# ESTIMATING EFFORT FOR LOW-CODE APPLICATIONS

by

Rhonda R. Butler

May, 2020

Director of Thesis: Dr. Nasseh Tabrizi

Major Department: Computer Science

Two issues continually plaguing the software industry are software size calculation and project effort estimation. Incorrect estimates may lead to inappropriate allocation of resources (people), shortage of time, insufficient funds, and possibly project failure. The purpose of this study is to explore current methods of software effort estimation and the applicability to low-code application development. The study seeks to answer the research question: Can the current methods be used to estimate effort for applications built with low-code platforms? The goal is to analyze some of the popular estimation methods to see if they can be applied to low-code application development.

# ESTIMATING EFFORT FOR LOW-CODE APPLICATIONS

A Thesis

Presented To the Faculty of the Department of Computer Science

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master of Science in Software Engineering

by

Rhonda R. Butler

May, 2020

ESTIMATING EFFORT FOR LOW-CODE APPLICATIONS

by

Rhonda R. Butler

APPROVED BY:

DIRECTOR OF
THESIS: ______________________________

M.N.H. Tabrizi, PhD

COMMITTEE MEMBER: ______________________________

Qin Ding, PhD

COMMITTEE MEMBER: ______________________________

Venkat Gudivada, PhD

CHAIR OF THE DEPARTMENT
OF COMPUTER SCIENCE: ______________________________

Venkat Gudivada, PhD

DEAN OF THE
GRADUATE SCHOOL: ______________________________

Paul J. Gemperline, PhD

## DEDICATION

This thesis is dedicated to my husband, Aaron, who has been there for the entire journey and has always given his love and support.

## ACKNOWLEDGEMENTS

I would like to thank my husband for his support and sacrifices that were made to allow me the time to complete my educational goals.  Also, I would like to thank my children who motivated me to complete my degree and always held me in the highest regard.  I would like to thank my siblings who were always proud of my accomplishments.  Lastly, I offer thanks to my parents, Donald and Margaret Redfern, whose numerous sacrifices did not go unnoticed.

I acknowledge my co-workers at the North Carolina Western District of the United States District Court who would not let me quit the journey.  To Dr. Tabrizi, I thank you for your guidance through this process.  To all the educators who made me feel exceptional, you planted seeds of confidence that have aided me through my life.  Finally, to all of you who sponsored math and science camps that encouraged women and minorities to pursue opportunities in STEM, know that your work was not in vain.

TABLE OF CONTENTS

LIST OF FIGURES

# CHAPTER 1 INTRODUCTION

“How long will it take for you to complete this project?” is a question dreaded by many software developers. *The Cone of Uncertainty* as referenced in Figure 1, described by Steve McConnel illustrates the unlikelihood of answering this question accurately at a project’s inception. This is due to the variability of multiple factors.

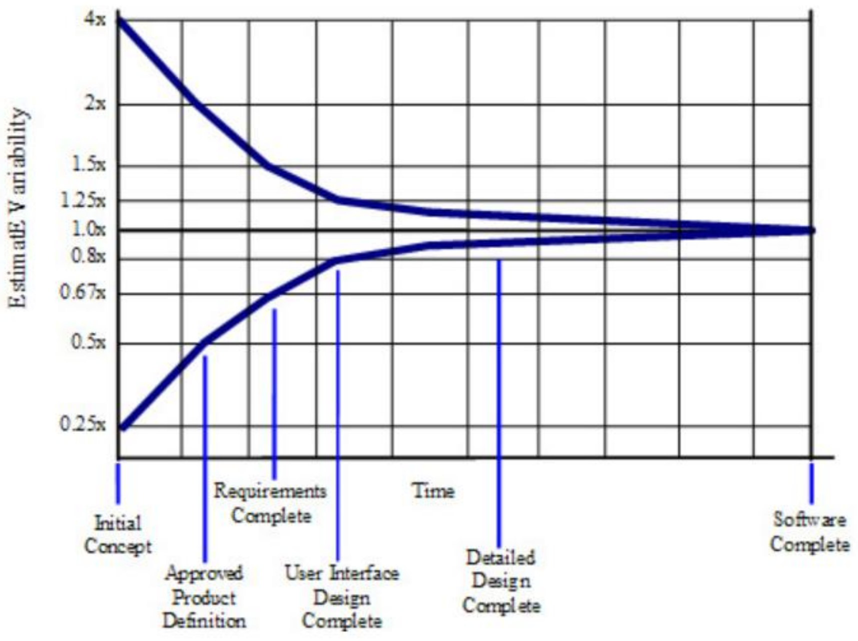


*Figure 1 Cone of Uncertainty*

Since the question is unavoidable, how do you accurately respond without inflating the estimate to an unacceptable time? Conversely, how do you respond without under estimating the project time and possibly missing the deadline? Even for the experienced, software project effort estimation remains an art that is difficult to master. Out of all cancelled software projects, approximately 40% were due to failures in effort estimation (Rubens 2014). Excluding the improvements reported in the Chaos Reports by the Standish Group, software estimation accuracy has not changed much since the 1990s (Jorgensen and Moløkken-Østvold 2006).

From the software developer's point of view, estimation methods have not changed much either. In Boehm's classification system, these methods were summarized into three categories: expert judgement, algorithmic estimation, and analogy based estimation (Li, Xie and Goh 2007). In spite of extensive research on formal estimation models, the dominating estimation methods are expert judgements derivatives (Jorgensen 2004).

Conversely, there have been extensive technological advances since the 1990s in the way software applications are developed (Skwirzynski, Springer and Verlag 1986). Low-code development is one innovation that has risen in popularity due to its faster delivery time (Skwirzynski 1986). This project researched existing effort estimation concepts and determined their suitability for application to a low-code developed project. The major contribution from this study is to highlight the deficiency of some existing methodologies and demonstrate the potential of other methodologies for estimating effort for low-code development projects.

## 1.1 Background of the Study

Clay Richardson, an analyst at Forrester Research, defines a low-code platform as one that enables fast application development and delivery "with a minimum of hand-coding and minimal upfront investment in setup, training, and deployment" (Rubens 2014). As referenced in Figure 2**,** there are four major practice changes accompanying low-code platforms. Usually, low-code development solutions focus on testing and learning instead of standardized development methods such as waterfall. They place funding emphasis on research instead of development and utilize minimal architecture practices. Furthermore, success is measured by customer outcomes. By focusing on these practices, low-code platforms expedite application delivery by providing visual tools for the quick definition and assembly of forms.

In addition, they allow the rapid creation of multi-stage workflows (Rubens 2014). Michelle Tackabery noted in her blog *"Low-Code" Development Explained* (2014), the standardized tools used in low-code platforms allow developers to streamline the creation of forms and user experiences but also accommodate a single point of control for maintenance and updates.

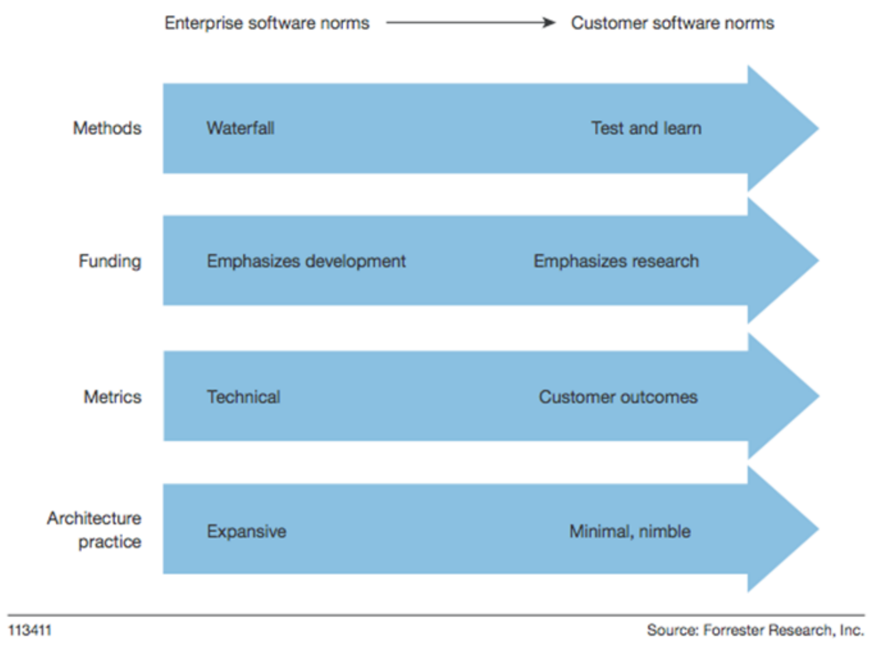


*Figure 2 Four Major Practice Changes Accompanying Low-Code Platforms*

Although low-code solutions are traditionally seen as a method for allowing non-technical business users with little or no programming skills to create process applications (Forbes 2016), an increasing number of organizations are adopting a bimodal application development approach. Full-time developers, who might otherwise code in Java, .NET or C#, are utilizing the tools offered in low-code development to create applications (Tackabery 2014)(Rubens 2014). As reported in *The Forrester Wave™: Low-Code Development Platforms, Q2 2016*, low-code platforms are still in their infancy stage but are growing rapidly.

The three dominant drivers for shaping the future of low-code platforms are:  the drive to expand and diversify the development talent pool by utilizing developers with nontraditional backgrounds, a desire to expand the low-code segment by offering support to the general purpose segment and an increase in funding that validates the marker for low-code platforms (Forrester Research, 2016).

It is important to consider what a low-code platform is when talking about software estimation.  For the scope of this research, low-development is defined based on the criteria used by Forrester Research in *The  Forrester Wave™: Low-Code Development Platforms For AD&D Professionals Q1 2019.*  The platform*:*  offers declarative tools to define data, logic, flows, forms and other application artifacts without writing code; adoption for a very low cost without requiring formal paid training courses to build applications; supports a range of use cases including web, mobile, transactional database, event-processing, business reporting, process automation and analytical applications; primarily targets large enterprises.

When given the task for estimation, three major factors are considered:  size, effort and schedule. Size is normally calculated first because most effort or schedule equations require a size parameter.  The size parameter is normally in either LOC (Lines of Code) or function points (Umair 2013).  Since one characteristic of a low-code platform is the visual nature of the development process (either point-and-click or drag-and-drop tools (Rubens 2014)), the code behind the application is often inaccessible.  This limits the use of some traditional estimation methods which rely on lines of code.

## 1.2 Statement of the Problem

Accurate software project size is one of the most desired goals of a successful software project manager. Accurate size estimates assist the project manager with realistic estimates on the amount of time and resources needed to complete a project. In addition, it allows the project manager to convey more accurate financial predictions to allow the stakeholder to make wise investments. Although there are many methodologies available to help predict software size, lines of code remain the basis for the estimations. As low-code development platforms increase in usage, the dearth of project estimation methodologies will have a direct impact on project success. The focus of this research is the evaluation of existing methodologies to determine suitability for low-code project size estimation.

## 1.3 Purpose of the Study

1. To investigate the historical use of some existing software project size estimation methodologies
2. To theoretically apply the methodologies to low-development platform project
3. To suggest a methodology to use for low-development projects (if one is found)

## 1.4 Definition of Terms

The following terms are defined for clarification purposes:

1. Function Point - The function point analysis determines the size of software development project based on the size of the problem. The basis of function point analysis is to count function points that can be classified into five broad categories: External Input, External

2. Logical Files, External Interface Files, and External Inquiry.
3. Lines of Code - The line of code is the oldest and one of the widely used techniques to derive project estimates. The size is calculated by estimating the total lines of code (program length). There is no widely accepted standard that specifies whether to include the program comments when counting the lines of code.
4. Project Effort Estimation - An estimated in person days or person months that indicates the total time that will be taken to complete the project.

## 1.5 Assumptions

The low-development project estimation topic was selected because of the lack of information available when completing projects in this area. Numerous searches were completed in academic literature related to low-development projects and were unsuccessful in locating plentiful information on this topic. Additional searches were performed in the current body of knowledge for project estimation techniques for low-development platforms and were unsuccessful in locating anything related to the topic. The possibility exists that proprietary models may be available for use within corporations; however, the information has not been published to the public. Further internet searches were performed to include broader terms and still no information was found that specifically addressed the thesis topic.

The following assumptions were made during this research:

1. There is not a methodology currently in existence specifically for low-development platforms.
2. It was assumed that current methodologies can be reasonably applied to a low-development platform project.

## 1.6 Limitations and Delimitations

This research has the following limitations:

1. It is theoretical in nature and the methodologies were not applied to an actual project created by a low-development platform.
2. There are various types of low-development platforms. Some are more robust than others. Low-development as used within this research is a generic term as defined in 'The Background of the Study'. It cannot be generalized to all low-code development platforms.

This research has the following delimitations:

1. The result of the research is the suitability of current methodologies to effort estimation of low-development platforms. Application was limited to a hypothetical project. Specific real-world testing was outside the scope of this research.
2. The research is based on the general use of the methodologies. It was not applied to all the various extensions and modifications available for the methods.

## 1.7 Thesis Contribution

The quest for a software project size estimation model for low-development platforms in the current body of knowledge did not yield any results. This research will:

- Stimulate research in the low-development platform project size estimation
- Demonstrate the disparity of current estimation models applicability to low-code development
- Suggest a possible future method for low-code effort estimation

# CHAPTER 2 RELATED WORK

Source line of code, function points, use-case points and object points are measurement standards used to estimate software size.  As notated in Figure 3 below, the measurement standards are over twenty five years old; however, they continue to be the basis of effort estimation methodologies.

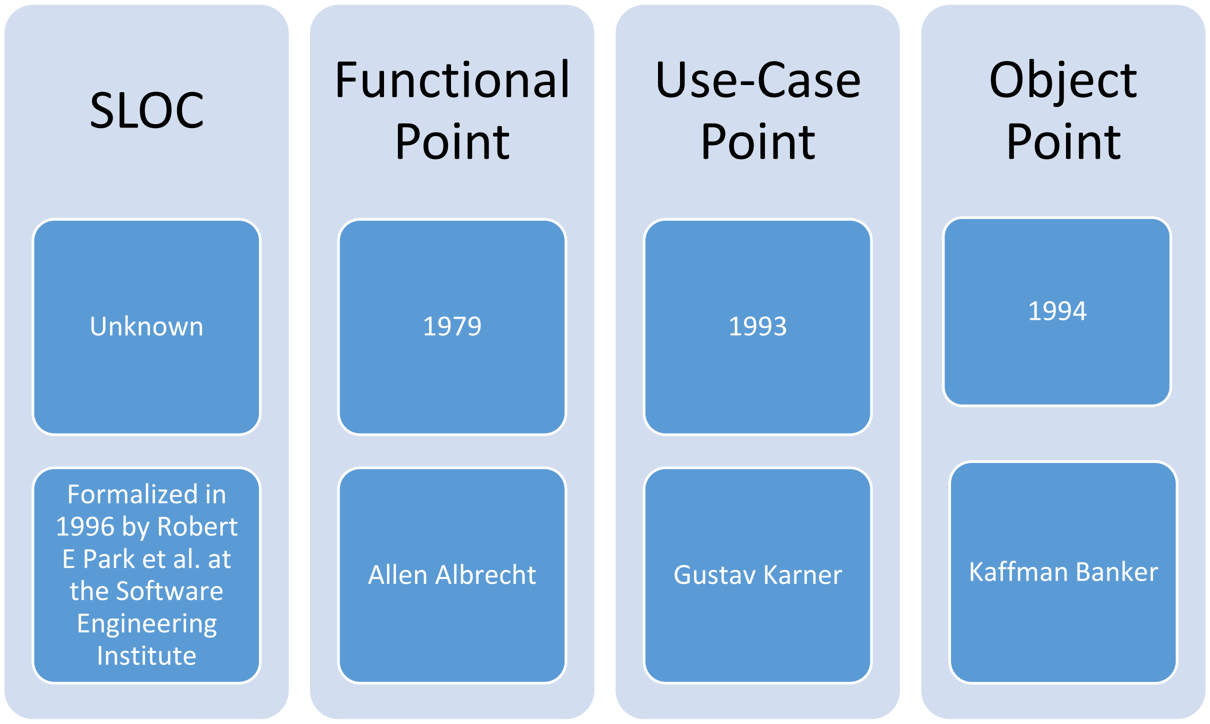


*Figure 3 Historical Metrics Creation Dates and Creator*

This first part of this chapter discusses the historical metrics which are key to understanding the current effort estimation process that are discussed in the latter part of the chapter.

## 2.1 Size Oriented Metrics

Source Lines of Code (SLOC) is the software metric used to measure the size of software program by counting the number of lines in the text of the program’s source code. This metric does not count blank lines, comment lines, and libraries. SLOC measurements are programming language dependent. They cannot easily accommodate nonprocedural languages (Jodpimai, Sophatsathit and Lursin 2009).

Boehm’s Deliverable Source Instruction (DSI) is another size-oriented metric. It is closely related to source lines of code. The main difference between DSI and SLOC is the "if-then-else" statement. The statement is counted as one line in source lines of code but might be counted as several lines in deliverable source instructions (Jodpimai 2009).

## 2.2 Function Oriented Metrics

Function Point (FP) defined by Allan Albrecht at IBM in 1979, is a unit of measurement to express the amount software functionality. Function point analysis (FPA) is the method of measuring the size of software. The advantage is that it can avoid source code error when selecting different programming languages. Function point is programming language independent which makes it ideal for applications using conventional and nonprocedural languages. It is based on data that is more likely to be known early in the project (Ingold 2013),(Jodpimai 2009).

## 2.3 Object Point Oriented Metrics

In 1994 Kaffman Banker introduced the concept of object oriented points which measure the software size from a different dimension. This measurement is based on the number and complexity of the following objects: screens, reports and 3GL components. It is easy to use at the early phase of the development cycle. Also object points are a measurement of size. Object points have been used in COCOMO II for cost estimation (Bogdan 2003).

## 2.4 Use-Case Oriented Metrics

Use case points were first described by Gustav Karner in 1993. The number of use case points in a project is a function of the following: the number and complexity of the use cases in the system, the number and complexity of the actors on the system, various non-functional requirements (such as portability, performance, maintainability) that are not written as use cases, and the environment in which the project will be developed (Ochodek 2011).

## 2.5 Effort Estimation Methods

According to Pichai, Peraphon, and Chidchanok (2009), software researchers have provided effort estimation for decades with most of them focusing on formal software estimation models, such as SLIM, COCOMO 81, COCOMO II, Kemerer and Albrecht-Gaffney. The metrics listed above are utilized in the following methods for effort estimation.

### 2.5.1 COnstructive COst MOdel (COCOMO)

Constructive Cost Model was first published in 1981 by Dr. Barry Boehm(COCOMO 81). Boehm proposed three models of the COCOMO model: basic, intermediate and advanced where software development effort is expressed as a function of program size in estimated thousand delivered source instructions (KDSI). Drawbacks of COCOMO are the difficulty to estimate KDSI early in a project when most effort estimates are required, misclassification of the development mode and the need to tune the model to the organization using historical data which may not be available (McConnell 2006)(Umair 2013).

### 2.5.2 COnstructive COst MOdel II(COCOMO II)

Constructive Cost Model II was published as a joint effort between the University of California Center for Software Engineering and the COCOMO II Project Affiliate Organizations in 1997. The contributors reengineered COCOMO focusing on issues such as non-sequential and rapid-development process models, reuse driven approaches, software process maturity affects and process driven quality estimation. COCOMO II differs from COCOMO in its use of thousand source line of code (KSLOC) which is logical code, usage of 161 data points instead of 63, and the use of the five scale factor as opposed to the three development modes. COCOMO II also adjusts for software reuse and reengineering where automated tools are used for translation of existing software (McConnell 2006).

### 2.5.3 COnstructive Rapid Application Development MOdel(CORADMO)

This approach is an extension of COCOMO II and was first introduced in 2000 by Ingol *et al* in response to accelerating development schedules. Unfortunately, at that time there was not enough data to sufficiently calibrate the model (Ingold, Boehm, and Koolmanojwong 2013); however, after additional research in expediting systems via lean and agile methods, a revised CORADMO was introduced in 2013 (Ingold). The study identified a set of key factors (Ford and Morris 2012) that, in combination with factors derived in the earlier CORADMO research (Ingold 2013), could be used to model rapid application development projects' schedule acceleration. These factors fall in the categories of product, process, project, people, and risk.

### 2.5.4 The Web Objects Method

The Web Objects method was proposed by Reifer to measure the size of Web applications (2000). This sizing metric is an adaptation of the COCOMO II model and is used to more accurately estimate web-based software development effort and duration. Reifer added four new web-related components: multimedia files, web building blocks, scripts, and links, to the five predictors of the FPA method. (Reifer 2000).

The metric computes size using Halstead's equation for volume (Halstead 1977). A table of predictors is used to identify uncounted web objects. Similar to FPA, the further step is to determine the complexity of the identified instances of the nine components. Reifer claims that computing size in this way offers advantages such as a mathematically sound foundation, easily extendable to include new predictors and the approach addresses the unique characteristics of web-based development. He acknowledges the disadvantages that software science as a basis is controversial, planning and data collection costs rise as you add additional predictors, and web objects counts are sensitive to counting conventions (Reifer 2000).

### 2.5.5 Work Break-Down Structure

According to Jorgensen, work break-down is the most commonly used effort estimation method (Jorgensen 2004). To use this method, you decompose the project into small parts of work tasks. Then, you estimate the effort for every task. Work break-down is an expert judgement method and it comes with two flavors: Three Point System and Delphic Oracle. Using the Three Point method an expert gives 3 estimations for every task:  Best Case, Most Probable, Worst Case. The effort for every task is the outcome of a weighted average of the three estimations where the most probable effort gets a higher weight.

In Delphic Oracle, three ‘experts’ estimate the tasks independently.  The final task effort is the average of the three(McConnell 2006)(Spyrosktenas 2013).

### 2.5.6  Analogy / Comparison

It is a Formal Estimation Method where projects with similar characteristics are located.  The project most similar to the one you are trying to estimate is chosen as the base for your new estimate.

Comparison based estimation involves considering the attributes of the project to be estimated, selecting projects with similar attributes and then using the median values for effort, duration, and other factors from the selected group of projects to produce an estimate of project effort(McConnell 2006),(Spyrosktenas 2013).

### 2.5.7  Bayesian Approach

This method consists of merging expert-opinion (Delphi) and project data, based on the variance of the two sources of information.  Effectively, a weighted average is produced which gives higher weights to parameter values with smaller variances (McConnell 2006).

# CHAPTER 3 ESTIMATING EFFORT FOR A LOW-CODE APPLICATION

In the following section, an attempt is made to apply the selected effort estimation methods discussed in Section 2.5. Due to the lack of available ‘real world’ project data, a hypothetical project was created for the purpose of applying the effort estimation methods.

## 3.1 Hypothetical Project Software Requirements

The purpose of the employee travel authorization management system is to allow employees to easily submit travel plans and expenses for pre-authorization. The requests are sent to the manager and if approved are relayed to the finance department for allocation of funds. The system will have a database backend to support storing the processed requests for financial auditing purposes. The user experience should allow for employees, managers and finance to easily see the status of requests. A low-development platform should be used to create the employee travel authorization management system.

### 3.1.1 Product Details

The travel request includes the employee, manager, reason for travel, travel dates, travel destination, mode of travel, estimated costs for the selected mode of travel, lodging cost, taxes and miscellaneous expenses.

### 3.1.2 User Class and Characteristics

The employee should be able to do the following functions:

- Submit travel authorization requests

- Attach supporting documentation
- View the status of pending requests
- View their completed requests

The manager should be able to do the following functions:

- View all their employee pending requests
- View their employee completed requests
- Approve or reject employee requests

The finance department should be able to do the following functions:

- View requests pending financial processing
- View completed requests
- Update approved requests with financial information

### 3.1.3 Operating Environment

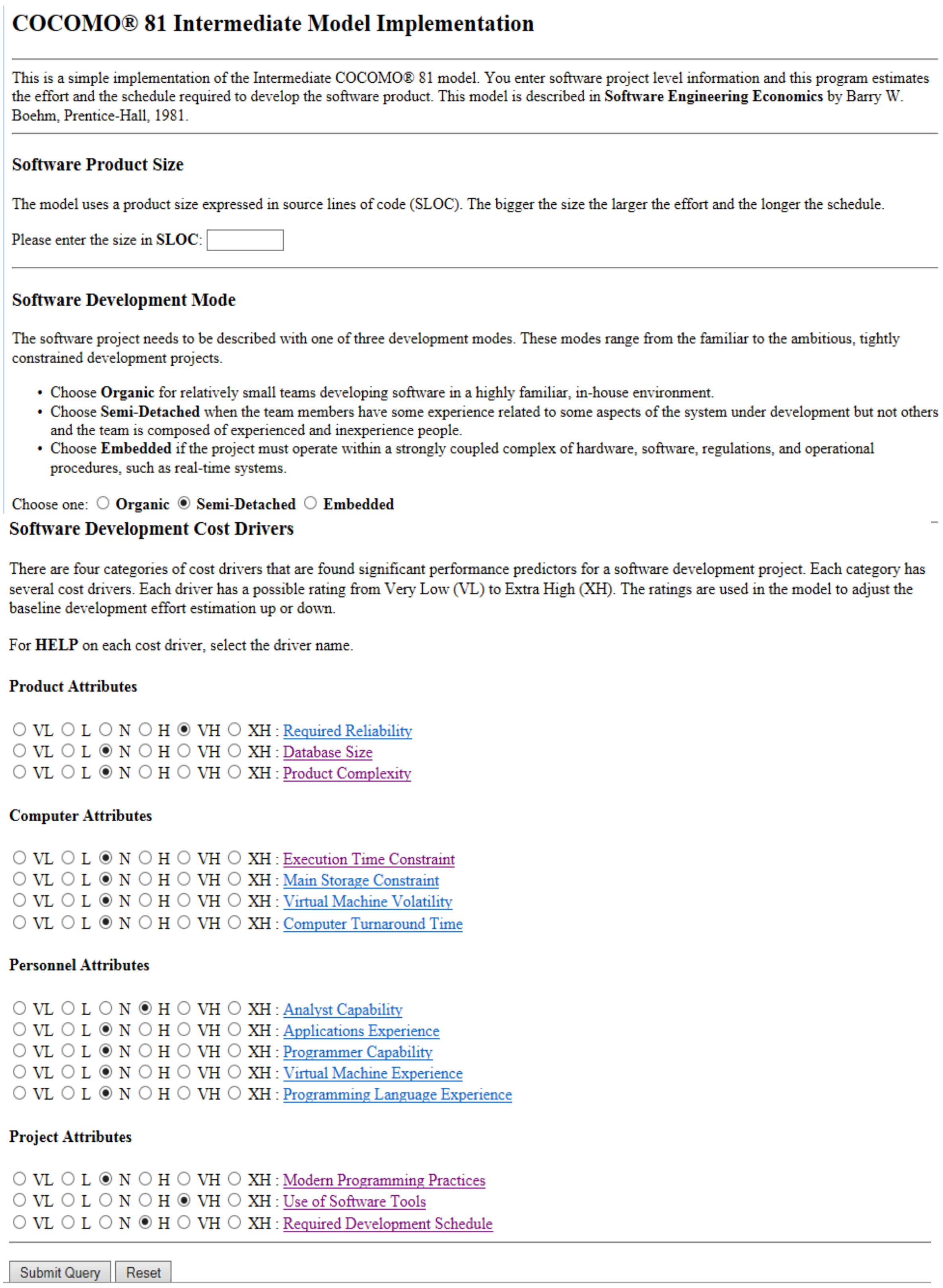
The operating environment for the employee travel authorization management system is:

- Accessible from Intranet only
- Low-code development platform
- Client/server system
- SQL Database

## 3.2 COnstructive COst MOdel (COCOMO) Implementation

To implement the COCOMO estimation model to a low-code development project, an estimate of effort must be obtained. In small projects, effort is a linear function of the project size and is calculated: Effort = a * Size + b. In larger projects, the size is scaled exponentially and is calculated as Effort = a * SIZE $^{b}$. The parameters, a and b, are determined by the version of COCOMO (basic or intermediate) and the type of project being developed (organic, semi-detached, or embedded) (Ingold 2013).

When using the online COCOMO calculator located at sunset.usc.edu/research to estimate effort for the fictional project, the missing required component is the 'lines of code' as seen in Figure 4. The source code for low-code development is inaccessible; therefore, the accurate amount for lines of codes is inaccessible. If the source code was available, the contained information could be misleading. It may take a few minutes to drag and drop a template onto a canvas which may equate to hundreds of lines of code. To effectively use the traditional COCOMO model, a gearing factory is needed to obtain the lines of code.



## COCOMO® 81 Intermediate Model Implementation

This is a simple implementation of the Intermediate COCOMO® 81 model. You enter software project level information and this program estimates the effort and the schedule required to develop the software product. This model is described in **Software Engineering Economics** by Barry W. Boehm, Prentice-Hall, 1981.

### Software Product Size

The model uses a product size expressed in source lines of code (SLOC). The bigger the size the larger the effort and the longer the schedule.

Please enter the size in **SLOC**:

### Software Development Mode

The software project needs to be described with one of three development modes. These modes range from the familiar to the ambitious, tightly constrained development projects.

- Choose **Organic** for relatively small teams developing software in a highly familiar, in-house environment.
- Choose **Semi-Detached** when the team members have some experience related to some aspects of the system under development but not others and the team is composed of experienced and inexperience people.
- Choose **Embedded** if the project must operate within a strongly coupled complex of hardware, software, regulations, and operational procedures, such as real-time systems.

Choose one: **Organic** [✓ **Semi-Detached**] **Embedded**

### Software Development Cost Drivers

There are four categories of cost drivers that are found significant performance predictors for a software development project. Each category has several cost drivers. Each driver has a possible rating from Very Low (VL) to Extra High (XH). The ratings are used in the model to adjust the baseline development effort estimation up or down.

For **HELP** on each cost driver, select the driver name.

**Product Attributes**

VL L N H [✓ VH] XH : Required Reliability
VL L [✓ N] H VH XH : Database Size
VL L [✓ N] H VH XH : Product Complexity

**Computer Attributes**

VL L [✓ N] H VH XH : Execution Time Constraint
VL L [✓ N] H VH XH : Main Storage Constraint
VL L [✓ N] H VH XH : Virtual Machine Volatility
VL L [✓ N] H VH XH : Computer Turnaround Time

**Personnel Attributes**

VL L N [✓ H] VH XH : Analyst Capability
VL L [✓ N] H VH XH : Applications Experience
VL L [✓ N] H VH XH : Programmer Capability
VL L [✓ N] H VH XH : Virtual Machine Experience
VL L [✓ N] H VH XH : Programming Language Experience

**Project Attributes**

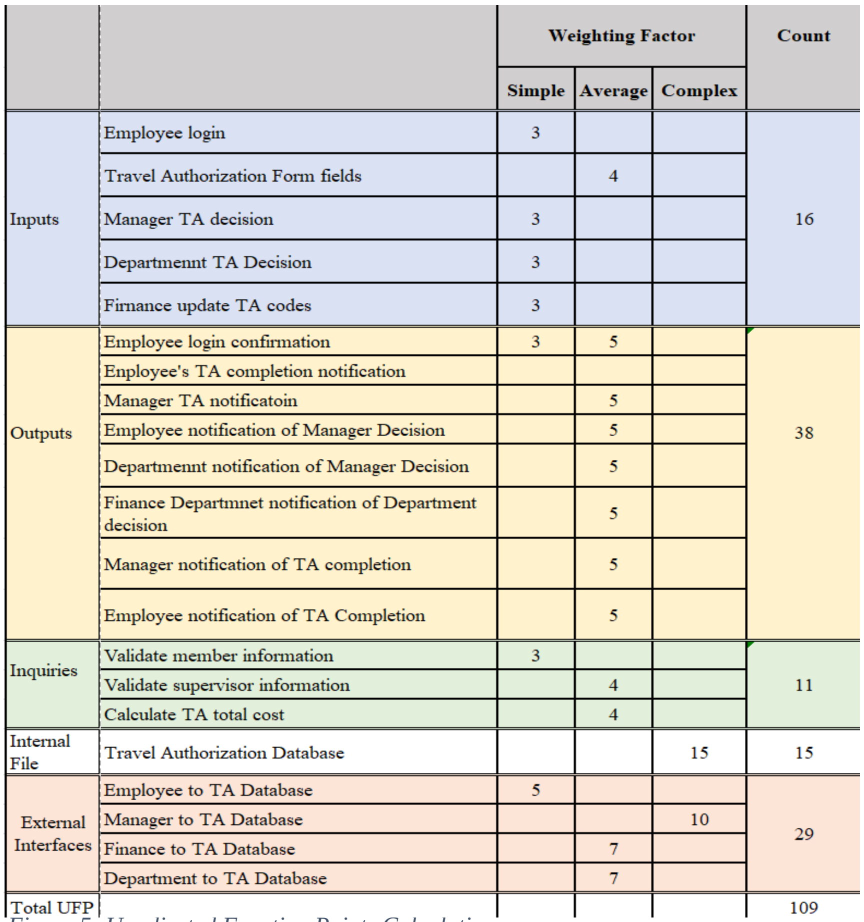
VL L [✓ N] H VH XH : Modern Programming Practices
VL L N H [✓ VH] XH : Use of Software Tools
VL L N [✓ H] VH XH : Required Development Schedule

*Figure 4 COCOMO Implementation*

## 3.3 COnstructive COst MOdel II(COCOMO II) Implementation

In COCOMO II effort (in person months) is based on the software project's size measured in thousands of lines of code where Effort = a*EAF*KSLOC$^{b}$ ( 'a' is 2.94 and the scaling factor 'b' is about 1.0997). The coefficient 'a', also known as the effort adjustment factor, is derived from seventeen cost drivers and the scaling factor comes from five scale drivers (McConnell 2006). Like the original COCOMO, COCOMO II uses lines of code as its basis for effort calculation which poses a conundrum in low-code developed applications because this parameter is not available. However, many automated COCOMO II tools give an option of using the function point sizing method. Using function point sizing, it is possible to continue with the COCOMO II estimate for low code.

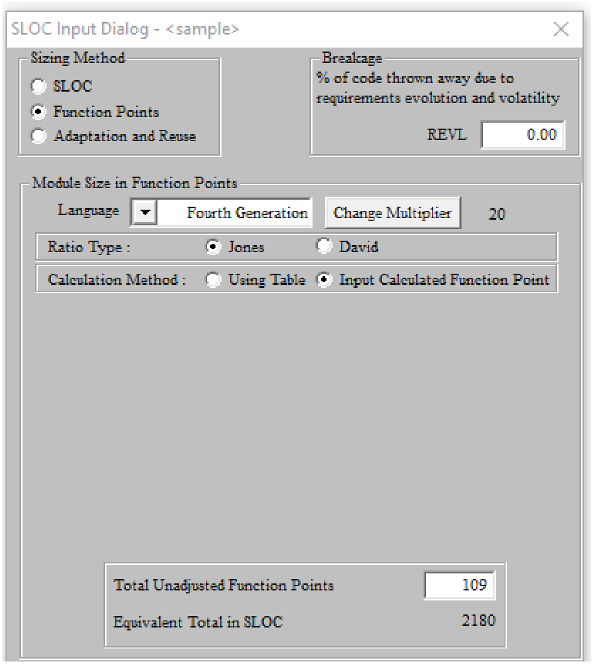
The next step is the calculation of unadjusted functions points. This requires the count and complexity of external inputs, external outputs, external queries, internal log files, and external log files. Based on the hypothetical project description, it was feasible to ascertain this information low-code development project. The detailed calculation for unadjusted function point is found in Figure 5 below. The unadjusted function points total for the fictional project is 109.



| | | Weighting Factor | | | Count |
|---|---|---|---|---|---|
| | | Simple | Average | Complex | |
| Inputs | Employee login | 3 | | | 16 |
| | Travel Authorization Form fields | | 4 | | |
| | Manager TA decision | 3 | | | |
| | Departmennt TA Decision | 3 | | | |
| | Firnance update TA codes | 3 | | | |
| Outputs | Employee login confirmation | 3 | 5 | | 38 |
| | Enployee's TA completion notification | | | | |
| | Manager TA notificatoin | | 5 | | |
| | Employee notification of Manager Decision | | 5 | | |
| | Departmennt notification of Manager Decision | | 5 | | |
| | Finance Departmnet notification of Department decision | | 5 | | |
| | Manager notification of TA completion | | 5 | | |
| | Employee notification of TA Completion | | 5 | | |
| Inquiries | Validate member information | 3 | | | 11 |
| | Validate supervisor information | | 4 | | |
| | Calculate TA total cost | | 4 | | |
| Internal File | Travel Authorization Database | | | 15 | 15 |
| External Interfaces | Employee to TA Database | 5 | | | 29 |
| | Manager to TA Database | | | 10 | |
| | Finance to TA Database | | 7 | | |
| | Department to TA Database | | 7 | | |
| Total UFP | | | | | 109 |

*Figure 5: Unadjusted Function Points Calculation*

Next, the unadjusted function points must be converted into lines of code based on the programming language. According to Forrester's Clay Richardson, the low-code development platform originates from fourth generation programming languages. The online version of COCOMO II at the University of Southern California's Center for Software Engineering site (https://csse.usc.edu/tools/COCOMOII.php) does not compute for languages above 3rd generation. As seen in Figure 6 below, COCOMO II.2000.4 desktop version accounts for fourth generation languages. The total unadjusted functions points were converted to an equivalent total of 2180 in sources lines of code.


SLOC Input Dialog - <sample>

Sizing Method: SLOC [✓ Function Points] Adaptation and Reuse

Breakage: % of code thrown away due to requirements evolution and volatility

REVL 0.00

Module Size in Function Points

Language: Fourth Generation | Change Multiplier | 20

Ratio Type : [✓ Jones] David

Calculation Method : Using Table [✓ Input Calculated Function Point]

Total Unadjusted Function Points 109

Equivalent Total in SLOC 2180

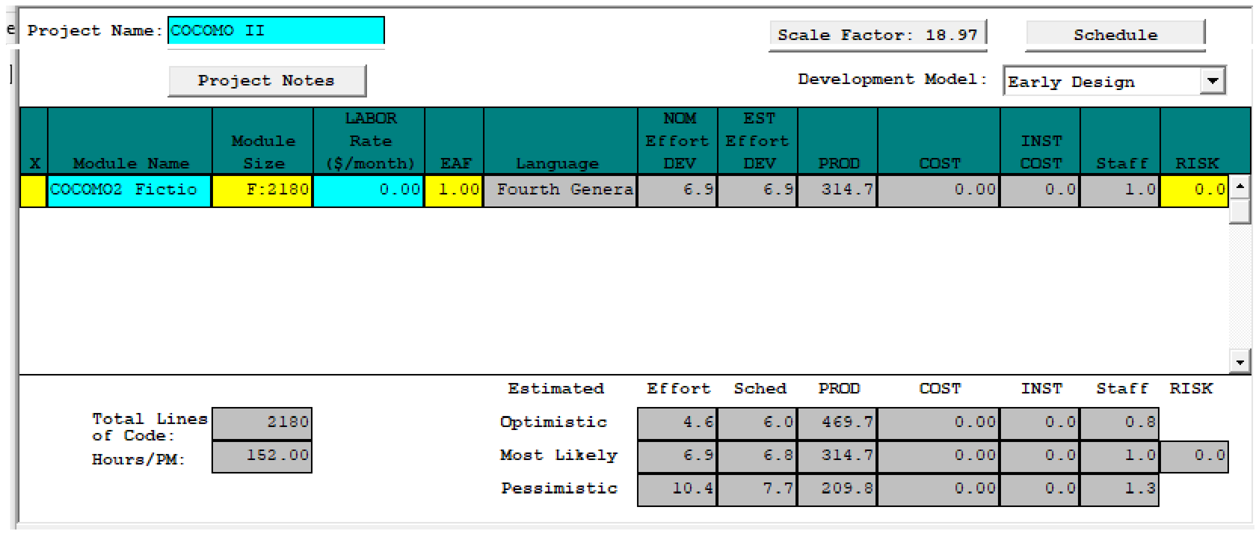
*Figure 6: COCOMO II Unadjusted Function Points Conversion*

Finally, I was able to use the 2108 SLOC as the module size within COCOMO II to successfully calculate effort the hypothetical project.  Figure 7 depicts the results:

Optimistic estimation  = 4.6 person-months

Most Likely estimation =  6.9 person-months

Pessimistic estimation = 10.4 person-months.


Project Name: COCOMO II | Scale Factor: 18.97 | Schedule

Project Notes | Development Model: Early Design

| X | Module Name | Module Size | LABOR Rate ($/month) | EAF | Language | NOM Effort DEV | EST Effort DEV | PROD | COST | INST COST | Staff | RISK |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | COCOMO2 Fictio | F:2180 | 0.00 | 1.00 | Fourth Genera | 6.9 | 6.9 | 314.7 | 0.00 | 0.0 | 1.0 | 0.0 |

| | |
|---|---|
| Total Lines of Code: | 2180 |
| Hours/PM: | 152.00 |

| Estimated | Effort | Sched | PROD | COST | INST | Staff | RISK |
|---|---|---|---|---|---|---|---|
| Optimistic | 4.6 | 6.0 | 469.7 | 0.00 | 0.0 | 0.8 | |
| Most Likely | 6.9 | 6.8 | 314.7 | 0.00 | 0.0 | 1.0 | 0.0 |
| Pessimistic | 10.4 | 7.7 | 209.8 | 0.00 | 0.0 | 1.3 | |

*Figure 7:  COCOMO II Implementation Results*

## 3.4  COnstructive Rapid Application Development MOdel(CORADMO) Implementation

The Constructive Rapid Application Develop Model's was developed in response to the difficulty of effort estimation for agile development methodologies (Ingold 2013).  The model is based on COCOMO II but concentrates on the effects of Life Cycle Objectives, Life Cycle Architecture, and Initial Operational Capability in a development life cycle (Ingold 2013).  According to the Center for Systems and Software Engineering website of the University of California, CORADMO is based on four stages with overlapping workflows as seen in Figure 8.  CORADMO is best suited for:  Development Process Re-engineering (DPRS), Rapid Prototyping (RPRO), Collaboration efficiency (CLAB), Architecture and risk resolution (RESL), Pre-Positioning of assets (PPOS), RAD Capability of Personnel (RCAP).

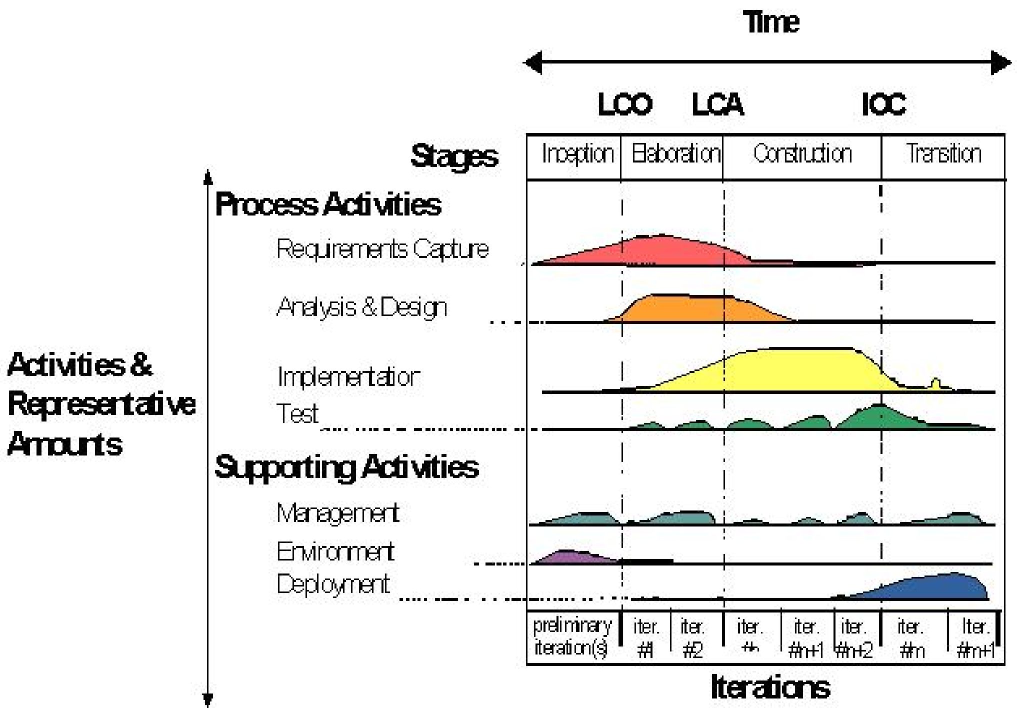
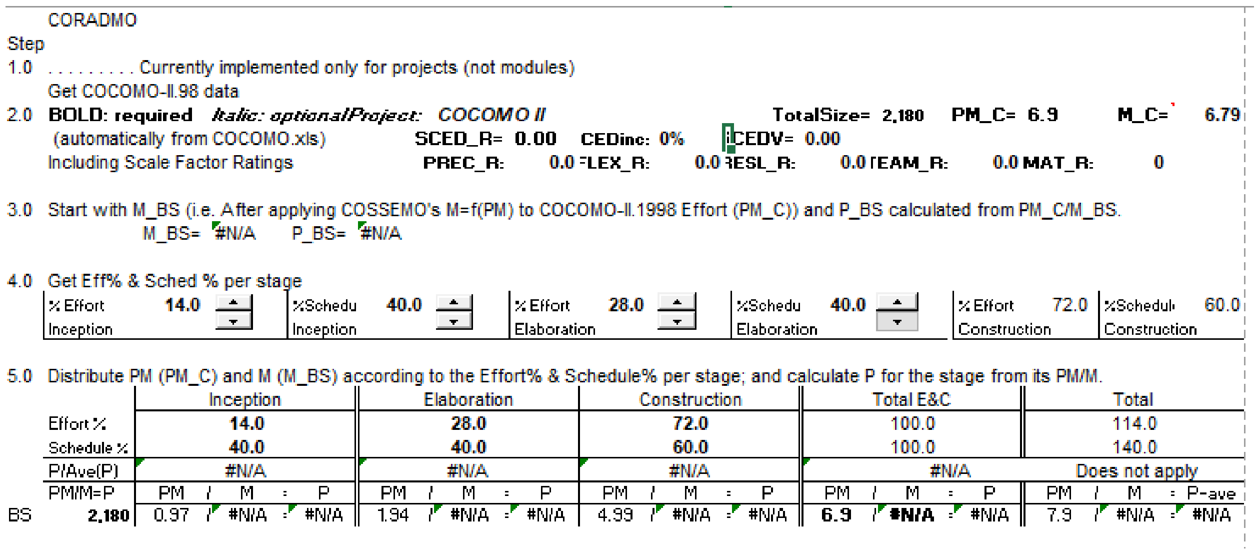


*Figure 8:  CORADMO Stages (CSEE, 2009)*

The first step to implement CORADMO for the hypothetical project is to apply the front end staged schedule and effort model seen in Figure 9 below.  Though similar to the COCOMO II, the schedule estimation uses more complex calculations for situations where effort is below 64 person-months (Winsor 1998).  The initial estimate is 7.9 person-months.



CORADMO

Step

1.0 . . . . . . . . . Currently implemented only for projects (not modules)

Get COCOMO-II.98 data

2.0 **BOLD: required** *Italic: optionalProject:* ***COCOMO II*** **TotalSize= 2,180** **PM_C= 6.9** **M_C= 6.79**

(automatically from COCOMO.xls) **SCED_R= 0.00** **CEDinc: 0%** **CEDV= 0.00**

Including Scale Factor Ratings **PREC_R: 0.0** **FLEX_R: 0.0** **RESL_R: 0.0** **TEAM_R: 0.0** **MAT_R: 0**

3.0 Start with M_BS (i.e. After applying COSSEMO's M=f(PM) to COCOMO-II.1998 Effort (PM_C)) and P_BS calculated from PM_C/M_BS.

M_BS= #N/A P_BS= #N/A

4.0 Get Eff% & Sched % per stage

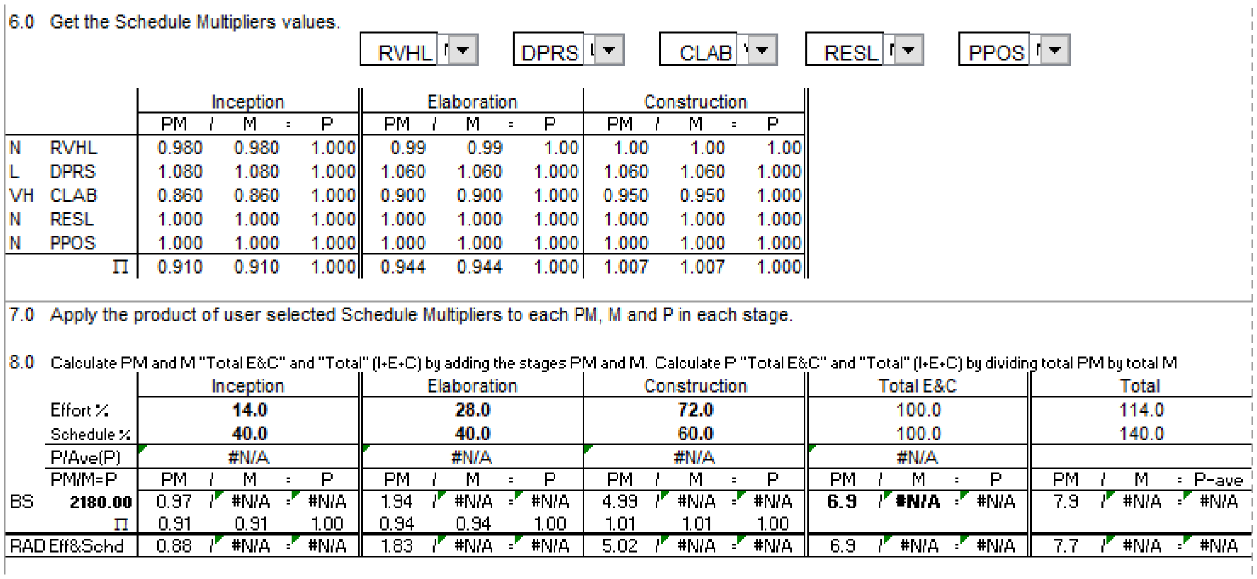
| | | | | | |
|---|---|---|---|---|---|
| % Effort Inception **14.0** | %Schedu Inception **40.0** | % Effort Elaboration **28.0** | %Schedu Elaboration **40.0** | % Effort Construction 72.0 | %Schedul Construction 60.0 |

5.0 Distribute PM (PM_C) and M (M_BS) according to the Effort% & Schedule% per stage; and calculate P for the stage from its PM/M.

| | | Inception | Elaboration | Construction | Total E&C | Total |
|---|---|---|---|---|---|---|
| | Effort % | **14.0** | **28.0** | **72.0** | 100.0 | 114.0 |
| | Schedule % | **40.0** | **40.0** | **60.0** | 100.0 | 140.0 |
| | P/Ave(P) | #N/A | #N/A | #N/A | #N/A | Does not apply |
| | PM/M=P | PM / M : P | PM / M : P | PM / M : P | PM / M : P | PM / M : P-ave |
| BS | **2,180** | 0.97 / #N/A : #N/A | 1.94 / #N/A : #N/A | 4.99 / #N/A : #N/A | **6.9** / **#N/A** : #N/A | 7.9 / #N/A : #N/A |

*Figure 9: CORADMO  Implementation Part 1*

The second part of implementation is to apply the RAD extensions.  The five RAD drivers are input into the RAD extension:  Reuse and VHLLs (RVLH), Development Process Reengineering and Streamlining (DPRS), Collaboration Efficiency (CLAB), Architecture/Risk Resolution (RESL), Prepositioning Assets (PPOS).  The result is a new schedule multiplier.  The final effort estimation is 7.7 person-months as seen in Figure 10.


6.0 Get the Schedule Multipliers values.

RVHL | DPRS | CLAB | RESL | PPOS

| | | Inception | | | Elaboration | | | Construction | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | PM / | M : | P | PM / | M : | P | PM / | M : | P |
| N | RVHL | 0.980 | 0.980 | 1.000 | 0.99 | 0.99 | 1.00 | 1.00 | 1.00 | 1.00 |
| L | DPRS | 1.080 | 1.080 | 1.000 | 1.060 | 1.060 | 1.000 | 1.060 | 1.060 | 1.000 |
| VH | CLAB | 0.860 | 0.860 | 1.000 | 0.900 | 0.900 | 1.000 | 0.950 | 0.950 | 1.000 |
| N | RESL | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| N | PPOS | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 | 1.000 |
| | Π | 0.910 | 0.910 | 1.000 | 0.944 | 0.944 | 1.000 | 1.007 | 1.007 | 1.000 |

7.0 Apply the product of user selected Schedule Multipliers to each PM, M and P in each stage.

8.0 Calculate PM and M "Total E&C" and "Total" (I+E+C) by adding the stages PM and M. Calculate P "Total E&C" and "Total" (I+E+C) by dividing total PM by total M

| | | Inception | | | Elaboration | | | Construction | | | Total E&C | | | Total | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Effort % | | **14.0** | | | **28.0** | | | **72.0** | | | 100.0 | | | 114.0 | |
| | Schedule % | | **40.0** | | | **40.0** | | | **60.0** | | | 100.0 | | | 140.0 | |
| | P/Ave(P) | | #N/A | | | #N/A | | | #N/A | | | #N/A | | | | |
| | PM/M=P | PM / | M : | P | PM / | M : | P | PM / | M : | P | PM / | M : | P | PM / | M : | P-ave |
| BS | **2180.00** | 0.97 | #N/A | #N/A | 1.94 | #N/A | #N/A | 4.99 | #N/A | #N/A | **6.9** | **#N/A** | #N/A | 7.9 | #N/A | #N/A |
| | Π | 0.91 | 0.91 | 1.00 | 0.94 | 0.94 | 1.00 | 1.01 | 1.01 | 1.00 | | | | | | |
| RAD | Eff&Schd | 0.88 | #N/A | #N/A | 1.83 | #N/A | #N/A | 5.02 | #N/A | #N/A | 6.9 | #N/A | #N/A | 7.7 | #N/A | #N/A |

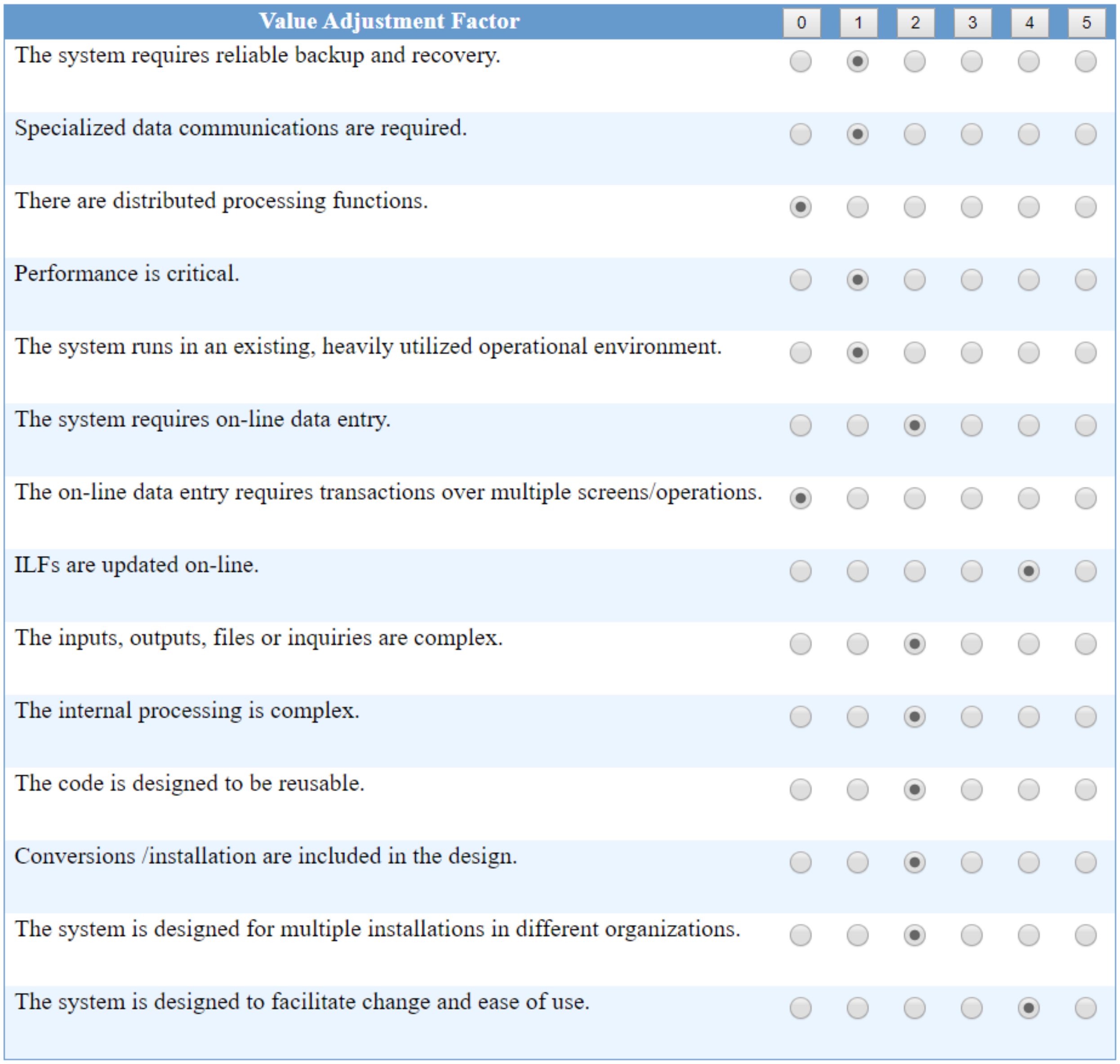
*Figure10: CORADMO Implementation Part 2*

## 3.5 The Web Objects Method Implementation

Reifer recognized the need for an estimation technique for 'estimating quick-to-market software (Reifer, 2000). He proposed an estimation method based on web objects using an ad hoc or work breakdown structure. To apply the Web Objects Method as stated in Reifer's original concept, function points must be computed and then the web components identified and calculated. This method's applicability is an improvement for low-code development because it does not require a programming language specification nor lines of code conversion.

On the other hand, to overcome what Reifer described as the 'challenges for the estimation process with web development', to implement the web objects method you must identify multimedia files, web building blocks, scripts and links. This is generally a non-issue when the low-code development platform is being used to create a web-based product (although in some instances, the fine grained building blocks may be running behind the scenes without the knowledge of the low-code user). Nonetheless, if the application being developed is not web based, the web objects method simply becomes function point analysis.

The first step to implement the web objects method is performing function point analysis. As stated before, this should be a feasible task; however, external interface files and internal logical files may not be easily calculated. Next the Total Degree of Influence (TDI) is calculated using the formula, **$\mathbf{TDI} = \sum^{14}$ Degrees of Influence**. Reference Figure 11 below for the detailed calculation for the general system characteristics. TDI for the fictional project is 24. Next the value adjustment function point is calculated using formula, **$\mathbf{VAF = (TDI \times 0.01) + 0.65}$.** VAF for the fictional project is 0.89. Finally, a calculation for adjusted functions is determined using formula, **Adjusted FP Count = Unadjusted FP Count × VAF.** The result is an adjusted function point count of calculation 97.



| Value Adjustment Factor | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| The system requires reliable backup and recovery. | ○ | ● | ○ | ○ | ○ | ○ |
| Specialized data communications are required. | ○ | ● | ○ | ○ | ○ | ○ |
| There are distributed processing functions. | ● | ○ | ○ | ○ | ○ | ○ |
| Performance is critical. | ○ | ● | ○ | ○ | ○ | ○ |
| The system runs in an existing, heavily utilized operational environment. | ○ | ● | ○ | ○ | ○ | ○ |
| The system requires on-line data entry. | ○ | ○ | ● | ○ | ○ | ○ |
| The on-line data entry requires transactions over multiple screens/operations. | ● | ○ | ○ | ○ | ○ | ○ |
| ILFs are updated on-line. | ○ | ○ | ○ | ○ | ● | ○ |
| The inputs, outputs, files or inquiries are complex. | ○ | ○ | ● | ○ | ○ | ○ |
| The internal processing is complex. | ○ | ○ | ● | ○ | ○ | ○ |
| The code is designed to be reusable. | ○ | ○ | ● | ○ | ○ | ○ |
| Conversions /installation are included in the design. | ○ | ○ | ● | ○ | ○ | ○ |
| The system is designed for multiple installations in different organizations. | ○ | ○ | ● | ○ | ○ | ○ |
| The system is designed to facilitate change and ease of use. | ○ | ○ | ○ | ○ | ● | ○ |

*Figure 11: General System Characteristics used in TDI Calculations*

The next step is to calculate the number of web-related components that were not included in the function point analysis (Di Martino, 2011). Reifer proposed using Halstead's Equation for Volume, $V = N \log_2 (n) = (N_1 + N_2) \log_2(n_1 + n_2)$, with the operators and operands being calculated using the *Web-Based Size Predictors* information found in Figure 12. For a low-code developed application that is not web based, this step could possibly be ignored.

| **Web-Based Size Predictors** | | |
|---|---|---|
| **Web Object Predictors** | **Example Operands** | **Example Operators** |
| **# of building blocks** | Widgets, fined-grained components (Active X, DCOM, OLE, etc) | Create, apply, call, dispatch, interface, terminate, etc |
| **# of COTS components** | COTS programs, library routines, web objects (carts), etc. | Transform, access, bind, generate, interface, etc. |
| **# of multi-media files** | Text, video, audio, etc. (not graphic files) | Create, cut, paste, clear, edit, etc. |
| **# of application or object points (or others proposed)** | # server tables, # client tables, # states, # entities, attributes, etc | Transform, access, modify, instantiate, generate, etc. |
| **# of web components** | Applets, agents, guards, etc. | Create, schedule, dispatch, etc. |
| **# of xml, sgml, html and query lines** | Lines including links to data attributes | Create, call, browse, link, traversal, etc. |
| **# of graphic files** | Templates, pictures, images, etc. | Apply, align, import, export, insert, etc. |
| **# of scripts (visual language, audio, motion, etc.)** | Macros, containers, etc | Create, store, edit, distribute, serialize, generate, etc. |

*Figure 12: Web-Based Size Predictors (Reifer, 2000)*

When attempting to calculate the web-based size predictors for the fictional project, some issues become apparent: how do you count the number of COTS components and how do you count the operators? The entire application is being created using ‘off the shelf’ components. Would you count the application, a template, a workflow, or a workflow graphic as component? After clarifying the operand definitions, there are still some predictors that are not available because the low-code development platform hides the mechanics from the user. In some instances, the operators are unknown. Therefore, it is impossible to accurately apply Halstead’s model to the fictional project. Figure 13, *Fictional Project Web-Based Size Predictors*, shows the unsuccessful attempt to apply Halstead’s model. The effort estimation is stopped with the adjusted function point calculation of 97.

| **Fictional Web-Based Size Predictors** | | |
|---|---|---|
| **Web Object Predictors** | **Operands** | **Operators** |
| **# of building blocks** | At least 5 | Create, apply, call, dispatch, interface, terminate, etc |
| **# of COTS components** | At least 30 | Transform, access, bind, generate, interface, etc. |
| **# of multi-media files** | 0 | 0 |
| **# of application or object points (or others proposed)** | At least 2 | Unsure. |
| **# of web components** | Unsure | Create, schedule, dispatch, etc. |
| **# of xml, sgml, html and query lines** | Unsure | Create, call, browse, link, traversal, etc. |
| **# of graphic files** | 0 | Apply, align, import, export, insert, etc. |
| **# of scripts (visual language, audio, motion, etc.)** | Unsure | Create, store, edit, distribute, serialize, generate, etc. |

*Figure 13: Fictional Project Web-Based Size Predictors*

## 3.6 Work Break-Down Structure Implementation

Although work break-down structure is often seen as a technique used by project managers, this expert judgement method is commonly used to answer: "How long will this take to complete?" (McConnell 2006),(Spyrosktenas 2013). The project is decomposed into smaller pieces and effort estimations are assigned to each level. The weakness and strength of the method is the reliance on expert estimation. In low-code development platforms, the availability of historical projects for comparison may be small. Experts may be non-existent. When neither of these items are available, task effort assignments are decided based on the developers' past experiences.

When applying the work break-down methodology using personal experience to assign estimates, the effort total is 2.25 person months. Figure 14, *Work Break-down Implementation for Fictional Project* contains the detailed deconstruction of tasks. The effort total is subjective based exclusively on the amount of time assigned each task. In the future, more accurate estimates can be obtained as familiarity with the low-development platform grows, more experts emerge, and historical project data is available(McConnell 2006),(Spyrosktenas 2013).

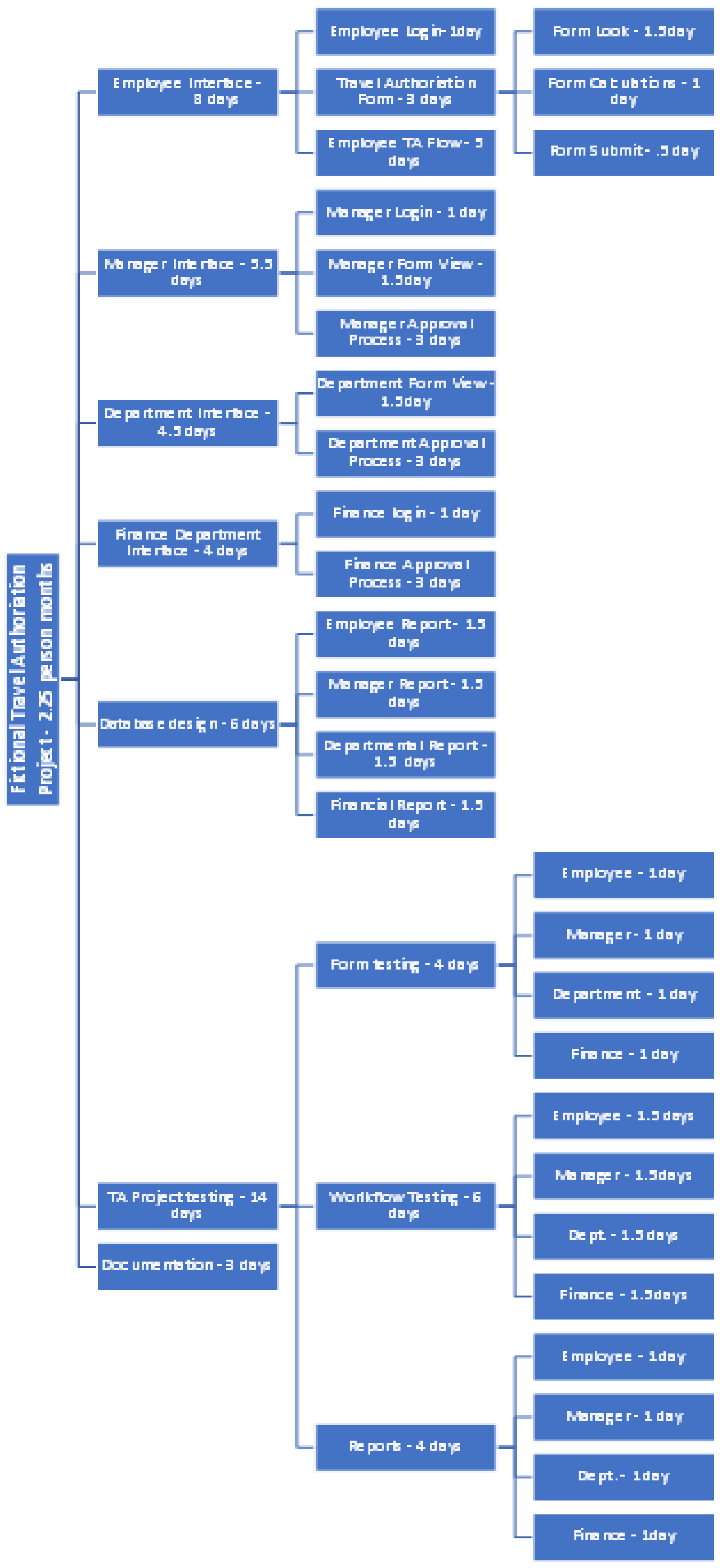

Fictional Travel Authorization Project - 2.25 person months
Employee Interface - 8 days
Employee Login- 1day
Travel Authorization Form - 3 days
Employee TA Flow - 5 days
Form Look - 1.5day
Form Calculations - 1 day
Form Submit - .5 day
Manager Interface - 5.5 days
Manager Login - 1 day
Manager Form View - 1.5day
Manager Approval Process - 3 days
Department Interface - 4.5 days
Department Form View - 1.5day
Department Approval Process - 3 days
Finance Department Interface - 4 days
Finance login - 1 day
Finance Approval Process - 3 days
Database design - 6 days
Employee Report - 1.5 days
Manager Report - 1.5 days
Departmental Report - 1.5 days
Financial Report - 1.5 days
TA Project testing - 14 days
Form testing - 4 days
Employee - 1day
Manager - 1 day
Department - 1 day
Finance - 1 day
Workflow Testing - 6 days
Employee - 1.5 days
Manager - 1.5days
Dept. - 1.5 days
Finance - 1.5days
Reports - 4 days
Employee - 1day
Manager - 1 day
Dept. - 1day
Finance - 1day
Documentation - 3 days

*Figure 14: Work Break-down Implementation for Fictional Project*

## 3.7 Analogy and Comparison Implementation

To implement an effort estimation for the fictional project using the analogy-based technique requires historical data from similar projects. When there are no similar projects, modules or activities that are similar to those in the current project can be used as a basis for the estimate (McConnell 2006), (Spyrosktenas 2013).

The complication with the application of this methodology is the lack of historical data for projects using the same low-code development platform. Like the work break-down structure method, the implementation of this estimation methodology will become feasible as use of the methodology increases. Unfortunately, without access to similar projects, an estimation cannot be completed for the fictional project using analogy or comparison.

## 3.8 Bayesian Approach Implementation

The Bayesian approach is an expert-opinion methodology. To implement the Bayesian approach successfully, a need exists for at least three experts who have worked on projects similar to the hypothetical project. Without the needed input, it is impossible to create the effort estimation. for historical data exists; therefore, the initial implementation is not feasible. Similar to the analogy and comparison methodology, the use of this technique will be feasible as more users adopt and share their experience with low-code development platforms(McConnell 2006).

# CHAPTER 4. VIABLE SOLUTION FOR LOW-CODE EFFORT ESTIMATION

To reiterate, low-code platforms expedite application delivery by providing visual tools for the quick definition and assembly of forms (Rubens 2014). Estimating effort for this type of project requires a size metric that does not depend on lines of code and is programming language independent (Reifer 2000).

`This research focused on the applicability of some commonly used effort estimation techniques to create effort estimate for low-code development platforms. This research revealed the need for more studies concerning low-code development. More research must be performed to better categorize low-code platforms. Is it low-code or no-code in nature? Is it an application platform running as a service? Does it integrate with existing technology? How customizable is the platform? What is the minimum functionality that quantifies the tool as a low-code platform? The proposed questions are some topics which should be answered in future projects. As a standard definition is created for 'low-code platform', the process of creating a future effort estimation model will become more feasible.

Based on this research, I suggest an effort estimation methodology for low-code development projects similar to function point analysis but with emphasis on the following factors as web-size predictors: number of user inputs, number of external connections and their complexities, workflow complexity, developer experience with the platform, customizations, and platform documentation. A depiction is seen in Figure 15 below.

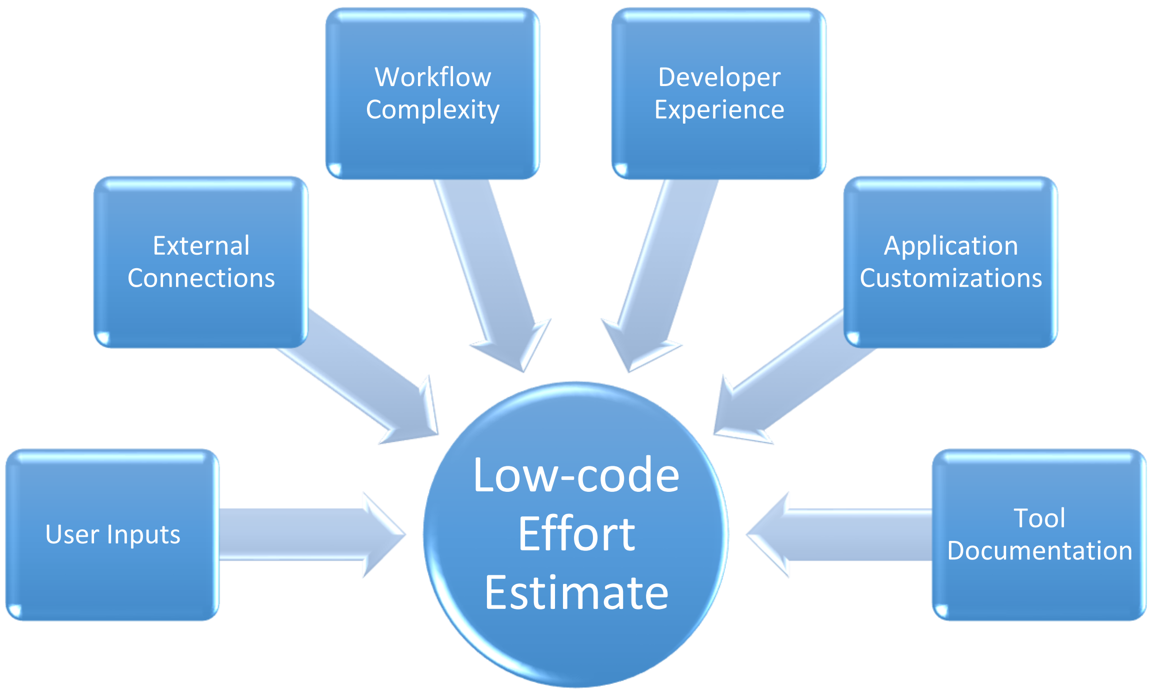


*Figure 15: Proposed Low-Code Effort Estimation Method*

Further information on the web-size predictors follows:

**User Inputs**: The number of user input forms and their complexity should be accounted for in estimating effort. Forms created directly from templates are low in complexity. Forms with minor customizations such as field name changes or other cosmetic changes would be moderate. Forms with external connections and more in depth designs such as repeating tables and external connections would be complex.

**External Connections:** Will the new application connect to a server, a database, or integrate with an existing system? Does the platform contain built in connectors to perform these tasks or will the connections need to be manually created?

**Workflow Complexity**: Are the processes followed that produce the desired results basic, moderate or complex? Do the steps require approvals before advancement? Are there parallel sequences that occur? Are there numerous decision points, rules and\or dependencies? Do the workflows cross to other systems?

**Project Customizations:** Is this new project being created based on a template with little or no change? Or will the creator need to edit workflows to fit organizational process? Will the look and style need modification? Will the creator need to develop new features to match the organization's specification?

**Experience with the platform:** Is this low-development platform new to the team? Have they created projects with the tool in the past? Will the 'developers' need additional training on the tool for this project? Are they well versed with the strengths and weaknesses of the platform?

**Platform documentation:** The accompanying files that explain how to use the low-code platform. Does the documentation exist? Is it well written and easy to understand? Are there user guides, a knowledge base, online training, or videos? Is there an existing user community to assist with problems?

**Calculation:** To calculate low-code development effort,

- Tally the number of user inputs, external connections, and workflows.
- Determine the complexity category for each factor.
    - When a factor has more than one line item, determine the complexity for each item.
    - If there is only one count, determine the complexity for that factor in its entirety.
- Apply the appropriate weight for each complexity
- Multiply the count by the weight for each line item to produce the 'total'. The weight for project customization, experience with platform, and platform documentation is always equal to '1'.
- Sum the items in the total column to create the 'effort total'

There is a direct correlation between the 'effort total' and the complexity of the application being developed. As complexity increases, the 'effort total' increases.
See Figure 16 below for an example worksheet which could be used to calculate 'effort total'.

| FACTOR | COUNT | COMPLEXITY | WEIGHT | TOTAL |
|---|---|---|---|---|
| **User Inputs**<br>**Low**<br>**Moderate**<br>**Complex** | # of user inputs | | Low=2<br>Moderate=3<br>Complex=5 | |
| **External Connections**<br><br>**Database**<br>**Server**<br>**Existing System Integration**<br>**Other** | # of external connections | | Database=2<br>Server=2<br>Existing System Integration=3<br>Other=3 | |
| **Workflow Complexity**<br>**Low**<br>**Moderate**<br>**Complex** | # of work flows | | Low=2<br>Moderate=3<br>Complex=5 | |
| **Project Customization**<br>**Low**<br>**Moderate**<br>**Complex** | 1 | | Low=2<br>Moderate=3<br>Complex=5 | |
| **Experience with Platform**<br>**Extensive**<br>**Moderate**<br>**None** | 1 | | Extensive=1<br>Moderate=3<br>None=5 | |
| **Platform Documentation**<br>**Extensive**<br>**Moderate**<br>**None** | 1 | | Extensive=1<br>Moderate=3<br>None=5 | |
| **EFFORT TOTAL** | | | | |

*Figure 16: Low-Code Development Effort Estimation*

Implementation and testing of the suggested low-code development effort estimation method was outside the scope of this research.  Further research should answer:

- How does the value created in the ‘effort total’ field translate to time in the real world?
- What improvements could be generated to allow accurate estimation?
- Should gearing factors be introduced to standardize between the low-code development platforms.

It is equally important to review the applicability of other models as historical data is gathered.  As more data is accumulated, the likelihood of a gearing factor increases.  Ultimately the goal exists to be able to confidently answer the question, ‘how long will it take to complete this project using a low-code development platform?’

# CHAPTER 5. CONCLUSIONS

Low-code platforms make it possible to create and deploy applications in a fraction of the amount of time of traditionally developed software by reducing the effort of manual coding. The increase in the usage of low-code platforms combined with the gap in the current body of knowledge in estimating project size for this type of development are key motivating factors in performing this research. Project success of future low-code projects will be directly impacted by the lack of low-code project estimation methodologies.

The main purpose of this research was to investigate the historical use of some commonly used project size estimation techniques, theoretically apply the techniques to a hypothetical low-development project, and determine which methods are suitable for use with low-code development projects.

## 5.1 Research Summary

The question "how do I measure size?" has not been definitively answered. Most traditional methods continue to use lines of code as the standard measurement. This is due in part to the length of time it has been around, the familiarity in the industry, and the fact that lines of code correlates well with functionality and effort. However, due to the inaccessibility of the source code within low-code platform applications, using this sizing metric is difficult. Another drawback is that it can be misleading at a micro-level. Generally, an efficient programmer can write a function in fewer lines than a novice. In the same way, low-code platforms have optimized functionality to be completed in as little code as feasible (Ingold 2103). As stated previously, it may take seconds to drag and drop a template onto a canvas which could equate to hundreds of lines of code.

As Reifer stated in his article, *Web Development: Estimating Quick-to-Market Software,* SLOC is not suitable for early estimation because web projects are design based. At times, function points may not be appropriate because the complexity of the applications. Although CORADMO attempts to address estimation for rapid deployment, it often attempts to solve the estimation problem by assigning more personnel to work on projects that are under six months in length (Ingold 2013).

Currently, methodologies that rely on historical data or expert judgement cannot calculate effort for low-code platforms due to the lack of historical data and 'experts'. As historical data is gathered and more experts are created, these tools will prove useful in estimating effort for low-code platforms.

The web object method did not provide a complete estimate for this fictional project due to the definition of its operands; however, as suggested in the Future Direction section, if the definition of operands and operators was updated to reflect modern programming it could provide better results for predicting effort for web application development over function point analysis (Di Martino, 2011).

In conclusion, three of the existing methodologies produced a successful estimation for the low-code development project: COCOMO II, CORADMO and Work Breakdown Structure. The estimates ranged from 2.25 person months to a pessimistic estimate of 10.4 person months with a standard deviation of 2.93 person months. The other approaches were unsuccessful due to inaccessibility of some parameters to complete the calculations, lack of historical projects for comparison and the dearth of experts to provide estimats.

## REFERENCES

Boehm, B., Clark, B., Horowitz, E. Brown, A.W., Reifer, D. Chulani, S., Madachy, R., and Steece, B. (2000).*Software Cost Estimation with Cocomo II with Cdrom*,1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Bogdan Stepien(2003). ”Software Development Cost Estimation Methods and Research Trends”,Computer Science, Vol.5,2003.

Brown, A. Winsor (1998). “CORADMO Summary”. University of Southern California.

Center for Systems and Software Engineering (n.d.). Retrieved November 03, 2016, from http://csse.usc.edu/csse/research/COCOMOII/cocomo_main.html

Effort Estimation for Software Development. (n.d.). Retrieved December 06, 2016, from http://open-works.org/blogs/effort-estimation-software-development

Di Martino S., Ferrucci F., Gravino C., Sarro F. (2011) Using Web Objects for Development Effort Estimation of Web Applications: A Replicated Study. In: Caivano D., Oivo M., Baldassarre M.T., Visaggio G. (eds) Product-Focused Software Process Improvement. PROFES 2011. Lecture Notes in Computer Science, vol 6759. Springer, Berlin, Heidelberg

Ford J., Colburn, R., and Morris, R. (2012) “Principles of Rapid Acquisition and Systems Engineering,” Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.

Five qualities that define Low-code development platforms. (2015, November 17). Retrieved December 05, 2016, from http://www.matssoft.com/news/five-qualities-that-define-low-code-development-platform/

Halstead, M. H. (1977). *Elements of software science*. New York: Elsevier.

Ingold, D., Boehm, B., & Koolmanojwong, S. (2013). A model for estimating agile project process and schedule acceleration. *Proceedings of the 2013 International Conference on Software and System Process - ICSSP 2013*. doi:10.1145/2486046.2486053

Jodpimai, P., Sophatsathit, P., & Lursinsap, C. (2009, 09). Analysis of effort estimation based on software project models. *2009 9th International Symposium on Communications and Information Technology*. doi:10.1109/iscit.2009.5341149

Jones, C. (1996). *Applied Software Measurement, Assuring Productivity and Quality,* McGraw-Hill, New York, N.Y.

Jorgensen, M. (2014). What We Do and Don't Know about Software Development Effort Estimation. *IEEE Software, 31*(2), 37-40. doi:10.1109/ms.2014.49

Jørgensen, M., & Moløkken-Østvold, K. (2006, 04). How large are software cost overruns? A review of the 1994 CHAOS report. *Information and Software Technology, 48*(4), 297-301. doi:10.1016/j.infsof.2005.07.002

Jørgensen, M. (2004, 02). A review of studies on expert estimation of software development effort. *Journal of Systems and Software, 70*(1-2), 37-60. doi:10.1016/s0164-1212(02)00156-5

Li, Y. F., Xie, M., & Goh, T. N. (2007, 12). A study of genetic algorithm for project selection for analogy based software cost estimation. *2007 IEEE International Conference on Industrial Engineering and Engineering Management*. doi:10.1109/ieem.2007.4419393

McConnell, Stephen. (2006). *Software estimation: Demystifying the black art*. Redmond, WA: Microsoft Press.

Ochodek, M., Nawrocki, J., & Kwarciak, K. (2011, 03). Simplifying effort estimation based on Use Case Points. *Information and Software Technology*, *53*(3), 200-213.

doi:10.1016/j.infsof.2010.10.005

Reifer, D. (2000). Web development: Estimating quick-to-market software. *IEEE Software, 17*(6), 57-64. doi:10.1109/52.895169

Rubens, P. (2014, November 10). Use Low-Code Platforms to Develop the Apps Customers Want. Retrieved December 04, 2016, from http://www.cio.com/article/2845378/development-tools/use-low-code-platforms-to-develop-the-apps-customers-want.html

Skwirzynski, J. K. (1986). *Software system design methods: The challenge of advanced computing technology*. Berlin: Springer-Verlag.

Spyrosktenas.(2013 August 8)Effort Estimation for Software Development [Blog Post] . Retrieved from http://open-works.org/blogs/effort-estimation-software-development

Tackabery, Michelle (2014)"Low-Code" Development Explained. Retrieved from https://www.progress.com/blogs/-low-code-development-explained

*The  Forrester Wave™: Low-Code Development Platforms*, Forrester Research Inc., Q2 2016.

*The  Forrester Wave™: Low-Code Development Platforms For AD&D Professionals*,  Forrester Research Inc., Q1 2019.