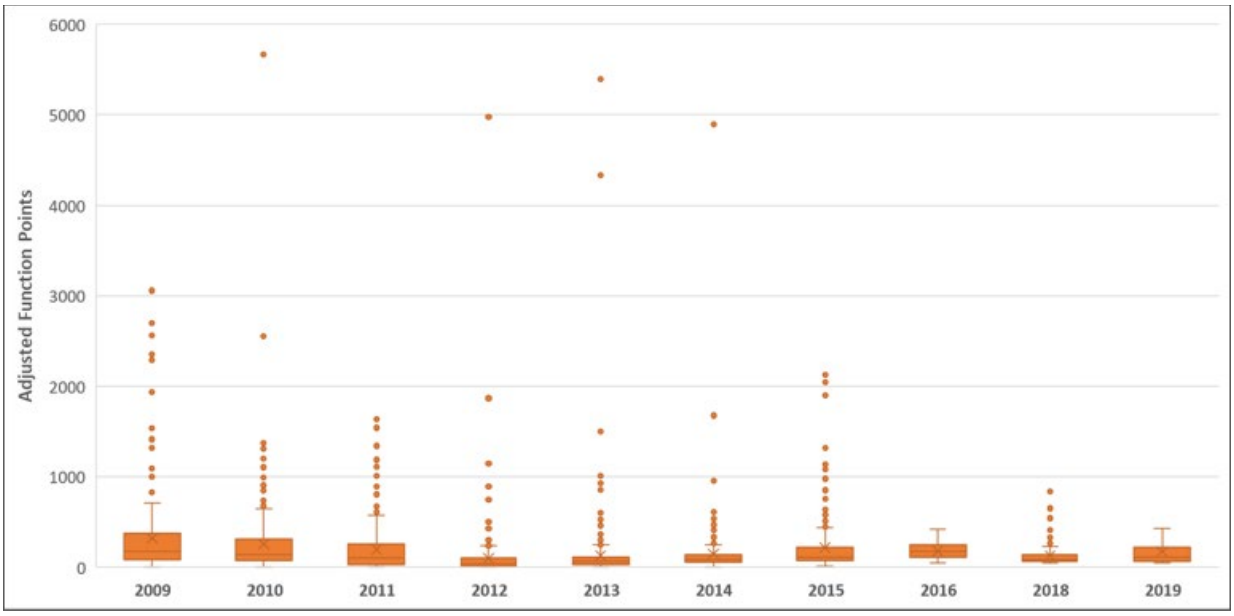
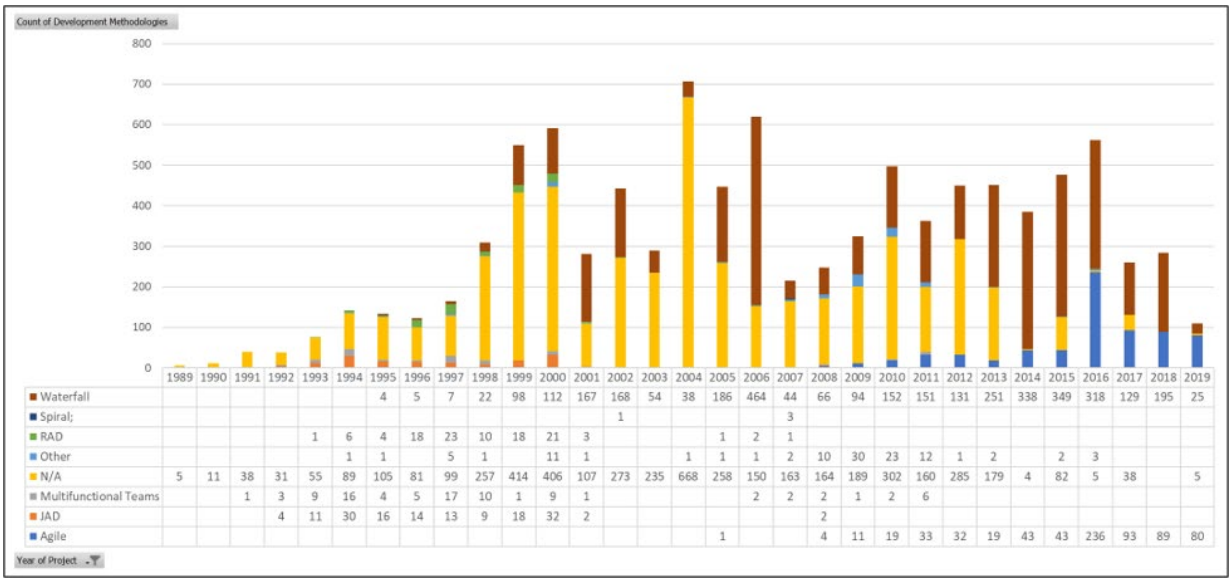


Figure 5: Number of Adjusted Function Points in ISBSG Projects (2009-2019)

Waterfall is the most frequent software development methodology represented in ISBSG, representing 3,568 data points. Agile development first appears in 2005, and increases in the following years. Figure 6 below shows the type of development efforts in the database by year.



Year of Project	1989	1990	1991	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002	2003	2004	2005	2006	2007	2008	2009	2010	2011	2012	2013	2014	2015	2016	2017	2018	2019
Waterfall							4	5	7	22	98	112	167	168	54	38	186	464	44	66	94	152	151	131	251	338	349	318	129	195	25
Spiral;														1					3												
RAD					1	6	4	18	23	10	18	21	3				1	2	1												
Other						1	1		5	1		11	1			1	1	1	2	10	30	23	12	1	2		2	3			
N/A	5	11	38	31	55	89	105	81	99	257	414	406	107	273	235	668	258	150	163	164	189	302	160	285	179	4	82	5	38		5
Multifunctional Teams			1	3	9	16	4	5	17	10	1	9	1					2	2	2	1	2	6								
JAD				4	11	30	16	14	13	9	18	32	2							2											
Agile																	1			4	11	19	33	32	19	43	43	236	93	89	80

Figure 6: ISBSG Data Points by Development Methodologies Over Time

Productivity Factor Analysis

After our initial analysis of the ISBSG dataset, we recognized that we had sufficient data points to proceed using function points as a sizing metric. First, we wanted to analyze productivity factors, which is a measurement of how much code can be developed in a given time frame. Our analysis generated factors expressed as function points (FPs) per hour.

To accomplish this, we established a subset of the data by stratifying the data as follows. We identified various sizing methods that are included in the ISBSG dataset and removed non-function points methods (e.g., 'other', Lines of Code (LOC)). We also removed any projects with D or lower Quality Grades and established a data age restriction of 25 years old or newer. For each project in this subset, we calculated a productivity factor by dividing the Adjusted Function Points by Normalized Work Effort to get a measurement of FP/hour. Adjusted Function Points is functional size adjusted by the Value Adjustment Factor (VAF), and Normalized Work Effort is the full lifecycle effort for all teams reported (not just the development team).

Next, we evaluated histograms of the productivity factors for this entire subset of data, as well as smaller subsets by relative size. The productivity histogram is shown in Figure 7 below.

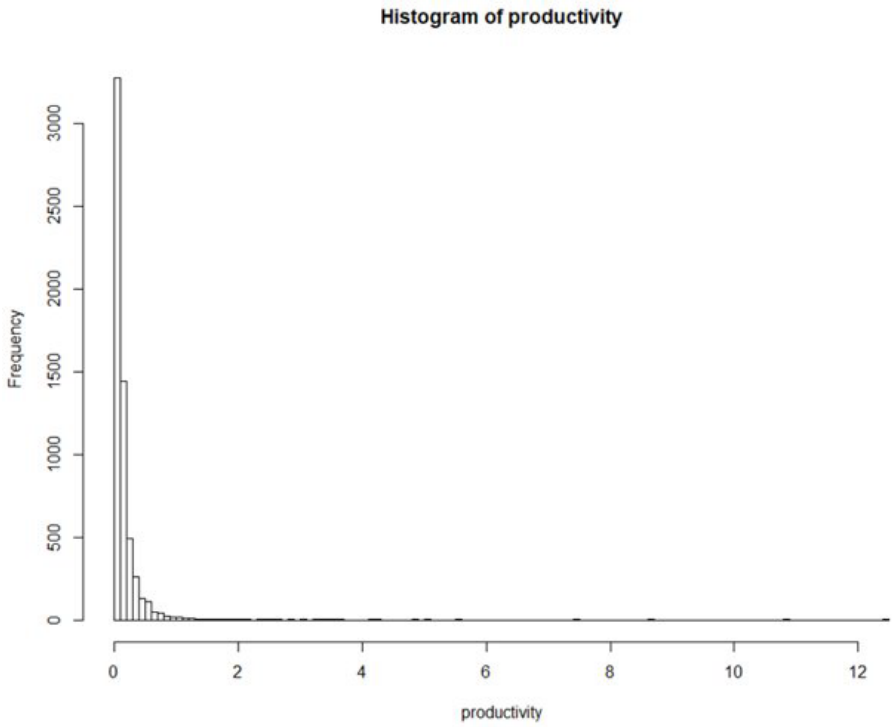


Figure 7: Distribution of Productivity (Overall)

For the total productivity data subset (5914 data points), the histogram skews right, with a median of 0.087 FPs/hour and a mean of 0.163 FPs/hour.

Next, we wanted to determine whether the relative size of software development projects has an effect on the productivity. The productivity dataset was further sub-divided by project size: small (less than 100 FPs); medium (100 to 1000 FPs); large (1000 to 9000 FPs); and extra-large (more than 9000 FPs). These histograms are presented in Figure 8, Figure 9, Figure 10, and Figure 11 below.

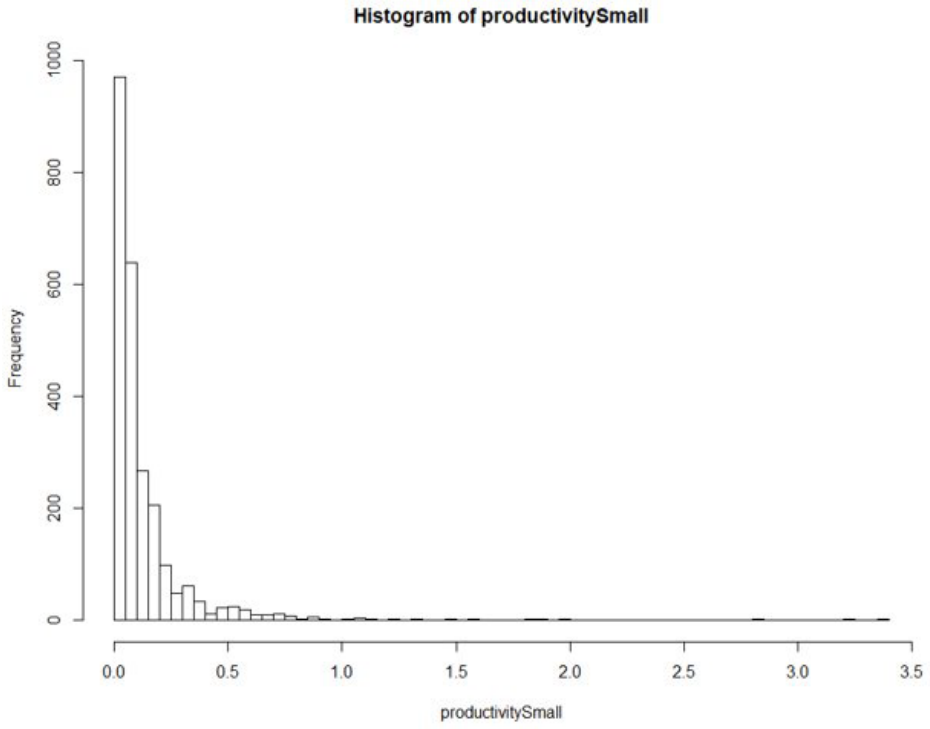


Figure 8: Distribution of Productivity (Small Projects)

For the 2448 small software development projects, the histogram again shows right skew, with a median of 0.068 FPs/hour and a mean of 0.121 FPs/hour.

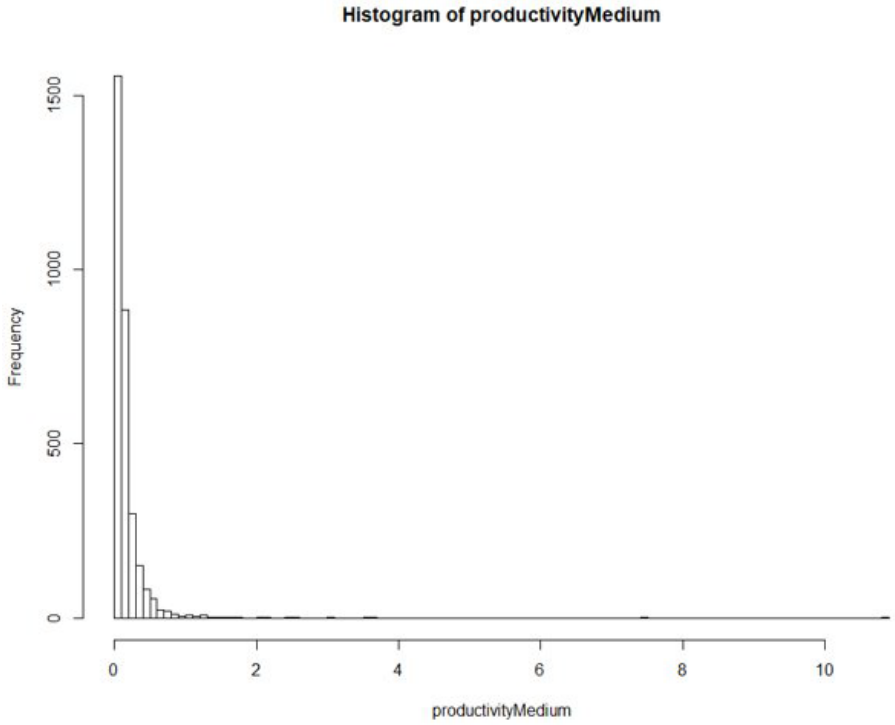


Figure 9: Distribution of Productivity (Medium Projects)

For the 3124 medium-sized projects, the histogram skews right, with a median of 0.101 FPs/hour and a mean of 0.167 FPs/hour.

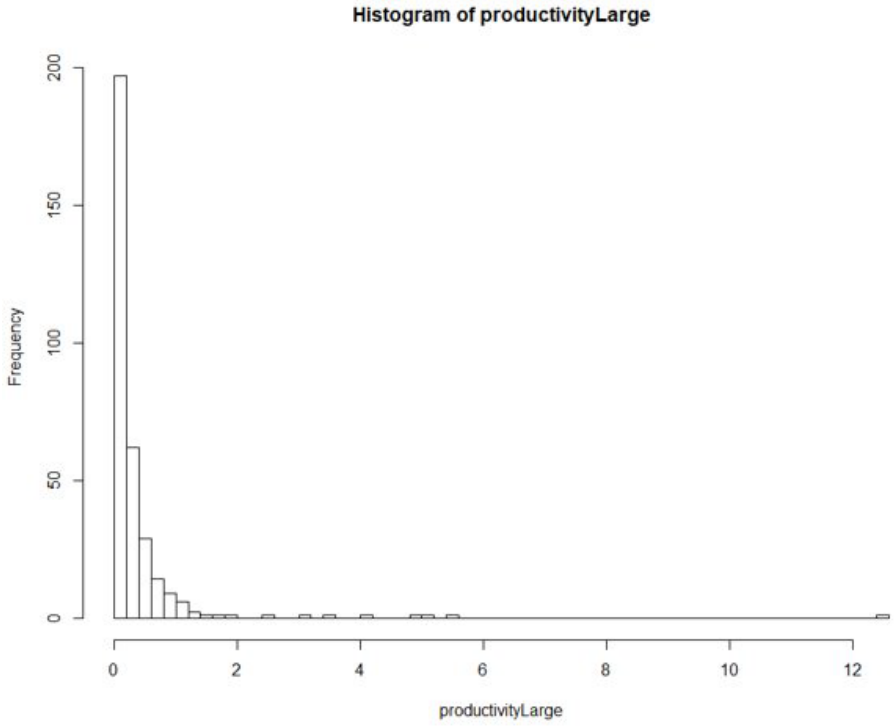


Figure 10: Distribution of Productivity (Large Projects)

For the 330 large projects, the histogram again shows right skewness with a median of 0.144 FPs/hour and a mean of 0.373 FPs/hour.

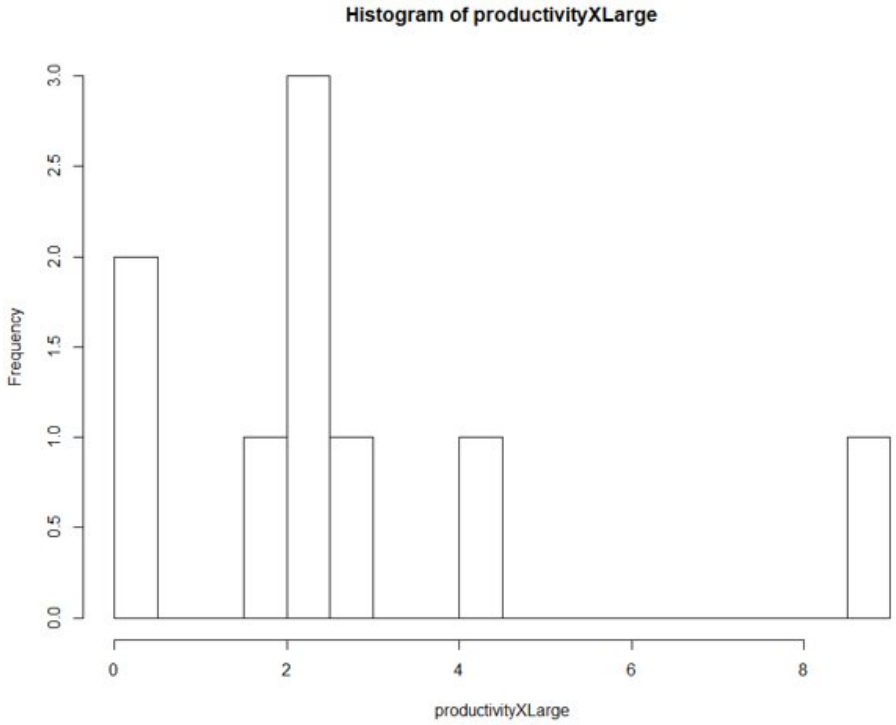


Figure 11: Distribution of Productivity (Extra-Large Projects)

Finally for the nine extra-large software development projects, the histogram is also skewed right. The median is 2.16 FPs/hour and the mean is 2.71 FPs/hour. The median and mean for extra-large projects show a significant increase in productivity over small, medium, and large projects; however, there are significantly fewer data points for extra-large projects.

Still, as the relative size of the software development project increases, the median and mean productivity factors increase, suggesting that software teams become more efficient as the project size increases, thereby achieving *economies of scale*.

Function Point Growth Analysis

After analyzing the effect of relative project size on productivity, we analyzed code growth for projects using function point sizing methods. To do this, we used a subset of data where the function point count approach was equal to the function point size

estimate approach. This was done to ensure that a change in the count methodology wasn't driving any growth.

First, we calculated a function point growth factor for each project using the formula,

$$FP\ Growth = \frac{Functional\ Size\ -\ Size\ Estimate}{Size\ Estimate}$$

where: functional size is the actual number of function points and size estimate is the estimated number of function points for each project. As such, we define function point growth as the ratio of the difference between the actual size and the estimated size to the estimated size.

Next, we calculated average function point growth factors for different subsets of the data segmented by variables such as: Count Approach, Development Methodology, Development Type, Language Type, and Relative Size.

We noticed that the majority of software projects had zero growth, meaning that the number of estimated function points was equal to the number of actual function points. Unfortunately, we are unable to determine if this is due to consistent requirements throughout the program or improper reporting of the data. Pending verification, we would usually recommend using the right-hand side ("Excluding Zero Code Growth") of the following tables, but the drastic reduction of the dataset is cause for concern.

Table 3: Code Growth by Count Approach

	Including Zero Code Growth		Excluding Zero Code Growth	
	Mean Code Growth Factor	Number of Data Points	Mean Code Growth Factor	Number of Data Points
IFPUG	2.8%	1129	17.9%	176
NESMA	1.3%	246	22.6%	14
COSMIC	0.5%	255	40.9%	3
EFP	0.0%	39	0.0%	0

For Count Approach, we noticed that the rankings of most-to-least code growth changes when excluding projects where the estimated size equaled the actual size. As expected, code growth factors increased as the number of data points decreased when we removed the projects with zero code growth.

Table 4: Code Growth by Development Methodology

	Including Zero Code Growth		Excluding Zero Code Growth	
	Mean Code Growth Factor	Number of Data Points	Mean Code Growth Factor	Number of Data Points
Agile Development;	0.1%	549	76.4%	1
Waterfall (incl Linear Processing & SSADM);	0.7%	769	31.4%	18
Incremental;	44.5%	29	117.4%	11
Interactive;	20.2%	1	20.2%	1
Unified Process;	12.2%	2	12.2%	2
Timeboxing;	-2.7%	16	-14.3%	3
Iterative;	6.9%	2	6.9%	2
Multifunctional Teams;	20.4%	5	34.0%	3
Joint Application Development (JAD);	41.7%	3	41.7%	3
Rapid Application Development (RAD);	-11.6%	2	-23.3%	1
IT Unified Process (ITUP);	-3.3%	1	-3.3%	1
RUP;	0.0%	1	0.0%	0
OCE;	-43.5%	2	-87.0%	1

For Development Methodology, the general trends were relatively similar when excluding the projects with zero code growth, but Agile and Waterfall code growth increased significantly with a large decrease in the number of data points. As expected, all positive code growth factors increased and all negative code growth factors decreased.

Table 5: Code Growth by Development Type

	Including Zero Code Growth		Excluding Zero Code Growth	
	Mean Code Growth Factor	Number of Data Points	Mean Code Growth Factor	Number of Data Points
Conversion	0.0%	28	0.0%	0
Enhancement	2.3%	1220	26.7%	105
Mix	0.0%	2	0.0%	0
New Development	1.6%	408	8.2%	81
Re-Development	10.7%	11	16.8%	7

For Development Type, the general trends were relatively similar when excluding projects with zero code growth. Again, as expected, all code growth factors increased with the decrease in data points.

Table 6: Code Growth by Language Type

	Including Zero Code Growth		Excluding Zero Code Growth	
	Mean Code Growth Factor	Number of Data Points	Mean Code Growth Factor	Number of Data Points
2GL	0.0%	1	0.0%	0
3GL	2.0%	875	15.0%	119
4GL	3.3%	556	36.3%	50
ApG	27.2%	12	32.6%	10

For Language Type, the general trends were relatively similar and all code growth factors increased when the software projects with zero growth were excluded.

Table 7: Code Growth by Relative Size

	Including Zero Code Growth		Excluding Zero Code Growth	
	Mean Code Growth Factor	Number of Data Points	Mean Code Growth Factor	Number of Data Points
XXS	0.0%	128	0.0%	0
XS	-0.3%	158	-9.6%	5
S	0.7%	405	8.9%	30
M1	1.8%	446	12.9%	62
M2	5.7%	389	28.6%	78
L	2.2%	123	18.0%	15
XL	4.5%	13	29.0%	2
XXL	1.7%	6	10.4%	1
XXXL	0.0%	1	0.0%	0

For Relative Size, again the general trends were relatively similar, and all positive code growth factors increased and negative code growth factors decreased when the software projects with zero growth were removed. Here we also saw a general trend that as the relative size of the software project increases, the code growth also increases. The smaller software projects generally had less code growth and, in some cases, had negative code growth where the actual project size was smaller than estimated. This runs counter to the “size effect” often encountered in risk, wherein larger efforts tend to grow by a larger absolute amount but a smaller percentage.

Independent and Dependent Variable Identification

After completing the preliminary ISBSG dataset analysis, we determined the independent and dependent variables that would provide the best software development estimations. First, we identified variables in the ISBSG dataset that represented logical candidates for a parametric estimating relationship.

We identified a handful of potential data fields that could be dependent variables, including size, cost, and effort. For size, this would be the amount of code required to complete the software development project, either in adjusted function points or SLOC. For cost, this would be the actual cost of the software development. For effort, this would be the number of hours required to complete the software development project.

Ultimately, we chose Normalized Work Effort expressed in hours as our dependent variable. We preferred this measure because it represents a cost estimating output rather than an input, such as function points or SLOC. Furthermore, an estimate expressed in hours enables use of developer-specific labor rates. Additionally, there

were significantly more data points with reported values for Normalized Work Effort vs. Total Project Cost (8,657 vs. 1,826). At the same time, Total Project Cost has a significant disadvantage as a dependent variable because it introduces exchange rate risk. There are 19 different currencies included in the data set, and only 741 of the 1,826 data points for Total Project Cost were in US dollars.

Next, we identified several potential independent variables. We made sure to use variables that represented actuals (vs. estimates) that could be drivers of the dependent variable, effort. We also wanted to include variables that a cost analyst should know pre-software development. Additionally, we made sure that the number of potential options for each variable was not large enough to affect statistical analysis. For example, there are 151 different programming languages in the ISBSG dataset. To mitigate this issue, we grouped seldom-used categories into an “other” category.

Prior to analyzing the potential independent variables, we also normalized the independent variable inputs. The ISBSG qualitative data fields are open inputs, which result in inconsistencies. We reviewed all of the independent variable inputs to ensure consistency (e.g., “Multi-tier” vs. “Multi tier”). Seldom-used inputs were classified as “other” to simplify the Effort Estimating Relationship (EER) and minimize the impact to the degrees of freedom.

Finally, we tested each of the 26 potential independent-to-dependent variable correlations using Spearman rank correlation. Table 8 shows a subset of the results from the Spearman correlations between the independent variables and Normalized Work Effort, the dependent variable.

Table 8: Significant Effort Drivers

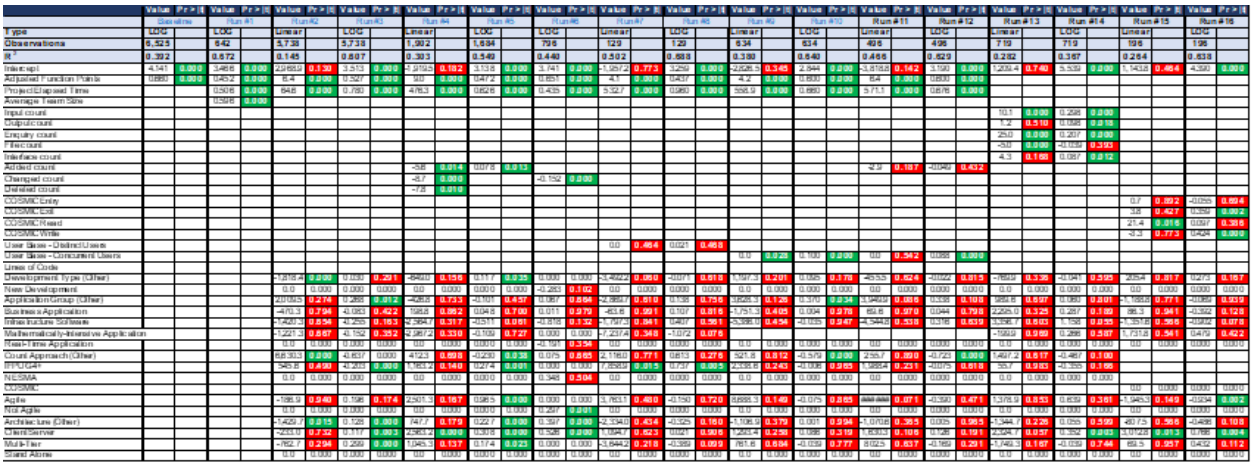
Effort to:	Coefficient	P-Value	Determination	n
Adjusted Function Points	0.620	0.0000000	0.385	6,660
Project Elapsed Time	0.554	0.0000000	0.307	8,446
Average Team Size	0.558	0.0000000	0.312	1,274
Input count	0.498	0.0000000	0.248	1,794
Output count	0.417	0.0000000	0.174	1,794
Enquiry count	0.475	0.0000000	0.226	1,776
File count	0.394	0.0000000	0.156	1,794
Interface count	0.252	0.0000000	0.064	1,776
Added count	0.547	0.0000000	0.299	2,291
Changed count	0.450	0.0000000	0.203	2,114
Deleted count	0.183	0.0205630	0.033	2,114
COSMIC Entry	0.673	0.0000000	0.453	258
COSMIC Exit	0.715	0.0000000	0.511	258
COSMIC Read	0.713	0.0000000	0.508	258
COSMIC Write	0.711	0.0000000	0.506	258
User Base - Distinct Users	0.327	0.0000000	0.107	672
User Base - Concurrent Users	0.270	0.0000000	0.073	797
Lines of Code	0.692	0.0000000	0.478	704

Effort Estimating Relationship (EER) Development

After choosing the dependent variable and finalizing the list of 26 potential independent variables, we created a baseline EER to compare with potential candidate EERs. This EER used Normalized Work Effort as the dependent variable and Adjusted Function Points as the independent variable. The baseline EER generated an R^2 of 0.3924.

Next, we created more than 50 candidate relationships for estimating software development effort. Different groups of independent variables were used for each run. Since every data field (i.e., variable) is not available for every observation, each EER is based on a different number of observations depending on the specific independent variables tested. Table 9 below shows the summary matrix used to compare all of the EERs.

Table 9: EER Comparison



After each run, we would assess the results and add new inputs that improved curve fit, such as CMM level and age of data. We also removed prospective inputs that didn't show statistical significance, such as *Application Group* and *Architecture*. We also removed *Total Elapsed Time*. While this variable showed significance and an increase in R^2, we were concerned with collinearity as well as inaccurate/aggressive schedule estimates early in a program's life cycle.

This EER exploration process generated numerous questions and provided numerous hard-won insights. While there are mechanical methods such as stepwise regression to cycle through various combinations of variable, we found significant value in going through the manual process, especially with as variegated a dataset as ISBSG. Rather than giving an analyst a proverbial fish – our preferred "off the shelf" EER – what if we could teach the analyst to fish – providing them the capability to generate their own EERs with transparency and traceability? That's where SWEET comes in!

Tailoring EERs

In order to assist cost analysts with EER development and analysis, we developed an effort estimating interface that allows the analyst to filter the dataset based on inputs available at the time an estimate is needed, such as early in the development or before contract award. We also wanted to be able to provide an option to remove data points, such as outliers or non-analogous projects, via a quick input. We decided a dynamic

EER development tool that runs without the use of VBA would help cost analysts develop reliable and defensible software development effort estimates. This vision resulted in development of the Software Effort Estimation Tool (SWEET).

We believe SWEET offers these benefits to the cost analyst:

- Custom EER generation, based on analyst-selected inputs and analyst-selected data.
- Dynamic (real time) recalculation of EER results.
- Transparent equations and transparent data. This, along with the use of an industry-leading dataset, enables the cost analyst to produce highly defensible estimates.

Software Effort Estimation Tool (SWEET)

Tool Overview

SWEET is a prototype Excel-based effort estimation tool that is built upon the analytical work discussed above and leverages the rich ISBSG data set. *The core tenet of SWEET is that it allows the analyst to dynamically build an EER based on analyst-selected inputs and analyst-selected data.* To do this, SWEET has three key interfaces: Inputs, Data Selection, and Results, and also includes a SLOC-to-FP conversion capability.

Tool Inputs

The Inputs screen as shown in Figure 12 below is built on a set of inputs that were determined to be characteristics that an analyst would either know or be able to determine in the course of understanding the software development effort to be estimated.

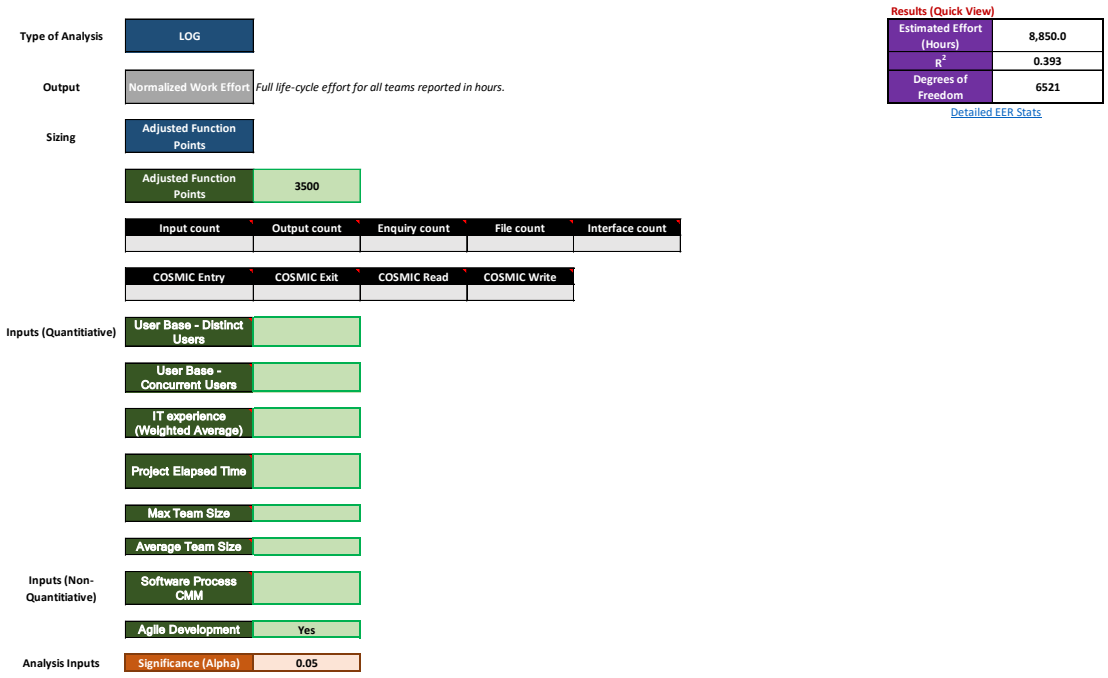


Figure 12: SWEET Inputs Screen

The tool uses a color-coding system to define how the analyst interacts with the various inputs as defined below.

- Blue: Settings, which can be adjusted by the user as needed. Additional options can be selected from a drop-down menu.
- Gray: Output variable. There is only one option; user cannot change variable.
- Green: Active user inputs.
- Black: Inactive user inputs. These cells will change color depending on the sizing method selected (in blue). For example, if the user changes the sizing method to COSMIC Data, then the Adjusted Function Points cells will change to black, and the COSMIC Data cells will change to green.
- Orange: User input for statistical significance.

The majority of the SWEET inputs are not required to generate an estimate but are used to help establish a clearer view of the estimate in question over time. Additionally, the choice of inputs used is one of the main drivers of the dynamic EERs that the tool generates. The list below defines the various elements that were selected and some insights into how or when they should be used.

- **Type of Analysis (*required*)**: The user may select LOG or Linear in this drop-down box. The selection determines the mathematical formulation used to derive the EER used by the tool, Log Ordinary Least Squares (LOLS) or Linear Ordinary Least Squares (OLS), respectively. SWEET defaults to LOG, which is the recommended analysis method. The advantage of a LOG model is that it can better accommodate economies or diseconomies of scale within the data by way of a power function and multiplicative indicator variables. However, some users may prefer the simplicity of a linear model. The user may also wish to try each choice and select the option that produces the best result.
- **Sizing (*required*)**: The user must select the method used to input software size. Software size is a measure of the volume of software required to perform a function or set of functions. The tool allows for three alternatives: Adjusted Function Points, Function Point Categories, or COSMIC Data. The tool also allows the user to input size as Source Lines of Code (SLOC), and then convert to Adjusted Function Points. When a sizing method is selected, the tool enables the one of the following sets of input variables:
    - **Choice of Sizing Method**: Select one of the following from the drop-down menu.
    - **Adjusted Function Points**: This input allows the user to enter the software size, measured in Adjusted Function Points. Function points are a unit of measure representing the business functionality being developed. There are several standards that can be used when conducting a function point count,

including IFPUG, COSMIC, and Simple Function Points. (For more information on sizing methods, see the References section.)

- **Function Point Categories**: This input method allows the user to separately enter each component of a function point count. This method should be used when details behind the function point count are known, and can be decomposed using the five variables listed below.
  - Input Count
  - Output Count
  - Enquiry Count
  - File Count
  - Interface Count
- **COSMIC Inputs**: This input method allows the user to separately enter each component of a COSMIC count. This method should be used when details behind the COSMIC count are known, and can be decomposed using the four variables listed below.
  - COSMIC Entry
  - COSMIC Exit
  - COSMIC Read
  - COSMIC Write
- **SLOC**: If the user wishes to enter size based on Source Lines of Code (SLOC), then the tool allows for a conversion from SLOC to Function Points. The tool tab titled "(OPTIONAL) SLOC to FP Converter" allows the user to enter Project SLOC, Programming Language, and SLOC per FP Factor. The tool applies the included backfiring table to produce an adjusted function point count, which can then be entered into the Inputs tab.

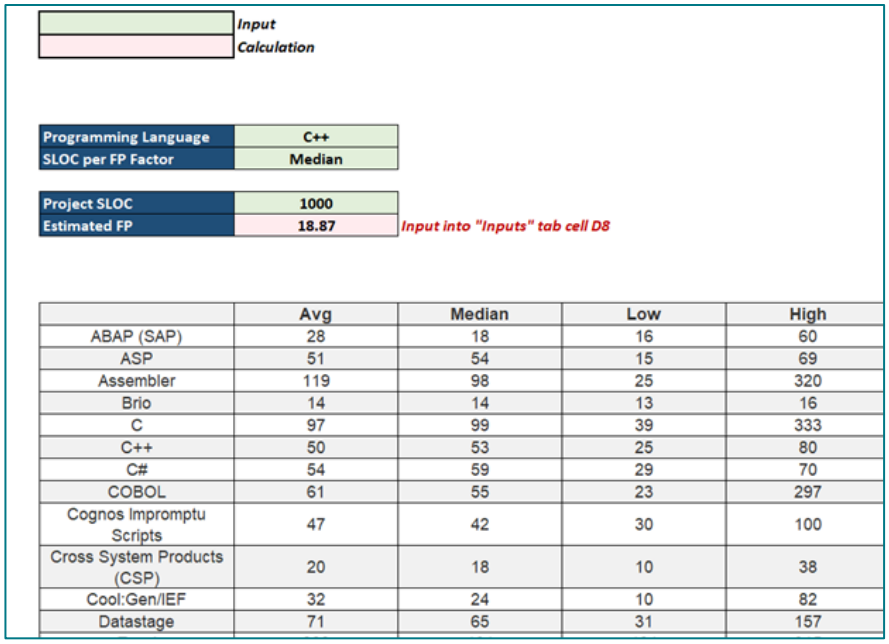
- **User Base – Distinct Users (*optional*)**: This optional input allows the user to enter the anticipated number of distinct users, which is an indication of how heavily the system will be utilized, as well as a measure of the complexity and diversity of the software's functional requirements.
- **User Base – Concurrent Users (*optional*)**: This optional input allows the user to enter the number of concurrent users. Unlike the distinct users input, concurrent users measures how many people are using the system at the same time.
- **IT Experience (weighted average) (*optional*)**: This optional input allows the user to enter an average number of years of software development experience among the software developers. Higher numbers, which indicate a more experienced team, are generally associated with higher productivity and therefore less overall development effort.
- **Project Elapsed Time (*optional*)**: This optional input allows the user to enter the anticipated development time, expressed in calendar months. In cases where the development project is not constrained by schedule, or the schedule is not known, the user would leave this input blank.
- **Max Team Size (*optional*)**: This optional input allows the user to enter the anticipated maximum (peak) software development team size. In cases where the max team size is not constrained, or the max team size is not known, the user would leave this input blank.
- **Average Team Size (*optional*)**: This optional input allows the user to enter the anticipated average software development team size. In cases where the average team size is not constrained or the average team size is not known, the user would leave this input blank.
- **Software Process CMM (*optional*)**: This optional input allows the user to select a value that indicates the capability of the software development organization, as measured by the Software Development Capability Maturity Model (CMM). If this

value is known, the user may indicate either a low level of maturity by selecting "Level<=2" or a high level of maturity by selecting "Level 3+". Although the full CMM scale ranges from 1 to 5, analysis of ISBSG source data indicates that the best statistical relationship is present when viewing CMM level in these larger categories.

- **Agile Development (*optional*)**: This optional input allows the user to specify whether the software development team will follow an agile development process. By selecting "Yes", results will be adjusted to account for efficiencies within the Agile Development process that are present within the source data of the tool.

The Inputs screen also includes a quick results box which provides the analyst with the estimated effort based on the inputs that have been entered. Additionally, it provides the R^2 Value and Degrees of Freedom for the dynamic EER and the estimate generated using the EER. An analyst has a quick view of the results so they can determine if they wish to enter further inputs or conduct further analysis to refine their existing inputs.

As discussed above, it is acknowledged that not every effort is sized in Function Points. For this reason, SWEET includes a SLOC to FP conversion tab to enable the analyst to convert SLOC counts to function points as shown in Figure 13.



	Input
	Calculation

Programming Language	C++
SLOC per FP Factor	Median

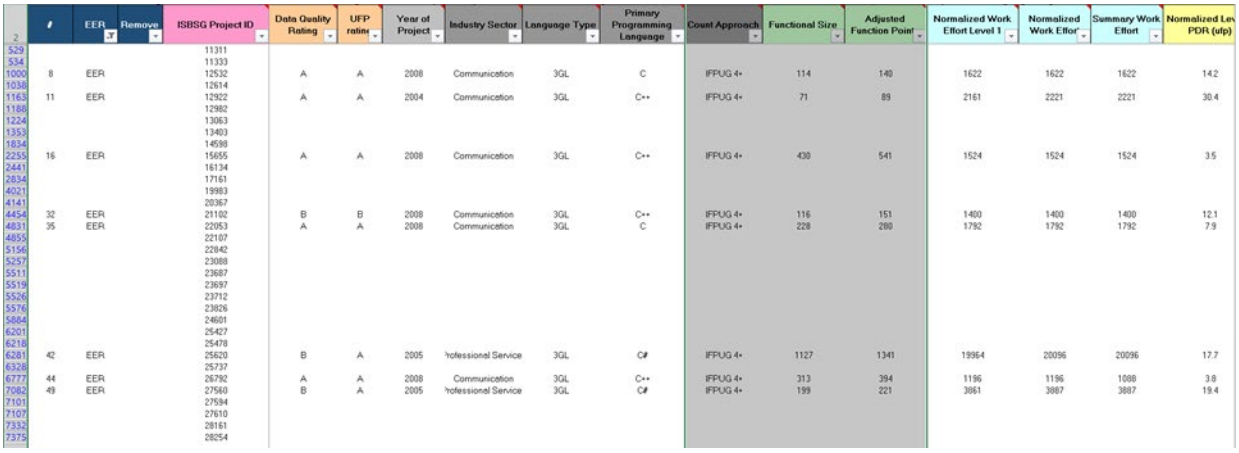
Project SLOC	1000	
Estimated FP	18.87	*Input into "Inputs" tab cell D8*

	Avg	Median	Low	High
ABAP (SAP)	28	18	16	60
ASP	51	54	15	69
Assembler	119	98	25	320
Brio	14	14	13	16
C	97	99	39	333
C++	50	53	25	80
C#	54	59	29	70
COBOL	61	55	23	297
Cognos Impromptu Scripts	47	42	30	100
Cross System Products (CSP)	20	18	10	38
Cool:Gen/IEF	32	24	10	82
Datastage	71	65	31	157

Figure 13: SLOC Conversion in SWEET

The conversion capability leverages the conversion table published by QSM on their website. (SLOC per FP, n.d.)

The Data Selection interface, depicted in Figure 14, shows filtered projects based on inputs entered by the analyst, with each project on a separate row. The columns in this view show a wide range of data and characteristics that are known about each project.



#	EER	Remove	ISBSG Project ID	Data Quality Rating	UFP rating	Year of Project	Industry Sector	Language Type	Primary Programming Language	Count Approach	Functional Size	Adjusted Function Point	Normalized Work Effort Level 1	Normalized Work Effort	Summary Work Effort	Normalized Lev PDR (ufp)
529			11311													
534			11333													
1000	8	EER	12532	A	A	2008	Communication	3GL	C	IFPUG 4+	114	140	1622	1622	1622	14.2
1038			12614													
1163	11	EER	12922	A	A	2004	Communication	3GL	C++	IFPUG 4+	71	89	2161	2221	2221	30.4
1188			12982													
1224			13063													
1353			13403													
1834			14598													
2255	16	EER	15655	A	A	2008	Communication	3GL	C++	IFPUG 4+	430	541	1524	1524	1524	3.5
2441			16134													
2834			17161													
4021			19983													
4141			20367													
4454	32	EER	21102	B	B	2008	Communication	3GL	C++	IFPUG 4+	116	151	1400	1400	1400	12.1
4831	35	EER	22053	A	A	2008	Communication	3GL	C	IFPUG 4+	228	280	1792	1792	1792	7.9
4855			22107													
5156			22842													
5257			23088													
5511			23687													
5519			23697													
5526			23712													
5576			23826													
5884			24601													
6201			25427													
6218			25478													
6281	42	EER	25620	B	A	2005	Professional Service	3GL	C#	IFPUG 4+	1127	1341	19964	20096	20096	17.7
6328			25737													
6777	44	EER	26792	A	A	2008	Communication	3GL	C++	IFPUG 4+	313	394	1196	1196	1088	3.8
7082	49	EER	27560	B	A	2005	Professional Service	3GL	C#	IFPUG 4+	199	221	3861	3887	3887	19.4
7101			27594													
7107			27610													
7332			28161													
7375			28254													

Figure 14: Data Selection in SWEET

Within the Data Selection tab, the analyst can refine the dataset being used by SWEET to develop the EER for the development effort in question. Because there is a wide variety of data in each row of the Data Selection tab, there are a practically infinite number of ways the EER could be customized. Here are a few examples of how an analyst might further refine the data set behind the EER:

- If the project is known to support a specific industry sector, such as "Government," the user may consider removing projects from different industry sectors.
- If the project is known to use a specific programming language such as Java, the user may consider removing projects that are developed with different languages.
- If the project is known to use a specific system architecture such as Client-Server, the user may consider removing projects that implement different system architectures.

Tool Outputs

The Results screen as shown in Figure 15 below provides the analyst with the results of the estimate generated by the EER that was created for the development effort in question, along with descriptive and inferential statistics that can be used to validate the method.

Results (Detailed Stats)

	1	2	3			
	2	1	Intercept			
	Agile (0 - Not Agile, 1 - Agile)	LN(Adjusted Function Points)	Intercept			
Coefficient	-0.440	0.660	4.144			
SE	0.177	0.010	0.052			
R^2 / SEE	0.393	1.102				
F / dF	2108.972	6521				
SSR / SSE	5120.617	7916.545				
T-stat	-2.487	64.898	79.335			
P-value	0.0129	0.0000	0.0000			

EERT Calculation

Estimated Effort (Hours)	8,850

EER

Log Form	Normalized Effort (hrs) = e^(4.14) * Adjusted Function Points^0.66* (e^(-0.44))^Agile Development
Linear Form	

Figure 15: EER Results in SWEET

The statistics are provided for each of the inputs that were chosen by the analyst with the intent of enabling a quick and thorough assessment of the overall validity of the estimate and whether there are any inputs that need further examination. The purple Estimated Effort box provides the analyst with the results of the EER in hours. The EER section provides the analyst with the equation for the EER to make it clear how the effort estimate was derived.

All of these outputs give the analyst maximum transparency into the effort estimate that is generated by SWEET, from the data being used all the way through the actual EER that produces the estimate.

Active Analysis

One of the benefits of SWEET is that it allows the user to see real-time impacts of changing inputs and dynamically updates the underlying dataset, the EER, *and* the effort estimate. The example presented below shows how this process might play out

as more information becomes known over time and at different stages of the development lifecycle at which the analyst would be developing an estimate.

Example Analyses

SWEET can generate an estimate based solely on a sizing input and user selection of development methodology, specifically agile or not. The figure below shows the Inputs form for such a case. It should be noted that the Agile input is set to “Yes” since it is a binary yes/no input, and the majority of development efforts these days use some flavor of agile development.

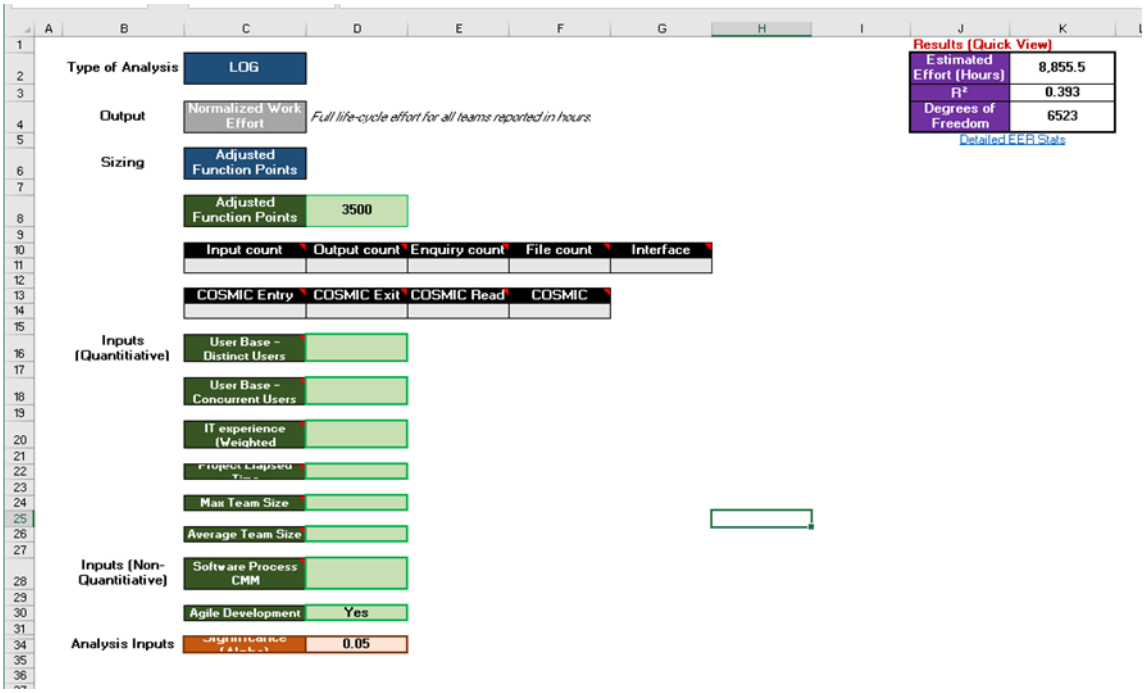
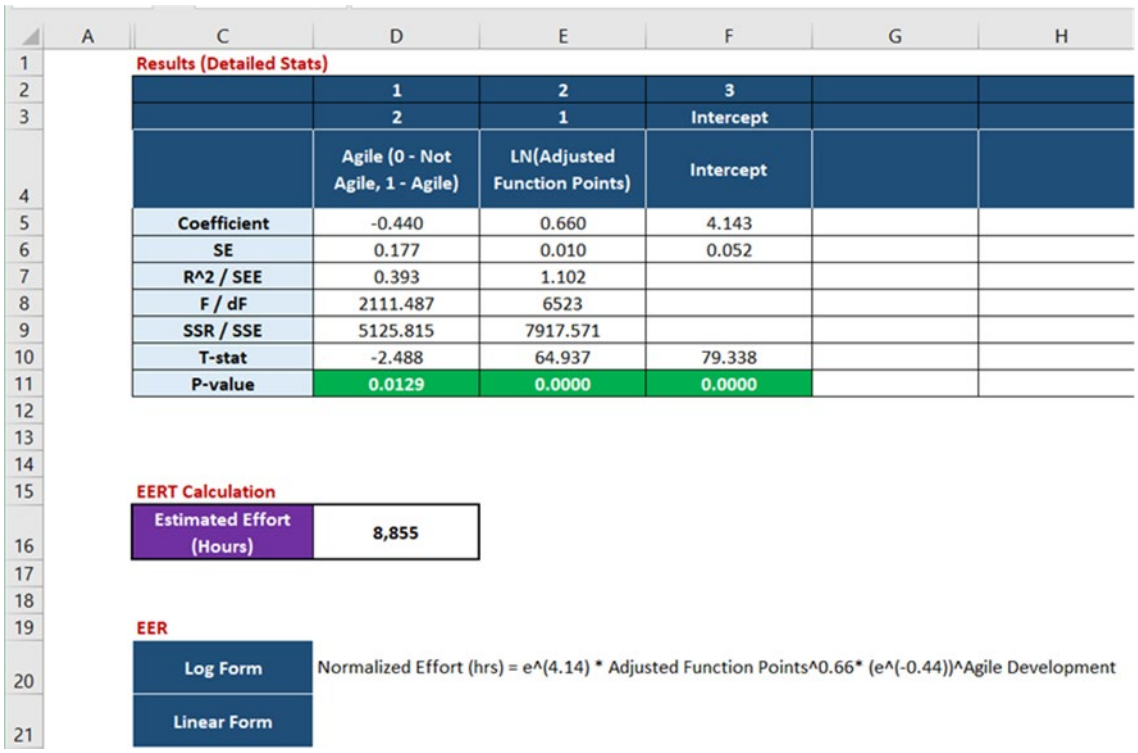


Figure 16: Example Inputs (Initial)

As shown in the top right corner of Figure 16, the inputs tab also provides a quick view of the results including an overall estimate, which in this case is **8,856 hours** to develop a project with a size of **3500 function points** using agile development methods. Figure 17, the detailed results tab, indicates that the underlying EER is statistically sound.



Results (Detailed Stats)

	1	2	3		
	2	1	Intercept		
	Agile (0 - Not Agile, 1 - Agile)	LN(Adjusted Function Points)	Intercept		
Coefficient	-0.440	0.660	4.143		
SE	0.177	0.010	0.052		
R^2 / SEE	0.393	1.102			
F / dF	2111.487	6523			
SSR / SSE	5125.815	7917.571			
T-stat	-2.488	64.937	79.338		
P-value	0.0129	0.0000	0.0000		

EERT Calculation

Estimated Effort (Hours)	8,855

EER

Log Form	Normalized Effort (hrs) = e^(4.14) * Adjusted Function Points^0.66* (e^(-0.44))^Agile Development
Linear Form	

Figure 17: Example Outputs (Initial)

This tab also provides the log form for the EER that SWEET generated for the user to produce an effort estimate. For this example, it translates to:

$$Normalized\ Effort\ (hrs) = e^{4.14} \cdot Adjusted\ Function\ Points^{0.66}(e^{-0.44})^{Agile\ Development}$$
$$= 62.8 \cdot Adjusted\ Function\ Points^{0.66}(0.644)^{Agile\ Development}$$

As the software development project being estimated is further defined, the analyst can enter additional inputs. Figure 18 shows a project estimating scenario that is based on the same 3500 FP input, but provides two inputs in addition to development methodology. Specifically, it is now known that the software being developed will have 150 distinct users and the development team has an average five years of experience.

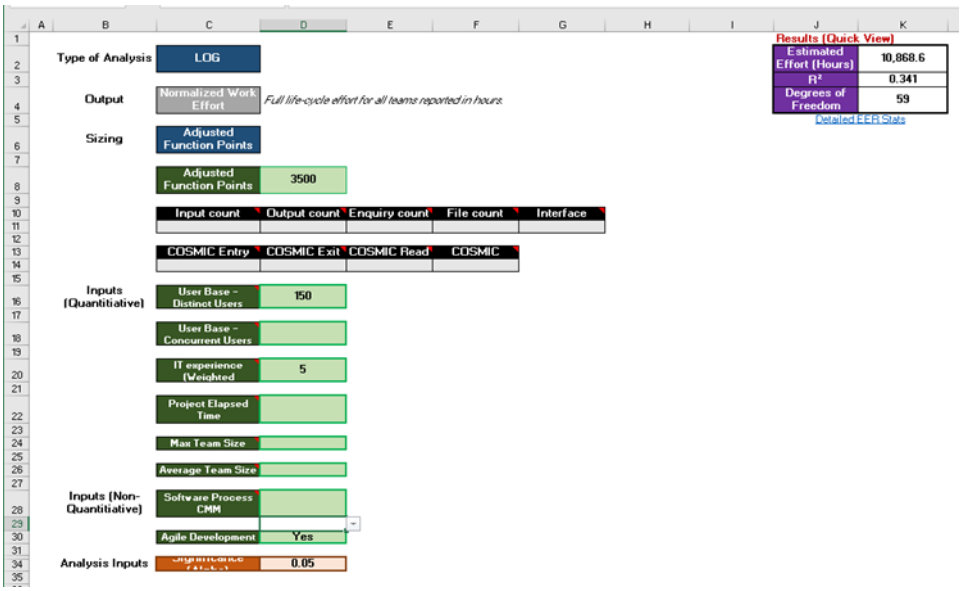
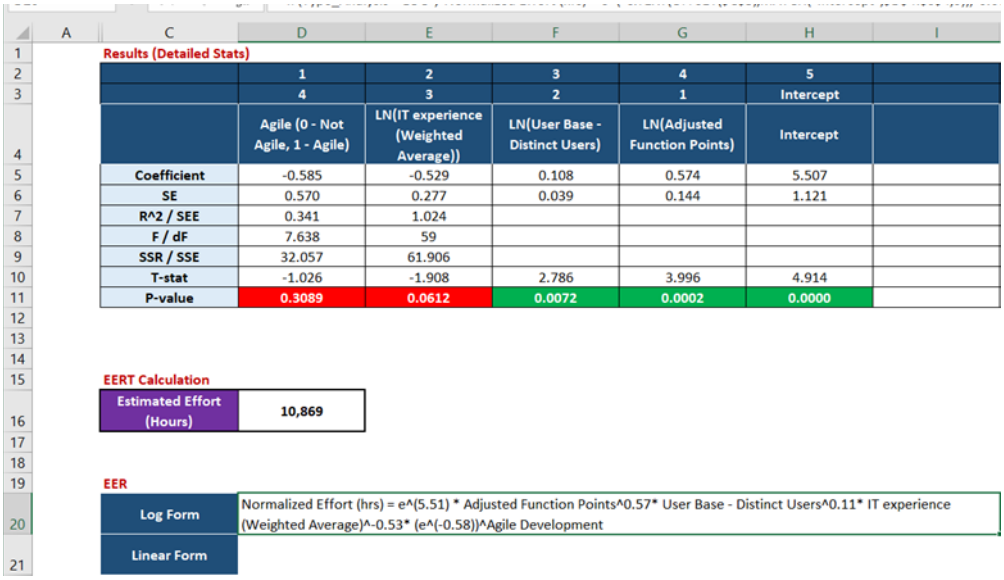


Figure 18: Example Inputs (Updated)

Based on these additional inputs, the estimated effort to develop the software in question has increased to **10,869 hours** and has fewer degrees of freedom than the initial estimate. This is because the number of data points in the SWEET data set that are applicable to the software in question is much smaller due the use of additional variables to be compared against.



Results (Detailed Stats)

	1	2	3	4	5
	4	3	2	1	Intercept
	Agile (0 - Not Agile, 1 - Agile)	LN(IT experience (Weighted Average))	LN(User Base - Distinct Users)	LN(Adjusted Function Points)	Intercept
Coefficient	-0.585	-0.529	0.108	0.574	5.507
SE	0.570	0.277	0.039	0.144	1.121
R^2 / SEE	0.341	1.024			
F / dF	7.638	59			
SSR / SSE	32.057	61.906			
T-stat	-1.026	-1.908	2.786	3.996	4.914
P-value	0.3089	0.0612	0.0072	0.0002	0.0000

EERT Calculation

Estimated Effort (Hours)	10,869

EER

Log Form	Normalized Effort (hrs) = e^(5.51) * Adjusted Function Points^0.57* User Base - Distinct Users^0.11* IT experience (Weighted Average)^-0.53* (e^(-0.58))^Agile Development
Linear Form	

Figure 19: Example Outputs (Updated)

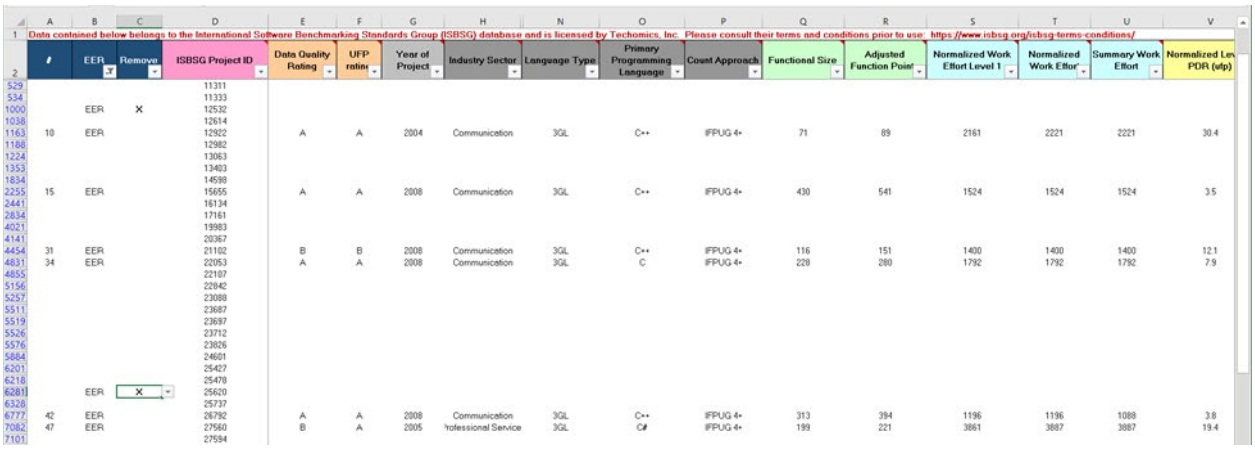
Figure 19, the detailed results tab, depicts the new EER in log form that the tool generated when two additional independent variables – Distinct User Base count and IT Experience – considered:

$$\begin{aligned} &Normalized\ Effort\ (hrs) \\ &\qquad = e^{5.51} \cdot Adjusted\ Function\ Points^{0.57} \cdot Distinct\ Users^{0.11} \\ &\qquad \cdot IT\ Experience^{-0.53} \cdot (e^{-0.58})^{Agile\ Development} \\ &= 247.2 \cdot Adjusted\ Function\ Points^{0.57} \cdot Distinct\ Users^{0.11} \\ &\qquad \cdot IT\ Experience^{-0.53} \cdot 0.56^{Agile\ Development} \end{aligned}$$

This demonstrates the unique power of SWEET to dynamically generate a new EER as an analyst enters additional data on the development project being estimated.

It is important to note that the statistics corresponding to the agile variable changed when the EER was revised to include two additional independent variables. Specifically, the agile independent variable is not as significant in the revised EER. Additionally, Figure 19 indicates that the IT experience variable is not necessarily statistically significant and warrants further scrutiny. Neither of these points should necessarily lead an analyst to determine the EER and resulting estimate aren't valid. However, they are cause to evaluate whether there are other candidate independent variables that offer greater explanatory value.

As discussed earlier, an analyst can refine the ISBSG-based data set that is being utilized to develop the EER. As Figure 20 depicts, the rows that indicate "EER" in the EER field (Column B) specify which data are used by default in the generation of the EER. The analyst has the option to remove projects by marking an 'x' in the *Remove* column which will further refine the data set being used.



Data contained below belongs to the International Software Benchmarking Standards Group (ISBSG) database and is licensed by Techomics, Inc. Please consult their terms and conditions prior to use: https://www.isbsg.org/isbsg-terms-conditions/

#	EER	Remove	ISBSG Project ID	Data Quality Rating	UFP rating	Year of Project	Industry Sector	Language Type	Primary Programming Language	Count Approach	Functional Size	Adjusted Function Point	Normalized Work Effort Level 1	Normalized Work Effort	Summary Work Effort	Normalized Level PDR (ufp)
529			11311													
534			11333													
1000	EER	X	12532													
1038			12614													
1163	10	EER	12922	A	A	2004	Communication	3GL	C++	IFPUG 4+	71	89	2161	2221	2221	30.4
1188			12982													
1224			13063													
1353			13403													
1834			14598													
2255	15	EER	15655	A	A	2008	Communication	3GL	C++	IFPUG 4+	430	541	1524	1524	1524	3.5
2441			16134													
2834			17161													
4021			19983													
4141			20367													
4454	31	EER	21102	B	B	2008	Communication	3GL	C++	IFPUG 4+	116	151	1400	1400	1400	12.1
4831	34	EER	22053	A	A	2008	Communication	3GL	C	IFPUG 4+	228	280	1792	1792	1792	7.9
4855			22107													
5156			22942													
5257			23088													
5511			23687													
5519			23697													
5526			23712													
5576			23826													
5884			24601													
6201			25427													
6218			25478													
6281		EER	X	25620													
6328			25737													
6777	42	EER	26792	A	A	2008	Communication	3GL	C++	IFPUG 4+	313	394	1196	1196	1088	3.8
7082	47	EER	27560	B	A	2005	Professional Service	3GL	C#	IFPUG 4+	199	221	3861	3887	3887	19.4
7101			27594													

Figure 20: Example Data Selection

This leads to a new output on the detailed results tab as shown below:

Results (Detailed Stats)

	1	2	3	4	5	
	4	3	2	1	Intercept	
	Agile (0 - Not Agile, 1 - Agile)	LN(IT experience (Weighted Average))	LN(User Base - Distinct Users)	LN(Adjusted Function Points)	Intercept	
Coefficient	-0.564	-0.546	0.108	0.553	5.664	
SE	0.577	0.292	0.039	0.148	1.157	
R^2 / SEE	0.324	1.037				
F / dF	6.838	57				
SSR / SSE	29.393	61.251				
T-stat	-0.976	-1.871	2.755	3.736	4.897	
P-value	0.3330	0.0665	0.0079	0.0004	0.0000	

EERT Calculation

Estimated Effort (Hours)	10,715

EER

Log Form	Normalized Effort (hrs) = e^(5.66) * Adjusted Function Points^0.55* User Base - Distinct Users^0.11* IT experience (Weighted Average)^-0.55* (e^(-0.56))^Agile Development
Linear Form	

Figure 21: Example Outputs (Final)

The effort estimate is now at **10,715 hours** instead of 10,869 hours; however, it did not change the LOG form for the EER in this case.

Overall, this shows that SWEET can dynamically adjust the EER being used as the analyst enters additional data, and will provide the analyst with information that can help determine if the effort estimate is valid for the analyst's needs or will require further refinement. In the estimating example discussed above, the analyst would probably want to perform further research and analysis. When doing so, the analyst should consider focusing on any input variables that are showing as not statistically significant. The analyst may then choose to further refine the analysis by removing non-significant variables or trying a different combination variables.

Data Modeling Limitations

The team identified some limitations in using the ISBSG data to build a dynamic regression model. ISBSG is considered a sparse dataset, where many data fields are minimally populated. When multiple fields are included as independent variables in a regression, the number of available data points generally drops, sometimes significantly. As a result, statistics tend to worsen as variables are added. This is somewhat counterintuitive. The more that is known about a program, for example towards the end of the development effort, more fields will be known and the statistics and fit may be worse than when few variables are known. In some instances, we might follow William of Occam and choose a simpler model over a more complex one. The power of SWEET, though, is that the analyst can easily generate multiple model forms as use them to cross-check each other.

SWEET is a dynamic regression model, where regressions are recalculated automatically based on known independent variables the user wants to include. Users may also hand-pick specific data points, resulting in a practically limitless number of possibilities for the regressions. The dynamic nature presents a challenge for users, since the results of the regression need to be verified to ensure the coefficients and exponents are such that the EER responds to the independent variables in a reasonable way.

Conclusion

Software estimation is historically difficult, and IT/software projects are prone to overruns. In order to address deficiencies in estimating techniques, our team developed the prototype SWEET Excel model for the Technomics Research Competition. Our approach addresses many issues plaguing black box models by leveraging ISBSG data and providing analysts with transparent insight into the effect of potential cost drivers. SWEET dynamically builds an EER, reflecting in real time the analyst's selection of data fields and source in a "clear-box" model. We balanced filtering out data fields of limited utility with providing the analyst the ability to generate an essentially limitless number of EERs.

Next Steps

Sizing Integration

As the primary sizing field in the ISBSG data set, a Function Point (FP) count is the only required input to SWEET and its complement of EERs. This critical fact accentuates the need for reliable software sizing methods throughout the life cycle.

Organizations like IFPUG, COSMIC, and NESMA have been at the forefront of developing FP counting standards and certification programs. In general, these methods are manually intensive and require sufficient program documentation often not readily available early in requirements and design. To surmount these challenges, we propose two potentially fruitful approaches:

1) **Expert-Based Analogy Scales**: For early phase assessments, programs typical rely on subject matter experts (SMEs). The accuracy and characterization of risk and uncertainty of these assessments can be improved via training and establishing an organization track record, as described in the cited paper (Braxton, 2022).

2) **Automated Counting Methods**: Natural Language Processing (NLP) and other automation techniques hold promise for parsing various forms of program documentation and producing preliminary FP counts. One of the dilemmas faced by such approaches is maturity vs. complexity. That is, if an automated method produces a low count, is that because the corresponding requirement is simple or because it is immature, and the documentation is not yet fully fleshed out? To be fair, a human FP counter would face the same conundrum, but they are arguably better equipped to discern which is closer to the truth. It should be noted that another recently completed Technomics Research Competition project did in fact create a prototype tool to accomplish simple FP point counting via NLP. Albeit preliminary in nature, the results of this project were encouraging enough to warrant continuing research. Stay tuned for more on the subject of NLP-based FP counting!

It is also important to note that the Joint Agile Software Innovation (JASI) Cost IPT is spearheading use of the Simple Function Point (SiFP) method as an alternative to traditional function point counting. SWEET would benefit from incorporating one or more of these three methods as a "front end."

Organizational Calibration

Since ISBSG is a benchmarking repository, it represents a wide variety of projects and development teams. Inevitably, if we are applying SWEET and its complement of EERs to estimate software projects within a given organization, we will find that a particular organization is more or less efficient on average than reflected by the comparable subset of the ISBSG data. The standard approach in this case is to calibrate the EERs to eliminate this "bias." This process is well established in the Cost Estimating Body of Knowledge (CEBoK), Module 3 Parametric Analysis, and in the legacy Parametric Estimating Handbook (PEH).

Depending on the maturity of the organization, additional data collection and the establishment of a track record may be required. Within the typical government

acquisition environment, relative efficiency or productivity may be driven by the buyer (government customer), the seller (industry developer), or both.

In any case, a built-in calibration functionality would be a worthwhile addition to SWEET.

Analytical Improvements

One of the strengths of SWEET is the way that it dynamically filters the data set based on the user's choice of data fields. It is an admirable goal to retain a greater proportion of the data set for analysis, and techniques such as *multiple imputation* are an established process for retaining data points with missing values for certain fields.

Our sense is that this represents a slippery slope for a database as extensive as ISBSG, but it may be possible to develop an approach where the majority of the desired fields are populated for the majority of the data points fitting a desired profile, and then imputation can be used to fill in the gaps on a limited basis. The predisposition of the cost analyst is to predict analysis on verifiable actuals wherever possible.

There is a largely untestable hypothesis in risk analysis that the programs with data are the better-behaved programs. Thus, as eye-popping as benchmarks for growth and uncertainty can be, they may be somewhat understated, since the programs for which we lack data may have (or, in the case of canceled programs, would have) performed even worse.

Tool Re-Hosting

SWEET uses Excel as a cost estimating *lingua franca* and eschews embedded Visual Basic for Applications (VBA) to ensure that the "live" calculations reflect the dynamically selected data set with maximum fidelity. The current version essentially functions as a prototype, and it could be re-hosted to make improvements to user interface (UI), speed of calculations, and/or visualization of results.

Business Intelligence (BI) tools such as Tableau and Power BI are well-suited for data visualization, but not as much for statistical calculations. Conversely, tools like R are well-suited for statistical calculations. By leveraging R Shiny in conjunction with R, we

might be able to achieve the best of both worlds. In the end, the purpose of the current research was not development of a tool, but rather the exploration of the ISBSG data set and the proof of concept for the Clear Box modeling approach. We are pleased to have accomplished these objectives … and more!

Appendix

Acronym List

Acronym	Expansion
BI	Business Intelligence
CER	Cost Estimating Relationship
COCOMO	Constructive Cost Model
EER	Effort Estimating Relationship
FP	Function Points
ISBSG	International Software Benchmarking Standards Group
JASI	Joint Agile Software Innovation
LOC	Lines of Code
LOLS	Log Ordinary Least Squares
NLP	Natural Language Processing
OLS	Ordinary Least Squares
SiFP	Simple Function Points
SLOC	Source Lines of Code
SME	Subject Matter Expert
SWEET	Software Effort Estimating Tool
UI	User Interface
VAF	Value Adjustment Factor
VBA	Visual Basic for Applications

Bibliography

Many papers have used ISBSG as a data source. The below list includes those that were available via the ICEAA papers repository or Research Gate.

(n.d.). Retrieved from International Software Benchmarking Standards Group (ISBSG): https://www.isbsg.org/

(n.d.). Retrieved from McKinsey: https://www.mckinsey.com/business-functions/mckinsey-digital/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value

Braxton, P. J. (2022). Uncertainty of Expert Judgment in Agile Software Sizing. *ICEAA Conference Proceedings.* Pittsburgh, PA: ICEAA.

Cost Estimating Body of Knowledge (CEBoK). (2013). Annandale, VA: ICEAA.

Deng, K., & MacDonell, S. (2008). Maximising data retention from the ISBSG repository. *Proceedings of the Twelfth International Conference on Evaluation and Assessment in Software Engineering (EASE2008).* Bari, Italy: British Computer Society.

Dery, D., & Abran, A. (n.d.). *Investigation of the Effort Data Consistency in the ISBSG Repository.*

Elyassami, S., & Idri, A. (2012). Investigating Effort Prediction of Software Projects on the ISBSG Dataset. *International Journal of Artificial Intelligence & Applications (IJAIA)*, 121-132.

Kosmakos, C., & Brown, D. H. (2022). Are We Agile Enough to Estimate Agile Software Development Costs? *ICEAA Conference Proceedings.* Pittsburgh, PA: ICEAA.

Minkiewicz, A. F. (2013). Lessons Learned from the ISBSG Data Base. *ICEAA Conference Proceedings.* New Orleans, LA: ICEAA.

Minkiewicz, A. F. (2021). Lessons Learned From Software Maintenance and Support Datasets. *ICEAA Online Workshop Proceedings.* Annandale, VA: ICEAA.

SLOC per FP. (n.d.). Retrieved from Quantitative Software Management (QSM): https://www.qsm.com/resources/function-point-languages-table

Smart, C. B. (2021). *Solving for Risk Management: Understanding the Critical Role of Uncertainty in Project Management.* New York, NY: McGraw Hill.

van Heeringen, H. (2013). Estimating Real-time software projects with the COSMIC FSMM and the ISBSG data repository. *ICEAA Conference Proceedings.* New Orleans, LA: ICEAA.

Annotated Sources

(International Software Benchmarking Standards Group (ISBSG), n.d.) – Informational web page for primary data source

(Minkiewicz, Lessons Learned from the ISBSG Data Base, 2013) – Describes development of estimating templates for filtering of ISBSG data and data normalization and calibration

(van Heeringen, 2013) – Specific focus on real-time software and COSMIC sizing within the ISBSG database

(Minkiewicz, Lessons Learned From Software Maintenance and Support Datasets, 2021) – Leveraging ISBSG Maintenance and Support (M&S) database to development Software Sustainment recommendations

(Deng & MacDonell, 2008) – Focus on maximizing the proportion of data points and data fields that can be retained from the ISBSG database

(Dery & Abran) – Develops a normalized work effort field derived from the project work effort field in ISBSG across data points with different number of project phases

(Elyassami & Idri, 2012) – Investigation of fuzzy ID3 decision tree models for software effort estimation