

SISE: A Novel Approach to Individual Effort Estimation

by

Russell Thackston

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
August 3, 2013

Keywords: effort, estimation, individual

Copyright 2013 by Russell Thackston

Approved by

David Umphress, Chair, Associate Professor of Computer Science and Software Engineering
James Cross, Professor of Computer Science and Software Engineering
Dean Hendrix, Associate Professor of Computer Science and Software Engineering

Abstract

Author's note: This dissertation has been prepared following a manuscript style. Each chapter has been constructed as a stand alone manuscript, suitable for separate publication. Therefore, each chapter contains an abstract and introductory subject material, as well as some overlapping content.

The software engineering discipline is filled with many varied examples of software process methods and tools focused on the team or organization. In recent years, the agile approach to software engineering has increased the focus of software process on small teams and individuals; however, not all aspects of software process have been deeply or fully addressed.

The majority of effort estimation models – traditional and agile – focus on teams or groups of software engineers. The discipline is ripe with various examples of team-based models including Wideband Delphi, Planning Poker, function point analysis, COCOMO, etc. The few examples of effort estimation models focused on the lone software engineer are limited to traditional mathematical models with (relatively) substantial complexity and required time investment; the Personal Software Process (PSP) contains one such model: PROBE. The discipline lacks a truly agile model based on a minimal combination of empirical data and expert judgment.

The SISE model under development at Auburn University's microISV Research Lab is a simple to understand, lightweight, and agile effort estimation model that specifically targets individual software engineers. SISE combines an individual's personal, empirical data with his or her expert judgment and experiences to produce relatively accurate estimates with a minimal investment of training and time.

The SISE model rests on two foundational principles. First, software engineers are capable of identifying the largest of a pair of tasks based solely on their descriptions. Second, a software engineer who is presented with a future work activity is capable of identifying two historical tasks – one larger, one smaller – which may serve as a prediction of the future activity’s size.

The name “SISE” is an acronym for the model’s four basic steps: Sort, Identify, Size, and Evaluate. The first step – Sort – involves the ordering of historical data by the actual effort required to complete the activity. The second step – Identify – involves choosing two tasks from the historical data set: one confidently known to be smaller, one confidently known to be larger, and both relatively close in size to the future work. Once the practitioner has chosen a pair of tasks, the third step – Size – produces a rough prediction interval of the future activity’s size using the actual effort values for the two completed tasks. The final step – Evaluate – involves shifting or resizing the prediction interval to account for any historical bias. This last step is optional and is only applied if the estimator is dissatisfied with the precision, accuracy, or confidence level of his or her estimate.

Validation of the SISE model included two major steps. First, the foundational principle that relative tasks sizing by software engineers is suitably accurate was validated. The validation occurred in the form of a survey, presented to over 100 software engineering students, which presented the respondents with a series of task pairs from which they were to identify the larger. Some of the pairs had a known, verifiable size difference based on ten years of time logs provided by students in the Software Process course, while some of the pairs did not. The results indicated that, on average, a majority of software engineers were able to identify the larger task, while not typically misidentifying the smaller. When presented with tasks demonstrating no significant difference in size, the respondents were typically swayed by the wording, format, or word count.

The second phase of validation involved a series of Software Process students who were asked to identify where a future activity should be placed in the ordered list of their completed tasks. In addition, the students were asked to construct a PCSE estimate. The results indicated that SISE predictions were no more or less accurate than the PCSE model's estimates. In addition, the students indicated that SISE, in their opinion, took less time and was based on less a complex model. In summary, SISE appears capable of producing results of equal quality, in less time, and with less training.

Acknowledgments

I would like to acknowledge David Umphress for his extensive contributions, both as a mentor and editor. I would also like to thank Laura Thackston – my wife, my peer, my best friend – for keeping me grounded and on track.

Table of Contents

Abstract	ii
Acknowledgments	v
List of Figures	ix
List of Tables	x
1 Individual Effort Estimating: Not Just for Teams Anymore	1
1.1 Abstract	1
1.2 Introduction	1
1.3 Rise of the one-person team	2
1.4 The plight of the individual estimator	3
1.5 Estimation landscape	5
1.6 Estimation for the one-person team	6
1.7 A “Better than Guessing” Agile Approach	8
1.8 Conclusion	10
1.9 Further Information	10
2 Support for Individual Effort Estimation	11
2.1 Abstract	11
2.2 The Importance of Effort Estimation	11
2.3 Effort Estimation and the Individual	13
2.4 Effort Estimating Tools, Techniques, and Approaches	13
2.5 Common Estimation Approaches	15
2.6 Common Estimation Tools and Techniques	19
2.7 Measuring Estimation Quality	22
2.8 Estimation Techniques for the Individual	28

2.9	Summary and Conclusions	30
3	SISE: A Novel Approach to Individual Effort Estimation	32
3.1	Abstract	32
3.2	Introduction	32
3.3	The SISE Steps	34
3.3.1	Step 1: Sort	34
3.3.2	Step 2: Identify	34
3.3.3	Step 3: Size	34
3.3.4	Step 4: Evaluate	35
3.4	Getting Started with SISE	38
3.5	Accuracy, Precision, and Confidence Level	39
3.6	SISE Example	41
3.7	Validation of SISE	45
3.8	Conclusion	45
4	Validation of the SISE Model	47
4.1	Abstract	47
4.2	Introduction	48
4.3	SISE: An Agile Estimation Model	49
4.4	Relative Sizing	50
4.4.1	Hypothesis	51
4.4.2	Survey	51
4.4.3	Metrics	53
4.4.4	Participants	54
4.4.5	Questions and Presentation	55
4.4.6	Results	55
4.4.7	Individual Respondents	64
4.4.8	Conclusions	65

4.5	Accuracy	65
4.5.1	Hypothesis	66
4.5.2	Experiment	66
4.5.3	Metrics	67
4.5.4	Participants	68
4.5.5	Questions and Presentation	69
4.5.6	Results	70
4.5.7	Conclusions	71
4.6	Time Investment and Perceived Value	71
4.6.1	Hypothesis	72
4.6.2	Survey	73
4.6.3	Metrics	73
4.6.4	Participants	74
4.6.5	Questions and Presentation	74
4.6.6	Results	76
4.6.7	Conclusions	77
4.7	Summary	77
5	Conclusions and Additional Research	79
5.1	Summary	79
5.2	Conclusions	81
5.3	Additional Research	81
	Bibliography	85
	Appendices	89
A	Output Listings	90
B	Themes in Relative Sizing Rationale	101
C	Relative Sizing Survey Questions	103
D	Attitudinal Survey Questions	114

List of Figures

4.1	Distribution of actual construction times (all data).	52
4.2	Distribution of actual construction times (values < 1,000 minutes).	52
4.3	Relative sizing survey results.	56
4.4	Distribution of percentage of correct answers (task pairs 1-4).	65
4.5	Sample PROBE calculation using the assignment spreadsheet.	69

List of Tables

2.1	Common approaches to estimation.	16
2.2	Common approaches to estimation.	23
2.3	Sample MARE values.	25
3.1	One completed activity.	41
3.2	Two completed activities.	42
3.3	Three completed activities.	42
3.4	Four completed activities.	43
3.5	Ten completed activities.	44
3.6	Adjusting for width bias.	44
4.1	Average construction effort (in minutes).	52
4.2	Average construction time comparison for assignment pairs.	53
4.3	Relative Sizing Survey Results.	56
4.4	Task pair 1 distribution of themes in respondents' rationales.	57
4.5	Task pair 2 distribution of themes in respondents' rationales.	58
4.6	Task pair 3 distribution of themes in respondents' rationales.	59
4.7	Task pair 4 distribution of themes in respondents' rationales.	60
4.8	Task pair 5 distribution of themes in respondents' rationales.	61
4.9	Task pair 6 distribution of themes in respondents' rationales.	61
4.10	Task pair 7 distribution of themes in respondents' rationales.	62
4.11	Task pair 8 distribution of themes in respondents' rationales.	63
4.12	Task pair 9 distribution of themes in respondents' rationales.	63
4.13	Sample assignment summary.	70
4.14	Summary of survey results.	76

Chapter 1

Individual Effort Estimating: Not Just for Teams Anymore

Author's note: This manuscript was originally published in the May/June 2012 edition of CrossTalk: Journal of Defense Software Engineering.

1.1 Abstract

Truly viable software – mobile device apps, services, components – are being written by one-person teams, thus demonstrating the need for engineering discipline at the individual level. This article examines effort estimation for individuals and proposes a lightweight approach based on a synthesis of a number of concepts derived from existing team estimation practices.

1.2 Introduction

Software engineering is usually portrayed as a team activity, one where a myriad of technical players are choreographed to build a software solution to a complex problem. With such a perspective, estimating the effort required to write software involves a top-down view of development projects. Effort is forecasted based on a characterization of previous projects that is intended to represent teams performing to the statistical norm. While large projects are predominant – especially in the DoD environment – the fact remains that teams are made of unique individuals, each of whom write software at a different tempo and with different properties. At some point in a project, the top-down prediction of required effort must be tempered with a bottom-up frame of reference in which the team fades from being a group of generic members to being a collection of distinct persons.

Historically, estimation methods have focused on effort at the team level. Recent agile software development practices have shed light on taking individuals – who are acting as part of a team – into consideration. Emerging trends in software development for mobile devices suggest that effort estimation methods can be employed for one-person endeavors and that those methods can benefit teams, but also that such methods are still very much in their infancy. This paper examines effort estimation from the one-person team perspective and describes a lightweight effort estimation technique in that light.

1.3 Rise of the one-person team

One-person software companies – historically referred to as shareware authors, but more recently labeled “micro Independent Software Vendors,” or microISVs – are on the rise, fueled in part by the proliferation of free and open source tools and new eco-systems, such the Apple App Store and the Android Market. In fact, metrics gathered by AdMob – a mobile advertising network – estimated the market for paid mobile apps in August of 2009 at \$203 million [1]. That is an estimated \$2.4 billion per year, for mobile apps alone, where the greatest barrier to entry is the cost of the mobile device.

For example, publishing a game app in the Apple App Store requires an investment of around \$100, not including the device, as the only requirement is the \$99/year to join the Apple iOS Developer Program [4]. Publishing an Android app is even less costly since the developer membership fee is only \$25 [3]. Both platforms boast generous support documentation, which is freely available on the Apple and Google websites, respectively. Furthermore, many books are available for app authors wishing to dive deeper into the technologies of these platforms.

In addition to the mobile app market, opportunities exist in other markets, such as the traditional downloadable software venue as well as the relatively new concept of Software as a Service (SaaS) websites. Gartner Research measured worldwide SaaS revenue for 2010 at \$10 billion and should reach \$12.1 billion in 2011 [14].

MicroISVs encapsulate the entire spectrum of company activities into a single individual. As a result, the successful microISV focuses only on those activities that provide clear, measurable benefit and contribute directly to the bottom line; all other activities are deemed unimportant and are, as a result, discarded. It is through this natural selection process that successful microISV founders learn what is essential to the operation of a business and what is not. Unfortunately, the benefits derived from certain activities may not be directly visible. Many business activities – such as planning and risk management – are proven to provide measurable benefit, but may seem unimportant by a microISV operator confronted with the daily operations of a business. Too often, this category includes non-technical activities, such as taking the time to understand the size and scale of upcoming work (e.g. estimating effort).

1.4 The plight of the individual estimator

MicroISVs are not the only one-person software development operations in action. Regardless of the size of the team (enterprise, company, firm, etc.), software development still boils down to the individual on the front line: the developer. No matter what label is given to the developer – microISV owner, consultant, or team member – it is his or her responsibility to help craft estimates and to manage his or her own time. The motivation varies, depending on the circumstances. For example, microISV owners might primarily use estimates to plan release cycles and orchestrate business and technical tasks. Consultants might use personal estimates to provide billing estimates. Team members could use personal estimates to validate requested delivery dates and coordinate time allocations among multiple activities and projects. However, individual software engineers remain generally ill equipped to construct personal estimates. This leaves the individual developer faced with the choice between guesswork, formal models, and/or relying on team-oriented techniques.

Unfortunately, team members, too, fall into behaviors which either avoid giving proper attention to estimating effort, or they fall back on one of the least accurate approaches: “gut feel” estimates.

Why do individual software developers not value good effort estimates? An informal survey of members of the Association of Software Professionals (ASP) reveals the perception that developing an estimate provides no direct value, either to microISV owners or their customers, the primary reason being that microISV product requirements are too fluid and are, therefore, not a good basis for an estimate [5]. Some microISV owners indicate that effort estimates are unnecessary unless a customer or client is holding them accountable; however, this is not often the case for microISVs, which tend to focus on shrink-wrapped products.

Many of the same arguments expressed by microISVs apply to individual developers acting as team members. Developers on a team may not be asked for effort estimates; someone in a senior position may directly provide deadlines to them. In the event that a developer is asked for an estimate, it is likely they are not properly equipped to provide a good estimate or they do not view this non-technical activity as an interesting and engaging problem, like coding. This can lead to an estimating rule of thumb, such as “make a guess and multiply by three.”

Regardless of the circumstances and rationale, many individual developers are not equipped to construct and derive benefit from personal effort estimates. The spectrum of effort estimation approaches is broad, often overwhelming so. At the near end of the spectrum lies guesswork; at the far end lies formal models. The former is quick, but difficult to tune; the latter involves complex mathematical calculations to model past performance and requires a heavy investment of time. The agile software development movement strives to strike a conciliatory balance that takes advantage of individual expert opinions adjusted by the collective wisdom of multiple participants. Planning Poker, for example, exemplifies this philosophy of guessing effort individually then attenuating the range of guesses through

team dialog [9]. Up to this point, there are no single person equivalents to Planning Poker (i.e., “Planning Solitaire”).

Additionally, developers do not see a benefit to themselves in constructing an effort estimate. This is due in part to the tools and techniques currently available to the individual, which are either too heavyweight or non-existent. Either the process of constructing the estimate takes too much time and effort (heavy-weight) or it produces low-quality results (light-weight, guessing). In either case, the ROI doesn’t fit the circumstances. What is needed is a lightweight effort estimating process, focused on the individual, capable of constructing an estimate with a reasonable degree of accuracy.

1.5 Estimation landscape

The process of estimating the cost of future software development efforts, in terms of time and resources, is a complex issue. Researchers have designed and tested models for predicting effort using a variety of approaches, techniques, and technologies. For example, while researchers found that half of all published research into effort estimation (up through 2004) utilized some form of regression technique – predicting future effort based on past effort – a good deal of research was still going into other techniques such as function point analysis, expert judgment, and analogy [22]. These models are capable of predicting effort in a variety of environments with varying degrees of success and accuracy.

Most approaches share a common thread of complex mathematical models, requiring calibration and tuning. For example, COCOMO – one of the most well known models – uses 15 cost drivers (or “attributes”) to estimate the size of the product to be created [7]. Organizations must accurately rate each cost driver, as well as determine software project coefficients that characterize their environment. Function point analysis employs a model based on determining the number and complexity of files internal to the system, files external to the system, inputs, outputs, and queries. This allows an organization to estimate effort by comparing the number and type of function points to historical data from past projects.

Like COCOMO, function point analysis requires adjusting the estimate based on a variety of technical and operational characteristics, such as performance and reusability.

Despite the large amount of research focused on producing an accurate effort estimation, the typical software project is on time, on budget, and fully functional only about one-third of the time [37]. Clearly, the factors behind this statistic are poorly enumerated and vaguely understood, at best, given the number of variables involved. Despite the complexity of the problems facing the software cost estimating discipline, a wide variety of research into the problem has been conducted, focusing on such aspects as estimating approaches and models [22]. Perhaps one of the contributing factors lies at the bottom of the hierarchy, with software engineers who are ill equipped to provide estimates and validate deadlines imposed by their managers.

While there are many approaches to estimating effort, it is clear that the available approaches focus on the team or enterprise. Few approaches deal with individual effort estimation, such as at the task level. Furthermore, few approaches can be characterized as “light-weight” and suitable to the agile environments of one-person software development teams.

1.6 Estimation for the one-person team

In 1995, Humphrey introduced the Personal Software Process (PSP), which defined the first published software process tailored for the individual [17]. PSP defines a highly structured approach to measuring specific aspects of an individual’s software development work. With respect to effort estimation, PSP employs a proxy-based approach referred to as PROBE (Proxy-Based Estimating). In general, a proxy is a generic stand-in for something else, where the two object’s natures are similar. With regard to software estimating, a proxy is a completed task of similar size to an incomplete task. Therefore, it can be assumed that the time to complete the second task should be similar to that of the first. PROBE specifically uses a lengthy, mathematical process for determining anticipated software lines

of code (SLOC), which are used to compare tasks and find an appropriate proxy from which a time estimate is derived.

PSP has demonstrated popularity in both academic and business environments. Businesses, such as Motorola and Union Switch & Signal, Inc., have adopted PSP and claimed varying degrees of success. Many universities have integrated PSP into software engineering courses in an effort to demonstrate and measure the value of a structured personal development process. A University of Hawaii case explored some of the benefits and criticisms of PSP [20]. The study demonstrated that students using PSP developed a stronger appreciation for utilizing a structured process. However, the study also noted PSP's "crushing clerical overhead," which could result in significant data recording errors.

Interestingly, Humphrey's Team Software Process (TSP) [18] corroborates the fusing individual estimation efforts into a team-level estimate. Individual members of TSP teams practice PSP and employ individual PSP-gathered measures in making team-level estimates.

Researchers at Auburn University have developed a lightweight alternative to PSP, known as the Principle-Centered Software Engineering (PCSE) [40]. PCSE uses a proxy-based approach to estimating software lines of code, which is clearly lighter weight, yet still relies on non-trivial, mathematical models to produce a result. On one hand, the lightweight nature of PCSE calculations overcomes the heavy mathematical models of PROBE. On the other hand, the use of software lines of code limits the usefulness of PCSE in graphical or web-based development, which may include graphics, user interface widgets, etc.

The informal survey of ASP members reveals anecdotal evidence that software engineers typically opt for a "gut-feel" approach, where effort estimation is based on no more than educated guesses [5]. This impression is especially true of software engineers working outside the constraints of an organization, which typically mandates the use of formal tools and processes. Since developers typically underestimate 20-30% lower than actual effort [41], this would explain why the gut feel approach is typically heavily padded.

1.7 A “Better than Guessing” Agile Approach

A void exists in the spectrum of effort estimating tools, specifically focused on light weight tools for the individual. SISE – which stands for Sort-Identify-Size-Evaluate – represents a new approach to forecasting future software development effort by combining a standard regression model with relative task sizing. SISE introduces an agile approach to sizing by the individual in much the same way Planning Poker introduced agile sizing to the team.

Specifically, the SISE model guides the estimator through the process of organizing future tasks, not by matching them to historical analogies, but by size ranking, relative to historical tasks with known effort measurements. The SISE model then derives its results based on a simple principle: if the perceived size of a future task falls in between the actual size of two historical tasks, then the actual effort of the future task should also lie between the actual effort of the two historical tasks.

For example, assume two tasks have been completed on a project by a software developer: the first task in four hours and the second in six hours. A third task is then assigned to the developer, who estimates the relative size of the task to be somewhere between the first two tasks. It can be assumed, then, that the actual effort for the third task is somewhere between four and six hours. Note that this approach does not necessarily produce a single value estimate, but rather an upper and lower bound for the estimate (i.e. prediction interval).

If a single-value estimate is required, it can be extracted from the prediction interval in a number of ways. One approach to deriving a specific value for the estimate is to take the upper bound values. This produces a high confidence estimate, yet strongly resembles the practice of “padding” the estimate (i.e. playing it safe). Another approach involves simply averaging the upper and lower values and relying on the law of averages to even out the errors. In many instances, this approach may produce a specific value that is “good enough” for the circumstances with a minimal amount of effort and complexity. A third approach relies on a simple statistical analysis of the developer’s historical data to produce a weighting

factor which is applied to the upper and lower bounds to produce a single value. Simply put, the weighting factor represents where, on average, the developer's actual effort for all tasks fell between the upper and lower bounds of the estimates for those tasks. For example, a developer who, on average, completes tasks exactly at the midpoint between the upper and lower bounds of the estimate will possess a weighing factor of 0.5.

In its simplest form, the SISE model is specifically targeted at individual software developers, such as microISV operators, independent consultants, and team members. The approach's strength lies in the fact that individuals may develop reasonably accurate personal estimates based on their own historical data, while excluding team-level factors such as communication and overhead costs. Although the factors involved in a team-level effort estimate are more numerous, project-level estimates may also be derived with the assistance of the SISE model by combining individual team members' estimates and applying existing, proven approaches to adjusting for overhead. Another, more radical application of SISE might involve treating entire projects as tasks and deriving an estimate in the same manner as used by individual developers. This type of application would be most useful in certain organizations, which insist on estimates extremely early in the development cycle (i.e. before requirements are fully elaborated and understood).

Obviously, the SISE approach requires a calibration period, during which a historical data pool is constructed for deriving future estimates. Fortunately, many organizations and individuals already track effort expenditures, via time sheets or project management tools, providing a ready source for historical data. This leaves a historical gap for new developers and for environments lacking historical records. Fortunately, the calibration period can be relatively short, depending on the typical size variations in tasks; organizations with task sizes that vary widely will require a longer calibration period to derive a suitable data pool of historical values. Although not recommended, in the absence of historical data, it is possible to use another software engineer's actuals as a surrogate data pool and then rotate surrogate data out as actual developer data becomes available.

A variety of other factors exist which may affect the accuracy or precision of an estimate. These factors – such as programming environment changes, statistical outliers in the data set, data recording and accuracy, and granularity – must be addressed in a consistent manner when implementing the SISE model in an organization’s environment. The key to successfully applying this approach lies in the individual’s ability to recognize and adjust for these factors.

1.8 Conclusion

The emergence of software written by one-person teams – mobile device apps, services, components – renders inaccurate the portrayal of software engineering exclusively as a team activity. It brings a vanguard of exciting concepts in which the individual plays a pivotal role in not only creating viable software, but in controlling the development process. Systematic effort estimation, once the province of teams, has benefit to individuals as well; however, it, like so many other software methods, has to be stripped of unnecessary tasks if individual developers are to reap that benefit. It has to be intuitive, usable, and produce results that are more accurate than outright guessing.

1.9 Further Information

The SISE model is currently under development at Auburn University by IT veteran and graduate student Russell Thackston. The model is currently under evaluation by Auburn University’s Computer Science and Software Engineering program.

Chapter 2

Support for Individual Effort Estimation

2.1 Abstract

Estimation models play an important role in developing software both within teams and for individuals, as they validate deadlines, design project plans, and schedule work. These models rely on approaches such as regression and analogy to provide practitioners with the tools to derive useful estimates. The perception of the quality of these models is often defined by the accuracy of their estimates. However, many practitioners contend that the true value of an estimate lies in its usefulness as a project-planning tool, not in its ability to predict the future.

Despite the wide variety of existing estimation techniques available to teams, individuals few realistic choices in building estimates: PSP's PROxy-Based Estimates (PROBE); PCSE's Component-Based Estimating; and expert judgment (i.e. guesswork). The structures of PSP's and PCSE's estimation models arguably demonstrate a level of time commitment higher than guesswork. Anecdotal evidence suggests that a subset of practitioners choose guesswork as an alternative to established models due to the perceived overhead associated with using such a model. The space between guesswork and established models (such as PROBE and PCSE) represents a gap that can be filled with a lighter-weight, reasonably accurate model tailored to the individual and suited to estimation of individual tasks.

2.2 The Importance of Effort Estimation

The exact benefits of accurate software effort estimates vary from organization to organization. However, some common threads exist regardless of the environment. First, a

software estimate predicts the scale – from small to massive – of the work being undertaken. From this estimate of scale, predictions for staffing, computing resources, milestones, and deadlines are derived. Without a reasonably accurate estimate, many project management activities could not be conducted without resorting to guesswork. For example, an accurate estimate allows the project manager to predict milestone and delivery dates. These milestones assist in predicting project staffing or outsourcing needs, which may require interaction and scheduling with external entities. This leads into a proper project schedule and efficient resource management; without a reasonable estimate, resources may be brought in too early, resulting in slack time, or too late, resulting in delays. Risk management is also heavily impacted in the absence of a good estimate, since the purpose of an estimate is to enable efficient work management, not predict exact schedules [26].

Activities outside project management may also be affected by the lack of an effort estimate. For example, consulting firms who competitively bid on projects must rely on the estimate to properly estimate their time and resource investment and, based on that estimate, how much they will bill their clients. Furthermore, most clients will wish to establish and work to delivery dates based on the initial estimates. Inaccurate or absent estimates require a level of guesswork be employed and may result in cost overruns or excessively high, non-competitive bids for work.

The importance of effort estimates has led to the development of a wide variety of estimating approaches, tools, and techniques. Naturally, much of the research focuses on projects and teams; team projects present some of the most complex challenges in estimating as they attempt to address the myriad factors, such as overhead and communication. However, few approaches and tools attempt to address the challenge of estimates at the level of the individual.

2.3 Effort Estimation and the Individual

A wide variety of organizations exist for the purpose of creating software. Ultimately, these organizations employ one or more software engineers, regardless of team size or composition, for the purpose of building the product. In addition to the well-established role of “team member,” software engineers can also be found in roles such as independent consultants, where the lone consultant performs all the functions and duties of a small team. In the extreme, software engineers are also found operating as one-person software companies, known as micro Independent Software Vendors (microISVs).

Software engineers, in all their incarnations, are faced with deriving, validating, and being managed to estimates. For example, software engineers working on teams are regularly given assignments with attached deadlines. A proper analysis of the allotted time (estimate) allows the software engineer to plan his or her schedule accordingly, or provide useful feedback to his or her superiors in order to adjust the deadline. Independent consultants must competitively bid projects. An accurate estimate is essential to deriving a high quality bid; if the estimate is inaccurate or incomplete, the consultant may underbid the project and incur a loss or may overbid and fail to secure the contract. MicroISV owners rely on estimates in selecting where best to spend their limited time: development, marketing, support, etc.

Only two established methodologies, discussed later, present individual software engineers with reasonable alternatives to guesswork, an all too common alternative to established methods. The remaining challenge for consultants, team members, and microISVs is to implement these models into their own personal process in a non-invasive manner with minimal overhead and time investment.

2.4 Effort Estimating Tools, Techniques, and Approaches

Extensive research in the field of effort estimation has produced a wide variety of tools and techniques. These tools and techniques typically implement one or more underlying

approaches to solving the problem. To differentiate among tools, techniques, and approaches, consider Mountain Goat Software’s implementation of Planning Poker [29]. In Planning Poker, a group of experts discuss the requirements behind the tasks to be estimated. For each task, the individual estimators select a number from a Fibonacci sequence to represent the complexity of the task, though not necessarily in hours or days. All estimators simultaneously reveal their estimate to the other members of the group. If the estimates differ significantly, a discussion is held in which the outliers describe the reasoning behind their selections. The process then repeats until the estimates converge on a common value.

The Planning Poker model may be described in terms of the tool used to implement the model as well as the underlying technique and approach. To clarify, a *tool* is defined as “an instrument or apparatus” and a *technique* as “the manner in which technical details are treated” [27]. The tool used to implement Planning Poker is a simple deck of playing cards. The technique can be most succinctly described as a group of experts iteratively developing, presenting, and discussing individual estimates of effort for tasks, which ends as consensus is reached. At a more detailed level, the technique behind Planning Poker involves aspects such as informality, speed, and simplicity.

Planning Poker does not, in fact, implement a single *approach* to estimation; like most estimation models, Planning Poker combines multiple approaches: expert opinion, analogy, and work breakdown (disaggregation). These approaches may be used in different combinations to produce a variety of techniques. For example, the Wideband Delphi estimation model uses the same approaches as Planning Poker and the descriptions of the two models sound very similar. In contrast, the Wideband Delphi process involves formal meetings, paper forms, and anonymous estimates. Furthermore, the process takes days or weeks, as opposed to minutes or hours.

The specific tools used to implement a technique could vary based on the needs of the organization or its structure. For example, Planning Poker could be implemented for

geographically disperse teams using computers, voting software, and video chat, without sacrificing the core principles of the technique.

2.5 Common Estimation Approaches

There are a wide variety of approaches to address the problem of effort estimation [22]. Table 2.1 lists the some common approaches with a brief description. This list is not intended to be comprehensive; rather, it is designed to present a general cross-section of some widely used approaches.

Regression, also known as regression analysis, involves modeling and analyzing multiple variables that are assumed to be interdependent. In software effort estimation, regression typically involves “sizing” a project in terms of some known or estimated quantity – features, inputs/outputs, or lines of code – then calculating the effort from that quantity. For example, COCOMO is a series of models designed to compute software development effort as a function of size in estimated software lines of code (SLOC) [7] [8]. The SLOC is translated into effort based on industry data and, depending on the COCOMO model employed, the project type, cost drivers, or phase.

Another common approach to estimating effort involves drawing analogies. Analogy-based reasoning involves drawing conclusions about a future occurrence based on the details of a similar, past occurrence. Estimating software effort by analogy involves four factors, which directly influence the accuracy of the estimate: the availability of an appropriate analogue; the soundness of the selection strategy; how the differences between the analogue and target are considered and adjusted for; and the accuracy of the available data points [42].

A third approach involves the expert judgment of one or more members of the project team. Simply put, expert judgment involves relying on an individual, or group of individuals, to gather, evaluate, discuss, and analyze data concerning a target project [21]. Instead of inputting the data into a formal analytical model and publishing the result, the estimators

Approach	Description
Analogy	Drawing conclusions about a future occurrence based on the details of a similar, past occurrence.
Artificial Neural Networks	Application of massively parallel, computer-simulated, biological neurological systems to predict outputs through the use of complex dependent and independent input variables.
Classification and Regression Trees	Building a binary tree with branches representing possible effort values for each estimation characteristic, then locating the "optimal" sub-tree. Traversing the sub-tree from terminal node to root allows for the calculation of an effort estimate.
Expert Judgment	Relying on an individual, or group of individuals, to gather, evaluate, discuss, and analyze data concerning a target project to build an estimate.
Function Point	The use of system inputs, outputs, and persistent data as a measure of the amount of functionality required by a system. The functionality is expressed as "function points," which can be used to derive effort.
Mathematical Models	A mathematical formula that predicts effort (output) based on the multiple inputs, such as team productivity and project scale.
Proxy-Based	Using a known or predicted unit of size (screens, lines of code, etc.) for a task to infer required future effort for the task.
Regression	Modeling and analyzing multiple variables that are assumed to be interdependent. For example, predicting task's duration based on the estimated lines of code.
Simulation	A computer model that attempts to simulate the abstract model of the required work effort for a set of activities.
Work Breakdown	Decomposition of an effort into individual tasks. Also known as "disaggregation."

Table 2.1: Common approaches to estimation.

produce an estimate based on their knowledge of the work to be performed and the environment. The estimators may follow a checklist or set of guidelines, but no mathematical model is employed to derive the final numbers. Some experts characterize expert judgment as a “gut-feel” alternative to established models. However, when viewed as an approach, expert judgment can be characterized as one aspect of a larger tool or technique. In fact, aspects of expert judgment find their way into most tools and techniques, simply because their input process is managed by individuals making decisions about how to divide, structure, and organize work.

Work breakdown, as an approach, is fundamental to many estimation tools and techniques, as well as project management in general. In simple terms, a work breakdown is a decomposition of an effort into individual tasks. The level of decomposition required is defined by many factors, such as the overall size of the project; the size and structure of the project team; and the type of project. One major benefit of creating a thorough work breakdown is that it helps reveal all the individual tasks involved in the effort, reducing the chance of leaving out small, but important steps [26]. The work breakdown also serves as a tool for comparing a future project to past projects that have been similarly decomposed into smaller tasks. The disadvantage of the work breakdown approach is that it requires a more complete knowledge of the system to be built at the time the estimate is derived.

Another approach, known as function point analysis, involves counting the number and complexity of functions performed by a software product [2]; such functions include files operations internal to the system, files operations external to the system, inputs, outputs, and queries. The number of function points may be translated into an estimate of SLOC, which in turn may be used to estimate effort in terms of time commitment, based on the historical productivity of the project team.

Proxy-based estimation approaches involve identifying and counting known features of a task or effort – such as the number of screens, lines of code, number of functions/procedures,

etc. – then inferring future effort based on those “proxies.” For example, a high level architectural design may describe the number of components that must be created to complete the software product. By combining the number of components, their individual estimated complexity, and historical data on productivity, an estimator can infer the amount of required effort to build the new components.

Artificial neural networks (ANN) are software models inspired by the architecture of biological neural networks [43]. ANNs represent a novel approach to estimation by utilizing a massively parallel network of interconnected nodes (representing virtual, biological neurons), each with a series of inputs, and each generating an output when the sum of the inputs exceeds a predefined threshold. Neural networks with a feedback – or learning – mechanism improve performance by fine-tuning the weighting of the inputs and/or the threshold for producing the output. ANNs are particularly suited to modeling software estimates when a non-linear relationship exists between the inputs (e.g. size) and outputs (e.g. effort) [13].

Estimation approaches based on mathematical models are also common. Researchers have attempted to derive mathematical models and formulas to represent the relationship between size and effort. The simplest model of this relationship is a linear relationship: as size increases, effort also increases at a steady rate. Linear models, however, are not suitable for estimating non-trivial projects in large and complex environments; therefore, more complex models were developed, such as Putnam’s model, which is based on a Rayleigh distribution [31]. These models, in their various forms, attempt to compensate for factors such as increased overhead and communications as the size of the project and/or team increases.

In addition to the aforementioned approaches, researchers and practitioners have developed a wide variety of approaches that are best described as “academic exercises” intended to explore novel theories or highly specialized situations [22]. While these additional approaches add significant value to the discipline, they are highly specialized and are beyond the scope of this analysis.

2.6 Common Estimation Tools and Techniques

The tools and techniques derived from various approaches may be categorized in several ways. For example, some tools are highly structured, requiring the practitioner to follow specific steps in a prescribed order. Some tools use a strict mathematical formula to calculate their output, while others rely more on expert judgment or human reasoning.

COCOMO is a well-structured, formal approach to developing software estimates [7]. COCOMO, in its original form (COCOMO 81), addressed the software development practices of the day, such as mainframe overnight batch processing. In 2000, COCOMO II was published in a revised form to reflect recent changes, such as software reuse and off-the-shelf software components [8]. Four major steps comprise the foundation of COCOMO. In the first step, the nominal effort is determined based on estimating the number of “source instructions,” or lines of code. Next, a series of fifteen cost drivers – relating to the product, environment, and hardware – are evaluated and each is assigned a weight or value. In the third step, the product of these “effort multipliers” is used to derive an effort value – usually in man-months – from the nominal effort from step 1. Lastly, the estimator adjusts for additional factors, beyond the first three steps. Boehm’s stated goal in developing the COCOMO model is to “help people understand the cost consequences of the decisions they will make in commissioning, developing, and supporting a software product” [7].

The Putnam Model is another formal technique based on historical data and mathematical analysis [31]. The Software Lifecycle Model (SLIM) is the proprietary tool released by QSM, Inc., a company founded by Putnam. The Putnam model follows the idea that historical project data such as time, effort, and size can be mapped to a consistent distribution of data, or curve on a graph. Therefore, an effort estimate, usually in man-months, may be derived for a future project by fitting it to the curve based on the projects estimated size (e.g. lines of code, etc.).

The term function point analysis refers to both an approach (described earlier) and a technique. Function point analysis is a highly structured technique that begins with a

thorough review of project requirements to uncover a comprehensive list of software “function points.” These function points are comprised of a list of inputs, inquiries, outputs, internal files, and external interfaces. The function points are counted, adjusted for factors such as complexity, and summed. The resulting value is a dimensionless number representing the relative measure of the number of functions defined by the requirements. By comparing this number to past projects and their corresponding values, an estimate of required effort can be proposed. Various tools exist – such as the Construx Estimate tool from Construx Software Builders [10] and ESTIMACS (known as CA-Estimacs) from Computer Associates [25] – to assist in deriving software estimates based on function points.

PROxy-Based Estimating (PROBE) – a component of the Personal Software Process (PSP) – is both a technique and a tool for deriving estimates [17]. As a technique, PROBE draws on several approaches, including proxy-based estimation, regression, and analogy. Like the preceding techniques, the PROBE tool is a well-defined model, however, its estimates are derived through the use of proxies or objects that are used to estimate the size of the software product. Each object is assigned a type – which loosely defines the relative complexity – and an estimated number of methods. The combination of type and method count defines the estimated size of the object. Based on the proxy list and historical data – past estimates and actuals – the overall size of the project is estimated.

PROBE is not the only proxy-based technique for deriving estimates. The Principle Centered Software Engineering (PCSE) process also derives estimates through the use of proxy-based estimating. While both PROBE and the PCSE estimation approach are proxy-based, the techniques differ significantly in terms of complexity and number of steps. PCSE’s technique derives an estimate of effort through the estimation of the number of “parts” required to build the software, which are translated into an estimate of software lines of code. The required effort is then inferred from the combination of lines of code and historical productivity.

Case-based reasoning, a superset of analogy-based reasoning, involves constructing a model of a problem, retrieving an appropriate analogue, transferring the solution used for the analogue to the target instance, mapping attributes between the analogue and the target, then adjusting the estimate to account for attribute differences. Estor is one example a software product, which combines the concepts of function points and case-based reasoning to derive estimates [42].

Analogy Software Tool (ANGEL) is another software product, which applies analogy-based reasoning to developing estimates [36]. Unlike Estor, which uses function points, ANGEL allows the estimator to define and input variables to describe the features of the project to be estimated. From the user-defined variables, a subset is selected and ANGEL locates the “closest” match between the target project and historical projects based on the Euclidean distance between the variable sets.

Web Objects is an approach designed to address the many disparate elements that make up web-based applications, such as static and dynamic pages, JavaScript, cascading style sheets, images, etc [33]. Web Objects calculates size using the language independent Halstead’s equation [15], which focuses on the *operands* and *operators* involved in the proposed application to determine the “volume of work” from which an estimate of effort is obtained. Within the Web Objects context, an operand is something done to an operator. The Web Objects approach accounts for complexity by weighting the operators/operands as low, medium, or high.

The Wideband Delphi method presents a formal approach to deriving estimates based on expert judgment [7]. In this approach, a group of experts follow a structured series of steps, managed by a coordinator, to reach a consensus on an estimate for a project or series of tasks. First, each expert is presented with the specification and forms to share his or her estimates. The experts meet, discuss the issues involved in completing the work, and then anonymously provide their estimates using the provided forms. If the estimates vary significantly, the coordinator calls another meeting in which the issues are further discussed.

The approach repeats the discussion/estimation steps until the estimates converge, forming the basis for the final estimate.

Planning Poker is an agile approach to estimation, similar to Wideband Delphi, which involves a group of experts iteratively involved in discussion and estimate presentation, in an attempt to reach consensus [29]. Unlike Wideband Delphi, however, no forms are used and the discussion is not anonymous. Furthermore, the process occurs in a single meeting, not over a period of days. In addition, the estimates are presented in terms of complexity, such as hours, days, or story points; a single number chosen from a deck of cards, usually made up of the values from a Fibonacci sequence, represents the expert’s estimate of complexity. As with the Wideband Delphi approach, significant differences in estimates are discussed; however, unlike the Wideband Delphi method, the discussion is immediate and public. Once a consensus is reached, the results form the basis of the full estimate.

Table 2.2 lists the previously discussed tools and techniques, cross-referenced with the underlying approaches demonstrated by the technique. Note that expert judgment plays some role in all techniques/tools, since the inputs to the processes are constructed or guided by a human, at some point. In addition, most techniques require some form of work breakdown to occur prior to beginning the estimation process.

2.7 Measuring Estimation Quality

Despite the large number of estimation approaches and techniques available, the software engineering discipline does not formally recognize a single “standard” quantitative measurement for the quality of a software effort estimation model. However, most research assumes the quality of a model is directly related to the model’s ability to accurately predict future effort. This accuracy, or lack thereof, may be measured in a variety of ways.

Most quality measurements begin by comparing the estimated effort to the actual effort. For example, the estimated effort for a task may be ten hours less than the actual effort. However, the value of “ten hours” has little meaning outside the context of the project.

Technique/Tool	Approach						
	Analogy/Case-Based	Expert Judgment	Function Points	Mathematical Models	Proxy-Based	Regression	Work Breakdown
ANGEL	•	•		•		•	•
CLAIR		•		•		•	•
COCOMO/COCOMO II		•		•		•	•
ESTIMACS		•	•	•		•	•
Estor	•	•	•			•	•
Expert Judgment		•					
Function Point Analysis		•	•		•	•	•
Planning Poker	•	•				•	•
PROBE		•			•	•	•
PCSE	•	•			•	•	•
Putnam Model/SLIM		•		•		•	•
Web Objects		•					•
Wideband Delphi	•	•				•	•

Table 2.2: Common approaches to estimation.

Ten hours could be a significant error in the context of a two hours task; on the other hand, ten hours would be virtually insignificant in the context of a five person-year project. Therefore, the relative error (RE) is used as a foundation for measuring estimation accuracy (see Equation 2.1). For example, if a 100-hour project is underestimated by ten hours, the relative error is 0.1 (10%), or ten divided by 100. This may produce either a positive number (underestimation) or a negative number (overestimation). In many cases, the magnitude of relative error (MRE) is used to eliminate negative numbers and simplify calculations (see Equation 2.2).

$$RE = \frac{(actual - estimate)}{actual} \quad (2.1)$$

$$MRE = \left| \frac{(actual - estimate)}{actual} \right| \quad (2.2)$$

The RE and MRE values are useful for determining the accuracy of a single estimate. However, a single large-scale estimate is typically composed of multiple smaller estimates; in such cases, the Mean Absolute Relative Error (MARE) is used [11]. The MARE value (see Equation 2.3) may be referred to as the Mean Magnitude of Relative Error (MMRE) or the average magnitude of relative error. Variations of MARE and MMRE also include the Median MRE.

$$MARE = \frac{1}{n} \sum_{i=1}^n \left| \frac{(actual_i - estimate_i)}{actual_i} \right| \quad (2.3)$$

where n is the number of individual tasks in the set.

The MARE produced by a particular model is sensitive to the environment in which the model is applied. Certain models have demonstrated widely varying MARE values when applied to different projects in different organizations with different teams. Furthermore, the calibration process appears to be one of the most significant influencing factors. For example, one study found MARE values for all tested models varying from an average of

Model	MARE	Notes
<i>Jeffery and Low, 1990 [19]</i>		
CLAIR	79%	Average of three organizations with values ranging from 43% to 117%.
Function Points	108%	Average of six organizations with values ranging from 39% to 132%
<i>Kemerer, 1987 [25]</i>		
SLIM	772%	Overestimated in all cases.
COCOMO	601%	Overestimated in all cases.
Function Points (FP)	102%	Function points to person months
Function Points (SLOC)	167%	SLOC to person months
Function Points (FP to SLOC)	38%	Large negative bias.
ESTIMACS	85%	Low (92%) confidence level.
<i>Mukhopadhyay et al., 1992 [30]</i>		
Expert Judgment	31%	-
Estor	53%	-
Function Points	103%	-
COCOMO	619%	-
<i>Ruhe et al., 2003 [32]</i>		
Function Points	33%	-
Web Objects	24%	-
Expert Opinion	37%	-
<i>Schoedel, 2006 [34]</i>		
PROBE	14%	Limited study of one student over 10 SQL programs with values ranging from 1% to 67%.
<i>Yenduri et al., 2007 [44]</i>		
Expert Judgment	59%	14 projects
COCOMO II	35%	49 projects

Table 2.3: Sample MARE values.

57 percent to almost 600 percent [30]. Table 2.3 lists several MARE values from various researchers and studies.

Practitioners should note that the MARE value represents the difference between an estimate of effort for a task and the actual effort expended to complete the task. A MARE value cannot reflect the complex environmental, architectural, and social factors that inevitably influence the time spent by an individual or team working to complete a task or project [24]. This is why many estimates should be given in the form of a low and high value, as opposed to a single value; these low and high values represent the reasonable best and worst cases. Furthermore, an estimate – in its truest form – is designed to facilitate project management controls, not predict the future. Therefore, while an “accurate” estimate is

desirable, an “inaccurate” estimate does not necessarily indicate a failure of the estimation model.

Whereas the MRE and MARE values represent the “accuracy” of an estimate, practitioners must also take into consideration a variety of other factors, such as confidence interval, prediction interval, and estimation bias. The confidence interval indicates the estimator’s confidence that the actual effort will fall within the range of the estimate, assuming the estimate is composed of a low and high value, such a “90 to 110 person hours.” A confidence interval of 90% or higher is recommended for project planning models [28]. Therefore, an estimator’s goal is to produce an estimate in which he or she has a high level of confidence in its accuracy.

The prediction interval (PI) represents the low and high bounds of the estimate. For example, an estimate of “90 to 110 person hours” for an activity has a prediction interval of [90,110]. A natural correlation exists between the confidence interval of an estimate and the prediction interval: as the prediction interval expands, the confidence in the estimate should increase. For example, a prediction interval of “between 10 and 1,000 person hours” is more likely to be correct than a prediction interval of “between 99 and 101 person hours.” Therefore, an estimate should include both the prediction interval and the confidence interval.

The quality of an estimation model may be evaluated over a period of time by analyzing the hit rate, width-accuracy correlation, and estimation bias. The hit rate is defined as the percentage of time the actual effort falls within the prediction interval [23]. Equation 2.4 shows the formula for calculating the hit rate. This value can be compared to the confidence interval; a hit rate that is lower than the confidence interval indicates overconfidence in the estimate and vice versa.

$$HitRate = \frac{1}{n} \sum_i h_i, h_i = \begin{cases} 1, \min_i \leq actual_i \leq \max_i \\ 0, actual_i > \max_i \vee actual_i < \min_i \end{cases} \quad (2.4)$$

where \min_i and \max_i are the minimum and maximum values, respectively, of the PI for the effort estimate of task i ; $actual_i$ is the actual effort of the task i ; and n is the number of the estimated task.

The width-accuracy correlation is a measurement of quality that attempts to determine if the estimates produced by the model are the result of informed analysis or wild guessing. It can be assumed that informed analysis will produce estimates with a high correlation between accuracy of the most likely effort and PI width. In other words, narrow PI widths should accompany accurate estimates, and wider PI widths should accompany inaccurate estimates. Wild guessing would, theoretically, produce a low correlation between the PI width and the estimate accuracy.

Equation 2.5 shows the formula for calculating the balanced relative error (BRE), which is used as the foundation for the width-accuracy correlation, due to the fact that the BRE allows for more realistic modeling of linear relationships, such the width-accuracy correlation.

$$BRE_i = \frac{actual_i - estimate_i}{\min(actual_i, estimate_i)} \quad (2.5)$$

where $\min(actual_i, estimate_i)$ is the lowest value for $actual_i$ and $estimate_i$.

Lastly, a set of estimates can be used to determine if estimation bias exists (i.e. a tendency toward the low or high ends of the PI). The Actual Effort Relative to PI (ARPI) value can be a good indicator of estimation bias. ARPI is a measure of the distance between the actual effort and the midpoint of the PI, normalized by the width of the PI. Equation 2.6 demonstrates the ARPI calculation.

$$ARPI_i = \frac{actual_i - PI_midpoint}{max_i - min_i} \quad (2.6)$$

where $PI_midpoint = (Max_i + Min_i) / 2$

In summary, the quality of an effort estimation model's output must be measured against a variety of factors. First, the model's accuracy – ability to predict future effort within an acceptable margin of error – must be measured and evaluated (i.e. relative error and magnitude of relative error). Next, the model's output for a set of activities must be considered (i.e. mean absolute relative error). Lastly, the attributes of an estimate should be considered to address factors such as confidence, quality over time, and bias.

2.8 Estimation Techniques for the Individual

Despite the wide variety of existing research and products available to teams, individual estimators possess few realistic options for deriving estimates for single tasks. Many of the aforementioned techniques and approaches require a significant investment of time and resources, which are not realistic for an individual to perform. An individual interested in deriving personal estimates at the task level has a few limited options.

The most obvious and least time-consuming option for deriving estimates is expert opinion, or guesswork. Naturally, the quality of estimates produced in this manner varies widely with the experience of the estimator, the type of work, and the environment. In fact, research suggests that subjective estimates are most accurate when derived from groups of estimators, as opposed to lone individuals [16].

In opposition to guesswork are the established models based on many of the aforementioned techniques and approaches. These established models require a significant time investment in the form of organizational historical data, carefully derived multipliers, and/or mathematical models. The required investment makes these models impractical for an individual estimator to implement, leaving a handful of models constructed specifically for the

individual. PSP's PROBE and the PCSE estimation approach are two such models designed specifically for individuals and small task set estimation.

The PROBE tool is characterized by a highly structured approach to software development, involving careful recordkeeping and data analysis. The fourteen-step estimation method and the five-step estimation script (with an accompanying one-page worksheet) lead estimators through the process of planning, preparing, and implementing a proxy-based estimate [17]. Although this highly structured approach makes it a good fit for PSP advocates, some practitioners see the “crushing clerical overhead” as too time consuming for certain environments [20].

The PCSE process is currently under development at Auburn University. One goal of the PCSE software process is to remain as lightweight as possible while providing the most benefit. The use of a proxy-based estimation technique in PCSE – which requires a moderate level of recordkeeping and a relatively simple regression formula – fits nicely between guesswork and PROBE.

The agile development movement tends to avoid complex and highly structured software processes and relies on simpler techniques, such as expert judgment or expert opinion. For example, Scrum endorses an agile, expert opinion-based approach to estimating effort, such as Planning Poker, which involves the entire team [35]. The key feature of Scrum estimates are the use of abstract values, such as Fibonacci numbers or story points to represent the relative size of a task, rather than actual size in hours, days, or weeks.

Taking into consideration the lightest weight approaches to effort estimation in terms of time spent preparing to derive an estimate, team-oriented models, such as COCOMO and SLIM, are impractical for an individual to set up, tune, and operate for single tasks. Guesswork/expert opinion requires the least commitment, as there is no mandatory recordkeeping requirement. PROBE and PCSE require the practitioner to maintain a detailed record of work activities; however, no additional analysis – such as deriving cost drivers – is required to

begin preparation of the estimate. In terms of *pre-estimation time* investment, a gap exists between guesswork and PROBE/PCSE.

With regard to time spent deriving an estimate, the alternatives – guesswork, PROBE, and PCSE – represent varying investments. Guesswork, arguably, requires the least time investment, as the estimate is derived as a result of careful consideration on the part of the estimator; no formal standards exist as to what steps should be employed to derive a value. PROBE and PCSE require a series of forms to be completed along with a set of non-trivial calculations. Arguably, PCSE’s forms and calculations are less in-depth and complex than PROBE, however, both instances require larger time investments than guesswork. Again, a gap exists between guesswork and PROBE/PCSE, this time in terms of *time spent calculating an estimate*.

As previously mentioned, a study of PSP’s PROBE model demonstrated a MARE value of 14%. On the other hand, studies of expert opinion-based estimation demonstrate MARE values ranging from 31% to 59%. Although the scope of the PROBE study was extremely limited, it is logical to assume that a structured estimation model would, on average, produce more accurate results than unstructured guesswork. This implies a third aspect to the gap between guesswork and established estimation models: *accuracy*. Arguably, these three aspects – preparation time, execution time, and accuracy – define an exploitable space between the simplest estimation approach (guesswork) and the lightest-weight established models (PSP/PROBE and PCSE).

2.9 Summary and Conclusions

Effort estimation plays an important role in software development. A broad spectrum of approaches, tools, and techniques exist to support the estimation process. The quality of the models built upon these techniques may be measured by their accuracy, consistency, and specificity. In fact, most models present organizations with an opportunity for improved project and risk management, regardless of their specific ability to predict the future.

A vast majority of these tools and techniques focus on teams or projects; few approaches address the needs of the individual software engineer or microISV. The exceptions are PSP and PCSE, the only two processes specifically designed for individuals. However, the time investment in both PSP and PCSE are non-trivial. The space between guesswork and these two estimating approaches is wide enough to accommodate a new model, targeted at individuals interested in an estimation approach that outperforms guesswork without the overhead of the estimation models currently available.

Ideally, an estimating approach tailored to the individual – such as a consultant, team member, or microISV owner – would have the features of simplicity, speed, and relative accuracy. Such attributes would naturally increase the likelihood of adoption and continued usage. While PSP represents a model designed specifically for the individual and PCSE represents a highly streamlined model, neither fully bridges the gap between established models and guessing.

Chapter 3

SISE: A Novel Approach to Individual Effort Estimation

3.1 Abstract

Individuals rely on their personal processes to develop software in a systematic and structured manner. Time management, which relies on relatively accurate effort estimation techniques, has been shown to be a key component in planning and executing software development activities. Despite a plethora of research into team-based effort estimation models, few models are suitable for use by individual software engineers. Models tailored to the individual include guesswork, an approach commonly used in industry; the PCSE model, under development at Auburn University; and the PROBE model, the only peer-reviewed model devoted to lone software engineers. This spectrum of choices features a gap between guesswork and more formal models, which could be filled with a lightweight, agile, and reasonably accurate alternative. The SISE model combines expert judgment – in the form of relative sizing decisions – with empirical, historical data to create such an alternative. The SISE model is based on a four-step process in which historical tasks are sorted by actual effort values; a future activity’s effort is forecasted, relative to the historical tasks’ requirements; and a prediction interval is constructed for the future activity.

3.2 Introduction

In recent years, dramatic changes to the software industry have brought individual developers to the forefront of software engineering practices. In addition, the rise of the software micropreneur in markets such as mobile app development and web applications has reinforced the need for lightweight, agile software engineering practices. For example,

recent surveys of the microISV industry have shown that time management and related issues topped their founders' list of "pain points" [39]. Historically, however, many of the software process tools available to software engineers have been team-oriented, making them impractical for the individual to benefit from their usage [38].

In response, researchers at Auburn University have been focusing their efforts on constructing tools targeted directly at the individual software engineer. One such tool is the SISE effort estimation model.

SISE is a lightweight, agile model designed to construct estimates based on expert knowledge and empirical evidence. In this respect, SISE outperforms simple guesswork, while incurring a much lower overhead than traditional, established models, which rely on complex software, algorithms, or mathematical calculations. The SISE Model

"SISE" is an acronym for the model's four-step process. The four steps, in order, are Sort, Identify, Size, and Evaluate. The first step – Sort – involves the ordering of historical data by the actual effort required to complete the activity. The second step – Identify – involves choosing two tasks from the historical data set: one confidently known to be smaller, one confidently known to be larger, and both relatively close in size to the future work. Using this pair of tasks, the estimator begins the third step – Size – by producing a rough prediction interval of the future activity's size using the actual effort values for the two completed tasks. The final step – Evaluate – involves shifting or resizing the prediction interval to account for any historical bias. This last step is optional and is only applied if the estimator is dissatisfied with the precision, accuracy, or confidence level of his or her estimate.

The design of the SISE model focuses specifically on the individual software engineer. Its estimates are based solidly on empirical data gathered by the software engineer and only applicable to that person. Personal skills and experiences are too numerous to list, quantify, and apply to every estimation scenario. Therefore, the SISE model seeks to join empirical

data to the process of expert judgment. This results in a model that must be individually calibrated by each software engineer using his or her own personal data.

3.3 The SISE Steps

3.3.1 Step 1: Sort

The Sort step involves the ordering of historical data by the actual effort required to complete the activity. The simplest approach is to maintain an electronic record of historical data, such as a spreadsheet or database. The data is then sorted by the actual effort, from smallest to largest. Next, the numeric values associated with each historical data point – estimated effort, actual effort, etc. – are hidden, leaving only the text description of the completed tasks; this prevents the software engineer from selecting tasks based on a desired numeric outcome, such as “eight hours.”

3.3.2 Step 2: Identify

Next, the description of the future activity is compared to the descriptions of the historical tasks. Two historical tasks must be located: one confidently smaller and one confidently larger than the future activity. The smaller task should be one which the estimator is confident is smaller than the future activity, but is as close as possible in size to the future activity. The larger task should be the inverse: larger in size, but still relatively close. Since the historical data set is already sorted, a very efficient way of locating these two tasks is through the use of a binary search algorithm.

3.3.3 Step 3: Size

Once the practitioner has chosen a pair of tasks, the third step – Size – produces a rough prediction interval of the future activity’s size. The size of the future activity is inferred by looking at the actual effort values of the two historical tasks. For example, assume the historical record contains twenty completed tasks and the estimator has selected tasks 9 and

14 as the two tasks confidently believed to be smaller and larger, respectively, than the future activity. The rough size of the future activity can, therefore, be inferred to fall between the actual sizes of tasks 9 and 14.

Prediction intervals are expressed using a low estimate and a high estimate, with the actual value expected to fall somewhere in between. Prediction intervals are expressed using the notation “[low, high].” For example, the prediction interval [5, 7] means we expect the actual value to fall somewhere between five and seven hours, inclusive [23].

The actual effort values associated with the bracketing tasks represent the low bound and high bound of a prediction interval. However, this interval is a rough estimate of the expected effort and may need to be refined.

3.3.4 Step 4: Evaluate

The final step – Evaluate – is optional and may be applied in the event the estimator is dissatisfied with the precision, accuracy, or confidence level of the estimate. The estimator may choose to shift the prediction interval based on an analysis of his or her historical bias. This involves analyzing the practitioner’s track record with using SISE and implies a prerequisite: the practitioner has been using SISE or some other prediction interval-based estimation approach and has an idea of his or her historical accuracy. This historical bias is then used to modify the rough estimate to produce a specific estimate. For more details on how to shift a prediction interval based on historical bias, refer to the sidebars *Removing Shift Bias* and *Removing Width Bias*.

It should be noted that within the SISE model estimation bias is not an indication or measure of error committed by an estimator. Rather, it is a measure of how the best efforts of the estimator translate through the SISE model to create an estimate that mirrors actual effort.

Removing Shift Bias

Shift bias involves a prediction interval that is too low or too high and may be corrected by shifting the interval. Shift bias exists only if the historical actuals fall predominately below or above the associated prediction intervals; estimation error that is spread equally between overestimates and underestimates is a width bias and must be corrected in a different manner.

To determine if a shift bias exists, a form of simulation must be conducted. The simulation involves (1) compiling a list of the historical estimation error values, (2) shifting all the historical prediction intervals by each error value, then (3) checking the change in overall hit rate with each shift.

Consider, for example, the following historical data:

Activity	Prediction Interval	Actual	Error
Task 1	10-15 hours	16 hours	1
Task 2	12-16 hours	18 hours	2
Task 3	2-5 hours	5 hours	0
Task 4	1-3 hours	3 hours	0

The hit rate for the unmodified data set is 50%. All the prediction intervals could be shifted by 1 hour, which would cause Task 1 to become a successful prediction. Additionally, all the prediction intervals could be shifted by 2 hours, which would cause Task 2 to become successful. But how would these shifts affect the other predictions?

If all the prediction intervals are shifted by 1 hour, the hit rate rises to 75%; task 1's prediction interval now contains the actual effort and Tasks 3 and 4 are still successful. If the intervals are shifted by 2 hours, the hit rate rises to 100%. So, given this limited data set, shifting future estimate's prediction intervals by 2 hours may produce more accurate results.

Removing Width Bias

Once shift bias has been accounted for, the estimator may wish to either improve their precision or confidence level. This action involves a trade-off since increasing one reduces the other. For example, if the estimator wishes to increase their confidence level, the prediction intervals must be widened, making the estimates less precise. If the estimator wished to increase the precision of their estimates, by reducing the size of the prediction interval, the confidence level in the estimate will be proportionally reduced.

Improving the confidence level is accomplished by symmetrically widening all past prediction intervals by whatever amount is necessary to reach a hit rate equal to the desired confidence level. For example, if the historical record demonstrates a hit rate of 50% and the estimator would like to reach a confidence level of 80%, then all the past estimates' prediction intervals are widened until 80% of the actuals fall within the associates prediction intervals.

The inverse operation may be performed to improve the precision of the estimates. Past prediction intervals may be symmetrically reduced in size until the desired prediction interval width is reached. The new (and reduced) confidence level may then be calculated by checking the hit rate for the entire historical record.

Here's an example of how widening the prediction interval may allow for an increase in the hit rate from 60% to 80%.

Activity	Original Prediction Interval	Actual	New Prediction Interval
Task 1	10-15 hours	10 hours	9-16 hours
Task 2	12-16 hours	16 hours	11-17 hours
Task 3	5-7 hours	6 hours	4-8 hours
Task 4	9-11 hours	12 hours	8-12 hours
Task 5	13-15 hours	11 hours	12-16 hours

Shifting the prediction intervals would not have improved the hit rate; however, if all the prediction intervals are increased by two hours (-1 to the low and +1 to the high), the hit rate moves from 60% to 80%.

3.4 Getting Started with SISE

Introducing the SISE model into an individual's software process is simple. As with all regression-based approaches, the first step is to begin tracking effort expended to complete the work activities. As each new task is completed, it is recorded in the historical record with its description, estimated effort, actual effort, etc. This historical record will be the basis for all future estimates. If an estimator has already been tracking his or her time, then this information may be used, as long as it matches the granularity of the future activities to be estimated.

The software engineer produces a SISE estimate by reviewing his or her historical record. The historical record is sorted from smallest to largest by actual effort and the numeric values are hidden from view (step 1). The engineer reviews the list looking for a task that he or she is confident is smaller than the future activity. If a task is located (step 2), the actual effort is revealed and that value is recorded as the low end of the future task's prediction interval (step 3). If the estimator is not confident that any historical task is smaller than the future activity, then a value of zero is recorded as the low end of the future task's prediction interval.

Next, the estimator reviews the list a second time to locate a confidently larger task, again using only the descriptions of the future and historical tasks. If one is located, the actual effort value is revealed and recorded as the high bound of the future task's prediction interval. If a larger task cannot be confidently identified, then the upper bound of the future activity's prediction interval is recorded as "unknown" using the sign for infinity (∞).

With the prediction interval for the future activity established, the software engineer proceeds with work on the activity. Once the activity is completed, the actual effort is recorded in the historical record and the process repeats.

Measuring Accuracy

The accuracy of a single value estimate is determined by the magnitude of the estimate's error, relative to the actual effort. For example, if an activity is estimated to take 4 hours, but actually takes 5, the magnitude of relative error (MRE) is 0.2 (or 20%). Here is the formula:

$$MRE = (actual-estimate)/actual$$

When using prediction intervals to describe an effort estimate, the practitioner's accuracy is determined by the number of activities with actual effort values that fall within the predicted interval. Here's the formula:

$$Hit\ Rate = No.\ hits / No.\ estimates$$

For example, consider the following list of work activities.

Activity	Prediction Interval	Actual
Task 1	10-15 hours	12 hours
Task 2	12-16 hours	15 hours
Task 3	2-5 hours	5 hours
Task 4	1-2 hours	3 hours
Task 5	16-22 hours	15 hours
Task 6	9-13 hours	12 hours
Task 7	4-6 hours	5 hours
Task 8	6-8 hours	8 hours
Task 9	3-4 hours	4 hours
Task 10	6-10 hours	9 hours

Eight of the ten activities were completed within the time frame defined by the prediction interval; Tasks 4 and 5 took more and less time, respectively, than predicted. Therefore, the hit rate for this sample is 0.8, or 80%.

3.5 Accuracy, Precision, and Confidence Level

By using a prediction interval as the basis for estimates, the SISE model presents the software engineer with a competing set of factors: accuracy, precision, and confidence.

The accuracy of an estimate is measured in different ways depending on the type of estimate. Many project managers and project management applications expect an effort estimate to be phrased as a single value. Single value estimates are easy to understand, simple to aggregate, and are expected to be wrong. After all, what is the probability that an activity estimated at 10 hours will take exactly 600.00 minutes? Therefore, the accuracy of a single value estimate is measured in terms of its error (see sidebar titled *Measuring Accuracy*).

The accuracy of a prediction interval, on the other hand, is measured by how often the actual effort falls within the interval. The overall percentage of actual effort values falling within their prediction intervals is known as the hit rate. Several logical observations can be made about the use of a hit rate. First, wider prediction intervals are less precise and will typically produce higher hit rates; conversely, smaller prediction intervals are more precise and will typically produce lower hit rates. In other words, precision and accuracy are inversely proportional, generally tasking the estimator with balancing the two.

For ease of use, the SISE model deliberately takes a statistically simplistic approach to assigning confidence levels; the model assumes the software engineer will repeatedly employ the same method for determining relative size and creating estimates. Based on this assumption, the estimator's past performance can be used as a predictor of future performance. For example, if an estimator's hit rate is 50%, it can be said that half of the activities they have estimated have had actual effort values that fell within his or her prediction interval. Therefore, all things being equal, a new estimate has a 50% probability of being correct. Put another way, the estimator has a 50% confidence level in his or her next estimate.

Note that confidence level should not be confused with an estimator's logical or emotional confidence in his or her abilities and estimates. It can be assumed that when an estimator produces an estimate, he or she does so to the best of their ability; the estimator is confident the estimate is correct. Confidence level, on the other hand, is a measure of the probability that the estimate will be correct and allows the estimator to make statements such as:

In the past, my estimates have been correct 90% of the time. Therefore, I have a 90% confident level in my next estimate, which I feel confident I have done my best in constructing.

Beginning with the first estimate, the SISE model assigns each new estimate a confidence level based on the estimator's current hit rate. As noted in the fourth step of SISE, however, the estimator may take steps to adjust this confidence level by compensating for historical

Smaller →	Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
	Design security model	0	∞	10
← Larger				
<i>Hit rate = 100%</i>				

Table 3.1: One completed activity.

bias (see sidebar Adjusting for Width Bias and Adjusting for Shift Bias). Note that shift and width biases are not to be viewed as errors on the part of the estimator; rather they are to be viewed as the manner in which the SISE model adapts to an individual software engineer’s perspective of past and future work.

3.6 SISE Example

Assume a software engineer, who has never engaged in time tracking, has decided to begin using the SISE model for his web development project. The developer been assigned a new work activity: “Design security model.” Given that the software engineer’s historical record is empty, he has no data points for an estimate; no smaller task or larger task can be identified to use as the basis for a prediction interval. Therefore, following the SISE model, the prediction interval for the first activity is $[0, \infty]$. Once the first activity is completed and the actual value is recorded, the hit rate is calculated to be 100% (see Table 3.1).

The next activity assigned to the software engineer is to “Design the user model.” Since only one items exists in the historical record, the first SISE step (sorting) is complete by default. Our software engineer hides all but the first column and compares the future activity’s description to the task description in the historical record. He decides that designing a user model is easier than designing a security model; we have a larger historical task, but no smaller one. The estimate, therefore, is a prediction interval of $[0,10]$. Our confidence in the estimate is equal to the hit rate, which is currently 100%.

Smaller →	Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
	Design user model	0	10	8
← Larger	Design security model	0	∞	10
<i>Hit rate = 100%</i>				

Table 3.2: Two completed activities.

Smaller →	Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
	Design user model	0	10	8
← Larger	Design security model	0	∞	10
	Design content model	8	10	11
<i>Hit rate = 67%</i>				

Table 3.3: Three completed activities.

Work proceeds and the activity is completed in eight hours. The estimate and actual are recorded and the new hit rate is calculated to be 100% (see Table 3.2). For convenience, the historical data in these examples will be kept sorted from smallest to largest task.

The third activity is assigned to the software engineer: “Design the content model.” Our software engineer scans the historical record, after hiding the numeric values, and decides that “designing a user model” is smaller and “designing a security model” is larger. Therefore, the prediction interval for the future activity is set at [8, 10]. The work is completed with an actual effort of 11 hours, giving a new hit rate of 67%, with two of the three completed tasks falling within his prediction intervals (see Table 3.3).

A fourth activity is assigned to the software engineer: “design database tables.” By scanning the historical record’s task descriptions, the software engineer decides the confidently larger task is “design content model,” but is unable to designate a smaller task. The prediction interval, therefore, is set as [0, 11].

Smaller →	Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
		Design database tables	0	11
	Design user model	0	10	8
← Larger	Design security model	0	∞	10
	Design content model	8	10	11
<i>Hit rate = 75%</i>				

Table 3.4: Four completed activities.

The confidence level is assumed to be 67%, based on the historical hit rate. After referring to the sidebars on adjusting for bias, the software engineer considers making a shift adjustment. A one-hour upward shift of all the historical prediction intervals would move the hit rate from 67% to 100%. This leaves the estimator with two choices. The estimate’s prediction interval could be shifted one hour upward to account for a possible historical bias, or the estimate could be left alone. In short, the estimator now has two options to choose from: $[0, 11]$ with a 67% confidence level or $[1, 12]$ with a confidence level of 100%. Assume the estimator chooses to not shift the estimate due to the small data set size; the work is performed and recorded (see Table 3.4).

Assuming the software engineer proceeds in this fashion, he will accumulate a sizable historical record. With each hit or miss within the prediction interval, the hit rate will rise and fall. The software engineer may, at some point, choose to adjust a future estimate for width bias in order to increase his confidence level in a new estimate. Here’s a simple example, assuming ten completed tasks, with no verifiable shift bias to correct.

As Table 3.5 indicates, the hit rate is 70%, with three of the ten tasks falling outside their prediction intervals. A future activity, “Create Contact Us page,” has been assigned a prediction interval of $[2, 8]$ and the confidence level is assumed to be 70%. In this case, however, the manager has requested a higher confidence level. To accomplish this, the software engineer adjusts for width bias.

Smaller → ← Larger	Task	Low Est. (hours)	High Est. (hours)	Actual (hours)	Missed Prediction Interval?
	Design FAQ data model	0	4	2	
	Create FAQ classes	2	6	2	
	Create security classes	2	8	3	
	Create user classes	5	8	4	Yes
	Create database tables in MySQL	0	6	5	
	Design database tables	0	11	6	
	Design user model	0	10	8	
	Design security model	0	∞	10	
	Design content model	8	10	11	Yes
Create data connector classes	0	11	14	Yes	
<i>Hit rate = 70%</i>					

Table 3.5: Ten completed activities.

Smaller → ← Larger	Task	Low Est. (hours)	Adj. Low	High Est. (hours)	Adj. High	Actual (hours)	Missed Prediction Interval?
	Design FAQ data model	0	0	4	5	2	
	Create FAQ classes	2	1	6	7	2	
	Create security classes	2	1	8	9	3	
	Create user classes	5	4	8	9	4	
	Create database tables in MySQL	0	0	6	7	5	
	Design database tables	0	0	11	12	6	
	Design user model	0	0	10	11	8	
	Design security model	0	0	∞	∞	10	
	Design content model	8	7	10	11	11	
Create data connector classes	0	0	11	12	14	Yes	
<i>Hit rate = 90%</i>							

Table 3.6: Adjusting for width bias.

The margins of error for each of the three tasks are one hour, one hour, and three hours, respectively. If the prediction intervals for all historical tasks were increased by one hour in each direction, the hit rate would rise to 90%. See Table 3.6.

Therefore, the prediction interval for the future activity “Create Contact Us page” must also be adjusted using a one-hour expansion, making it [1, 9] with a confidence level of 90%. In summary, the software engineer has a choice between two, fact-based estimates: [2,8] with a 70% confidence level or [1,9] with a 90% confidence level.

Each of the subsequent iterations through the SISE model follows a similar pattern to those reviewed above. The software engineer is assigned a new activity to complete. The activity is compared to previously completed tasks to identify a smaller and larger task,

which leads to a prediction interval. The prediction interval is adjusted, if necessary and possible, to achieve a desired confidence level or prediction interval.

3.7 Validation of SISE

The SISE model has been validated through a multi-step process. First, over 100 software engineering students participated in a relative sizing activity, where they were asked to identify the larger of two tasks, based solely on the task descriptions. The results demonstrated that a majority of students were able to identify the larger task two-thirds of the time. Equally as important, the results indicated that students, on average, were unlikely to incorrectly identify a task's size; instead, they tended to identify the tasks as similar in size.

The next step in validating SISE involved sizing estimates using classroom programming assignments. Each student constructed a SISE-style estimate, as well as, an estimate based on a proxy-based model, derived from PSP's PROBE model. Overall, the SISE model's predictions proved no more or less accurate than the proxy-based approach. In addition, the students indicated that SISE, in their opinion, took less time and was based on less a complex model.

3.8 Conclusion

The SISE model represents an empirically based approach to effort estimation that relies less on complex mathematical models and more on intuitive expert judgment, without sacrificing the quality of the final product. Software engineers willing to take the first tentative steps toward adopting a personal process now have access to a truly lightweight, agile estimation model. The SISE model does not burden the practitioner with any more work than the absolute minimum necessary to produce a reasonably accurate, fact-based effort estimate. In addition, the model is the first of its kind, suitable for use by a single software engineer.

Further development and improvements to the model are currently underway at Auburn University's microISV Research Lab. We are formalizing ways in which the SISE model may be integrated into team-based software processes, as well as tool development. For more information, visit our website at <http://microisvresearch.org>.

Chapter 4

Validation of the SISE Model

4.1 Abstract

Personal software processes rely on individuals approaching software development activities in a systematic and structured manner. Time management has been shown to be a key component in meeting obligations, but relies on relatively accurate effort estimation techniques. Despite a plethora of research into team-based effort estimation models, few models are suitable for use by individual software engineers. Models tailored to the individual include guesswork, an approach commonly used in industry; the PCSE model, under development at Auburn University; and the PROBE model, the only peer-reviewed model devoted to lone software engineers. This spectrum of choices features a gap between guesswork and more formal models, which could be filled with a lightweight, agile, and reasonably accurate alternative. The SISE model combines expert judgment – in the form of relative sizing decisions – with empirical, historical data to create such an alternative. Four key features of the model have undergone validation through a series of surveys and experiments: relative sizing by software engineers, model accuracy, perceived time investment, and perceived value. This research has demonstrated that software engineers are generally capable of sizing development tasks relative to each other based solely on the tasks' descriptions, which is a key feature of SISE. Additionally, this research has demonstrated that the SISE model's accuracy is not significantly different from that of PROBE, its nearest validated competitor. This research has also indirectly demonstrated that software engineers perceive the SISE model to require a smaller time investment than the PROBE mode, by relative comparison to the PCSE model, which represents a subset of the PROBE model. Lastly, this

research has demonstrated that software engineers perceive the value of the SISE model's results to be higher than that of guesswork.

4.2 Introduction

Software engineering as a discipline seeks to create a systematic and structured approach to the development of software. In order to effectively manage software development efforts, monitoring and controlling activities must be applied, especially at the team level. However, such management activities should also be self-imposed by individual software engineers as they manage themselves and their individual efforts. Time management is one of the most critical of these elements. A software engineer must be willing and capable of properly allocating his or her time and effort as necessary to either complete tasks on time or provide project management resources with the information necessary to properly manage risks.

A major component of time management is the forecasting of future effort, known as effort estimation or, less formally, sizing. The ability of a software engineer or engineering team to predict future effort directly impacts the ability to allocate resources, schedule development cycles, and meet client needs in a timely fashion. As a result, a great deal of time and effort has been spent developing estimation models for teams. These models encompass the full spectrum, from lightweight, agile models such as Planning Poker, to complex, all-encompassing models such as COCOMO.

The vast majority of effort estimation models rely on a team of individuals to set up and implement. Planning Poker, for example, relies on the members of a team to discuss requirements, share their thoughts, and iteratively refine a series of estimates. COCOMO, on the other hand, requires weeks, months, or years of data collection and planning to properly tune the core metrics that drive the underlying estimation model, which is focused on large-scale effort that are too large for a single developer to attempt. As a result, individual software engineers do not have access to a simple, agile estimation tool to use for their own benefit when posed the all-too-common question “When can you have this done?”

Two estimation models exist, which were specifically designed for the individual software engineer. Watts S. Humphrey introduced the first model – PROBE – in *A Discipline for Software Engineering* [17]. PROBE relies on careful record-keeping and the use of proxies to estimate future effort. The underlying model is built upon a series of complex mathematical formulas and workflows to derive an estimate. The second model – PCSE – is currently under development at Auburn University and it also relies on the use of proxies to build estimates, but uses only a subset of PROBE features.

The final estimation approach frequently used in industry is expert judgment, commonly known as guesswork [21]. Expert judgment relies on an individual software engineer’s ability to assign a numeric value to the estimate based on his or her knowledge, experiences, and intuition. Expert judgment is not a prescriptive model and estimators may use a variety of conscious or subconscious approaches to construct the estimate, such as analogy or work breakdown [42] [26].

Unlike the complete spectrum of effort estimation options available to teams, the individual software engineer’s array of estimation tools has a gap. The gap falls directly between expert knowledge and the two aforementioned models (PROBE and PCSE). Any viable solution for filling this gap would be best describes as lightweight, agile, and reasonably accurate.

4.3 SISE: An Agile Estimation Model

The SISE model is designed to provide individual software engineers with a lightweight, agile, and reasonably accurate effort estimation model. The SISE model is built upon a few common estimation concepts. First, SISE relies on regression, the principle that future effort may be predicted based on past actual effort. Second, effort predictions should always be formulated as a prediction interval: a range in which the actual effort is expected to fall. Third, a confidence level must be assigned to any estimate to indicate the likelihood of the

actual effort falling within the prediction interval. Lastly, the confidence level for an estimate is directly related to the practitioner’s historical accuracy in constructing estimates.

Building on these four concepts, the SISE model assumes the following: a past activity, which is perceived to be smaller than a future activity, may be used as the low bound of the future task’s prediction interval; the reverse is also true for a larger activity used as the high bound of the prediction interval. In addition, an analysis of an estimator’s historical accuracy (or error) in predicting future effort can be used to establish an optimal size for the prediction interval as it correlates to a desired confidence level.

“SISE” is an acronym for the four-step process underlying the model: Sort, Identify, Size, Evaluate. The model begins by having the estimator *sort* his or her historical work activities by actual effort from smallest to largest. Next, a future task’s description is compared to the descriptions of the past activities; comparisons of numbers (e.g. actual effort, estimated effort, etc.) should be avoided to prevent the introduction of biases based on unconscious guesswork by the estimator. The estimator must *identify* two tasks, one perceived to be smaller than the future activity, one perceived to be larger, and both as close to the future activity as possible. The actual effort values of these two tasks constitute the rough prediction interval, or *size*, of the future task. The last step involves *evaluating* the rough estimate and – if the estimator is dissatisfied with the precision, accuracy, or confidence level of the estimate – adjusting for historical bias.

4.4 Relative Sizing

The first step in validating the SISE model lies in an examination of its core underlying assumption: software engineers are generally capable of identifying the larger of two software developments tasks. This principle of the SISE model was tested through a relative sizing survey.

4.4.1 Hypothesis

The SISE model relies on the abilities of an estimator to predict the relative size of each future activity, as compared to past, completed tasks. To confirm that a software engineer is generally capable of relative sizing, a survey was constructed to test the following hypothesis:

Ha₁: An estimator is, on average, capable of identifying the larger of a pair of tasks, in terms of required effort to complete (selection accuracy > 0.5).

with the null hypothesis as

H₀: An estimator is, on average, unable to identify the larger of a pair of tasks, in terms of required effort to complete (selection accuracy ≤ 0.5).

4.4.2 Survey

Project data was gathered from the Auburn University’s Software Process course for the last 10 years; the data represented 4,060 individual programming projects for 772 students with 53 unique programming assignments. Students completing these assignments were required to maintain a time log of their efforts. Table 4.1 lists the name, average effort, and sample size for nine distinct tasks. *Appendix C – Relative Sizing Survey Questions* contains the ten survey questions with the stated requirements, as provided to the students completing the work and those responding to the survey.

Data Distribution

An inspection of the actual effort values demonstrated that the data points do not follow a normal distribution. Figure 4.1 shows the entire dataset and Figure 4.2 shows the less extreme data points between 1 and 1,000 minutes. Due to the non-normal distribution of data, parametric tests, such as a t-test, were ruled inappropriate for data analysis; therefore, non-parametric tests were utilized, such as the Mann–Whitney U test.

Assignment	Avg. Effort (min.)	Sample Size
CriticalPath	408.9	114
T-Dist	292.5	279
T-Dist2	285.7	29
ComponentInfo	265.2	27
5-Slot	215.8	29
Text	202.5	38
CalcCorr	200.6	28
A-M-S	194.0	84
M-P-P-S	143.1	49

Table 4.1: Average construction effort (in minutes).

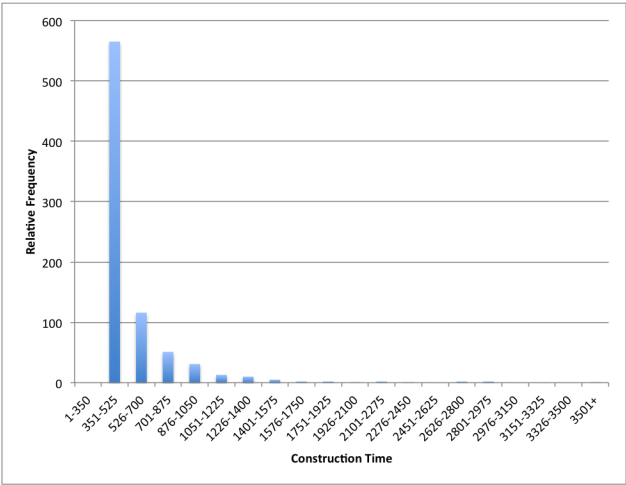


Figure 4.1: Distribution of actual construction times (all data).

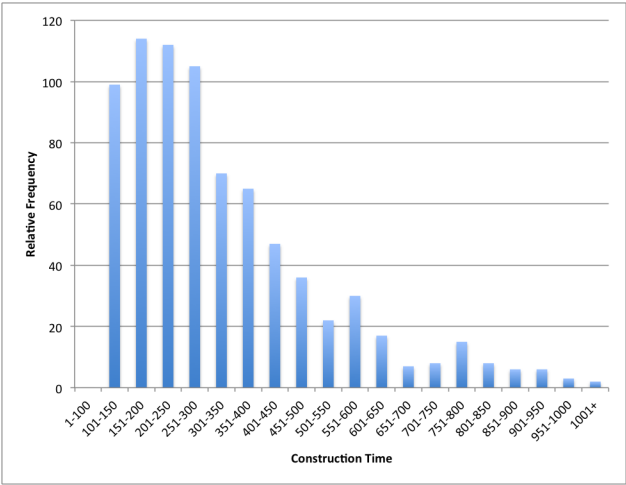


Figure 4.2: Distribution of actual construction times (values < 1,000 minutes).

No.	Assignment Pair	Diff.	Psuedomedian	95% CI	P-Value
1	T-Dist vs. M-P-P-S	173	105.0	75.0 - INF	1.007e-09
2	CriticalPath vs. T-Dist	91	67.0	37.0 - INF	0.0001667
3	T-Dist vs. 5-Slot	90	62.0	15.0 - INF	0.01516
4	Text vs. M-P-P-S	57	47.4	16.0 - INF	0.004951
5	T-Dist2 vs. CalcCorr	95	46.0	-5.0 - INF	0.06837
6	5-Slot vs. M-P-P-S	83	43.8	-5.4 - INF	0.05171
7	ComponentInfo vs. 5-Slot	362	38.0	-34.0 - INF	0.1399
8	Text vs. A-M-S	6	22.0	-6.0 - INF	0.1012
9	Text vs. CalcCorr	2	21.0	-20.0 - INF	0.1689
10	A-M-S vs. M-P-P-S	51	27.0	-3.0 - INF	0.06977

Table 4.2: Average construction time comparison for assignment pairs.

Task Pair Selection

Based on the student time logs, four *diverse* task pairs, numbered 1 through 4, were identified, which have a statistically significant difference in size (95% confidence) based on actual construction effort. Six *uniform* task pairs, numbered 5 through 10, were identified that had no statistically provable difference in size. The statistical difference was determined via a Mann–Whitney U test using R. Output Listings 1 - 10 show the results from R; Table 4.2 summarizes the task pairs, the difference in average construction times, an estimate of the psuedomedian, the 95% nonparametric confidence interval for the difference, and the associated p-value for the comparison.

4.4.3 Metrics

Estimators presented with two tasks that are significantly different in size (i.e. a correct answer exists) should demonstrate a significant tendency to choose the larger of the two. Given that random chance in an A/B scenario would result in 50% accuracy, a group of estimators who demonstrate selection accuracy significantly *greater than 50%* are making selections in a non-random manner. To determine if the group’s selection accuracy is significantly greater than 50%, a 1-sample proportion test was employed using R (accuracy > 0.5).

Conversely, estimators presented with two tasks that are not significantly different in size (i.e there is no correct answer) should approximate random chance in their selections. In this case, a group of estimators who demonstrate selection accuracy significantly *different from 50%* are making selections in a non-random manner. To determine if the group’s selection accuracy is significantly different from 50%, a 1-sample proportion test was employed using R (accuracy $\neq 0.5$).

An additional metric was employed to determine if a significant proportion of the respondents were able to select a simple majority of correct answers; since only task pairs 1-4 had a statistically correct answer, the respondents’ answers to task pairs 5-10 were excluded from consideration. To determine if this proportion was significant, the number of respondents who correctly identified more than half the larger tasks was compared to the total number of respondents, via a 1-sample proportion test using R (proportion > 0.5). Output Listing 21 lists the results.

Finally, a linguistic analysis was performed on the rationales provided by each respondent to determine what general themes motivated his or her selections. The analysis involved a subjective assignment of a theme category based on word and phrase choice, such as “complex,” “familiar with,” or “number of methods.” The number of responses including each theme was tabulated and compared.

4.4.4 Participants

Survey respondents were selected from Auburn University’s Department of Computer Science and Software Engineer classes, including *Software Process* (COMP 5700) and *Data Structures* (COMP 2210), in Fall 2012. The Software Process course is designed to provide: insight into process-oriented software development; exposure to common engineering processes; and experience with a software process [12]. The Data Structures course is a continuation of the introductory programming course with emphasis on data structures such

as lists, trees, graphs, and hash tables [6]. A total of 113 software engineering students responded.

4.4.5 Questions and Presentation

The survey was presented as a ten page questionnaire, with each page detailing two task descriptions, side-by-side. *Appendix D – Attitudinal Survey Questions* contains the complete descriptions of each of the nine tasks used in the task pairs. The following instruction were provided with each task pair:

Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the “larger” task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Respondents were not allowed to specify that a task pair was equal in size and had to choose a larger task in all instances.

In addition to the relative sizing questions, each respondent was asked to identify his or her enrollment (undergraduate, masters, doctoral), program of study, and completed courses.

4.4.6 Results

Table 4.3 shows the respondents’ accuracy for task pairs 1-4 and the respondents’ selection of the first vs. second task for task pairs 5-10; the table also shows the corresponding 95% confidence interval for the proportion of correct answers.

Figure 4.3 also shows the results of the survey, including the accuracy of the responses and the corresponding 95% confidence interval for the proportion. The possibility that a particular answer was selected at random is 50%, as demonstrated by the red line. Response ranges that do not include the 50% mark are considered to be non-random.

	Task Pair	Correct	95% Prop. CI
Diverse	1	84%	77-100%
	2	71	63-100
	3	50	41-100
	4	63	63-100
Uniform	Task Pair	First Task	95% Prop. CI
	5	46%	36-55%
	6	81	72-87
	7	56	47-66
	8	67	58-77
	9	68	59-76
	10	95	88-98

Table 4.3: Relative Sizing Survey Results.

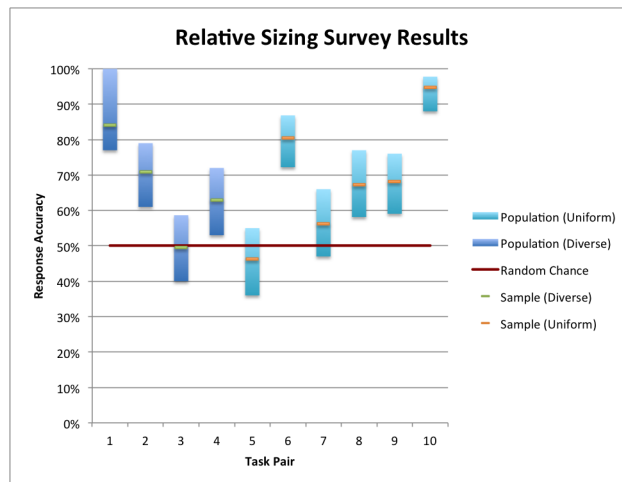


Figure 4.3: Relative sizing survey results.

Theme	M-P-P-S		T-Dist		Total	
	Count	Pct.	Count	Pct.	Count	Pct.
Data	3	17%	0	0%	3	3%
Familiarity	3	17	14	15	17	15
Math	3	17	15	16	18	16
Methods	5	28	5	5	10	9
Planning	0	0	3	3	3	3
Problem	1	6	7	7	8	7
Reuse	0	0	4	4	4	4
Simplicity	2	11	44	46	46	41
Testing	0	0	2	2	2	2
Unspecified	1	6	1	1	2	2
Total	18	16	95	84	113	

Table 4.4: Task pair 1 distribution of themes in respondents’ rationales.

Diverse Pairs

For task pairs 1, 2, and 4, the proportion of correct responses was significantly greater than 50%, indicating non-random selection. The proportion of correct responses for task pair 3 was not significantly different from 50% and failed to demonstrate non-random selection. The linguistic analyses of themes in the respondents’ rationales are listed in Tables 4.4 - 4.7.

For task pair 1, a significant majority of the respondents correctly selected the T-Dist assignment as larger. The dominant theme was the *simplicity* of the M-P-P-S assignment or the *complexity* of the T-Dist assignment, indicating a majority of the respondents intuitively selected the larger task.

For task pair 2, a significant majority of the respondents correctly selected the CriticalPath assignment as larger. Although the dominant theme was, again, the *simplicity* or the *complexity*, the respondents for this theme were roughly split between the two assignments. However, a significant number of respondents voting in favor of the CriticalPath assignment as larger, cited their *familiarity* with the assignment as the rationale, breaking the tie. In addition, a large number of respondents cited the *data* handling requirements and other *problem-specific* features of the Critical Path assignment as reason for choosing it as the larger task. In summary, this indicates that a roughly equal number of students

	CriticalPath		T-Dist		Total	
Theme	Count	Pct.	Count	Pct.	Count	Pct.
Data	10	13	2	6	12	11
Familiarity	15	19	1	3	16	14
Math	1	1	2	6	3	3
Methods	6	8	4	12	10	9
Planning	7	9	2	6	9	8
Problem	11	14	4	12	15	13
Reuse	1	1	0	0	1	1
Simplicity	22	28	18	55	40	35
Testing	4	5	0	0	4	4
Unspecified	3	4	0	0	3	3
Total	80	71	33	29	113	

Table 4.5: Task pair 2 distribution of themes in respondents’ rationales.

made an intuitive decision about the assignments’ relative size, some correctly and some incorrectly. However, a significant number of students made their choices based on the larger assignment’s complexity by evaluating the problem details, resulting in a correct decision, on average.

For task pair 3, the respondents were split 56/57 between the T-Dist and 5-Slot assignments, despite the T-Dist assignment being historically larger. The provided rationales indicated that a majority – 37 vs. 12 – of respondents intuitively (and incorrectly) viewed the 5-Slot assignment as more complex or the T-Dist assignment as simpler. On the other hand, a larger majority – 22 vs. 2 – of respondents (correctly) cited the T-Dist assignment as larger due to the size of the assignment in terms of lines of code, methods, or functions; in fact, an optimal implementation of the 5-Slot assignment will primarily involve a single method, called by the remaining methods with different parameter values. In summary, a significant portion of the respondents failed to recognize certain attributes of the assignments, such as code reuse, leading to an incorrect relative sizing.

For task pair 4, a significant majority of the respondents correctly selected the Text assignment as larger. Although the dominant theme was the *simplicity* or the *complexity*, the respondents for this theme were roughly split between the two assignments. However,

Theme	T-Dist		5-Slot		Total	
	Count	Pct.	Count	Pct.	Count	Pct.
Data	4	7	0	0	4	4
Familiarity	6	11	6	11	12	11
Math	2	4	0	0	2	2
Methods	22	39	2	4	24	21
Problem	2	4	2	4	4	4
Reuse	1	2	4	7	5	4
Simplicity	12	21	37	65	49	43
Testing	0	0	2	4	2	2
Unspecified	7	13	4	7	10	9
Total	56	50	57	50	113	

Table 4.6: Task pair 3 distribution of themes in respondents' rationales.

a significant number of respondents voting in favor of the Text assignment as larger, cited the complexities of *file* processing or other *problem-specific* attributes of the assignment, breaking the tie. In summary, this indicates that a roughly equal number of students made an intuitive decision about the assignments' relative size, some correctly and some incorrectly. However, a significant number of students made their choices based on the larger assignment's complexity by evaluating the problem details, resulting in a correct decision, on average.

In summary, when presented with two tasks of distinctly different size, a majority of software engineering students were able to successfully identify the larger task in three-quarters of the instances.

Uniform Pairs

An analysis of the responses to the uniform task pairs revealed varying results. In a situation where there is no discernible difference in size for two tasks, one would expect the survey responses to approach a random distribution in choosing between one task and the other (i.e. the responses should be split 50-50). However, this only occurred for two of the six uniform task pairs. For the other four, a majority of respondents selected one of the tasks as larger.

	M-P-P-S		Text		Total	
Theme	Count	Pct.	Count	Pct.	Count	Pct.
Data	1	2	1	1	2	2
Familiarity	5	12	4	6	9	8
File	2	5	12	17	14	12
Math	4	10	5	7	9	8
Methods	5	12	3	4	8	7
Planning	1	2	6	8	7	6
Problem	8	19	11	15	19	17
Reuse	2	5	2	3	4	4
Simplicity	14	33	17	24	31	27
Testing	0	0	6	8	6	5
Unspecified	0	0	4	6	4	4
Total	42	37	71	63	113	

Table 4.7: Task pair 4 distribution of themes in respondents’ rationales.

A linguistic analysis was performed on the respondents’ rationale for choosing a particular task. The analysis indicated that respondents were influenced either by specific aspects of the problem definition or by the format, word count, and word choice of the requirements.

For task pair 5, the respondents were split 59/51, which is consistent with the fact that neither assignment is historically larger than the other. The respondents cited a variety of themes – familiarity, length of code, problem details, simplicity – with no single theme demonstrating a significant majority. In summary, the students were unable to cite a single, consistent reason for sizing one task larger than the other.

For task pair 6, respondents selected the 5-Slot assignment as larger than the M-P-P-S assignment by more than a 4-to-1 margin. The two most commonly cited themes were the relative *simplicities/complexities* of the assignments and the length of the required code. In summary, a majority of the respondents incorrectly viewed the 5-Slot assignment as larger than the M-P-P-S assignment, similar to the results of task pair 3.

For task pair 7, the respondents were split 57/55, which is consistent with the fact that neither assignment is historically larger than the other. The respondents cited a variety of themes – length of code, reuse, simplicity – with no single theme demonstrating a significant

Theme	CalcCorr		T-Dist2		Total	
	Count	Pct.	Count	Pct.	Count	Pct.
Data	1	2	0	0	1	1
Familiarity	9	15	4	8	13	12
File	1	2	0	0	1	1
Math	3	5	3	6	6	5
Methods	7	12	18	35	25	23
Planning	1	2	0	0	1	1
Problem	3	5	7	14	10	9
Reuse	1	2	0	0	1	1
Simplicity	27	46	15	29	42	38
Testing	0	0	1	2	1	1
Unspecified	6	10	3	6	9	8
Total	59	54	51	46	110	

Table 4.8: Task pair 5 distribution of themes in respondents' rationales.

Theme	M-P-P-S		5-Slot		Total	
	Count	Pct.	Count	Pct.	Count	Pct.
Data	0	0	1	1	1	1
Familiarity	0	0	2	2	2	2
File	1	5	0	0	1	1
Math	2	9	6	7	8	7
Methods	6	27	51	56	57	50
Problem	7	32	8	9	15	13
Reuse	3	14	2	2	5	4
Simplicity	2	9	17	19	19	17
Testing	0	0	2	2	2	2
Unspecified	1	5	2	2	3	3
Total	22	19	91	81	113	

Table 4.9: Task pair 6 distribution of themes in respondents' rationales.

Theme	ComponentInfo		5-Slot		Total	
	Count	Pct.	Count	Pct.	Count	Pct.
Data	3	5	1	2	4	4
Familiarity	2	4	5	9	7	6
File	1	2	3	5	4	4
Math	4	7	5	9	9	8
Methods	9	16	11	20	20	18
Planning	3	5	3	5	6	5
Problem	4	7	1	2	5	4
Reuse	8	14	6	11	14	13
Simplicity	13	23	16	29	29	26
Testing	2	4	1	2	3	3
Text	5	9	2	4	7	6
Unspecified	3	5	1	2	4	4
Total	57	51	55	49	112	

Table 4.10: Task pair 7 distribution of themes in respondents' rationales.

majority. In summary, the students were unable to cite a single, consistent reason for sizing one task larger than the other.

For task pair 8, respondents selected the Text assignment as larger than the A-M-S assignment by a 2-to-1 margin. Two of the three most commonly cited themes were the relative *simplicities/complexities* of the assignments and the *mathematics* involved; references to these themes were roughly split between the two assignments. The third commonly referenced theme was challenge of implementing code with *file* handling; respondents in favor of the Text assignment as larger referenced this theme more often by a 10-to-1 margin. In summary, a majority of the respondents incorrectly viewed the Text assignment as larger than the A-M-S assignment due to the perceived complexities of the required file operations.

For task pair 9, respondents selected the Text assignment as larger than the CalcCorr assignment by more than a 2-to-1 margin; it should be noted that the CalcCorr assignment was accompanied by a six-page handout describing, in detail, the process for completing the required mathematical calculations. The most commonly cited themes were the students' *familiarity* with the requirements, relative *simplicities/complexities* of the assignments, the requirement to read *files*, the requirements to process *text*, and the *mathematics* involved. Of

	Text		A-M-S		Total	
Theme	Count	Pct.	Count	Pct.	Count	Pct.
Data	1	1	0	0	1	1
Familiarity	7	9	2	5	9	8
File	20	26	2	5	22	19
Math	10	13	14	38	24	21
Methods	4	5	4	11	8	7
Planning	4	5	0	0	4	4
Problem	1	1	1	3	2	2
Reuse	1	1	0	0	1	1
Simplicity	18	24	7	19	25	22
Testing	2	3	0	0	2	2
Text	1	1	0	0	1	1
Unspecified	7	9	7	19	14	12
Total	76	67	37	33	113	

Table 4.11: Task pair 8 distribution of themes in respondents’ rationales.

the respondents selecting the CalcCorr assignment as larger, a significant portion referenced the file and text processing themes; of the respondents selecting the Text assignment as larger, a significant portion referenced their familiarity, the mathematics, and the intuitive judgment of the relative simplicity/complexity. In summary, a majority of the respondents incorrectly viewed the Text assignment as larger than the CalcCorr assignment due to a general attitude was that unstructured text processing presenting more inherent difficulties than a highly structured and well-documented, complex math formula.

	CalcCorr		Text		Total	
Theme	Count	Pct.	Count	Pct.	Count	Pct.
Familiarity	2	6	11	14	13	12
File	7	19	2	3	9	8
Math	5	14	33	43	38	34
Planning	4	11	6	8	10	9
Simplicity	5	14	21	27	26	23
Testing	1	3	1	1	2	2
Text	12	33	0	0	12	11
Unspecified	0	0	3	4	2	2
Total	36	32	77	68	113	

Table 4.12: Task pair 9 distribution of themes in respondents’ rationales.

For task pair 10, 95% of the respondents selected the M-P-P-S assignment as larger than the A-M-S assignment, despite the fact that, based on the historical data, the A-M-S assignment took an average of 51 minutes longer to complete (though this difference is not statistically significant, given the small dataset). A review of the requirements and student comments revealed the A-M-S assignment to be a subset of the M-P-P-S assignment; in other words, to complete the M-P-P-S assignment, a software engineer would need to first complete all the requirements of A-M-S, then, additional work would need to be done to complete M-P-P-S. Interestingly, 95% of the respondents recognized this relationship and identified the M-P-P-S assignment as larger.

In summary, when presented with two tasks in which neither task is significantly larger than the other, software engineering students do not rely on a single, consistent method or process; their decision-making involves a variety of influencers such as personal experience, problem-specific attributes, word choice, and/or description length.

4.4.7 Individual Respondents

An analysis was performed to determine if a significant proportion of the respondents were able to select a simple majority of correct answers. Since task pairs 1-4 were the only pairs with a demonstrably correct answer, task pairs 5-10 were excluded. Of the 113 respondents, 72 correctly identified at least 3 out of the four larger tasks. Figure 4.4 shows the distribution of correct answers among individual respondents and Output Listing 21 shows the 1-sample proportion test conducted using R (accuracy > 0.5).

In summary, the 95% confidence interval for the proportion of respondents with a simple majority of correct answers is 55.6-100% with a p-value < 0.0001 making the results significant.

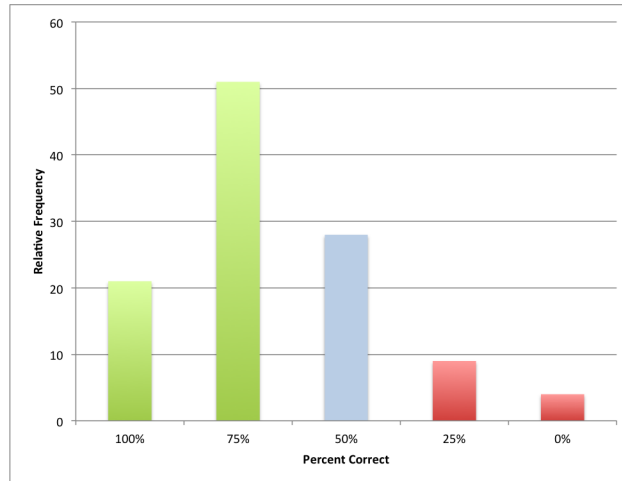


Figure 4.4: Distribution of percentage of correct answers (task pairs 1-4).

4.4.8 Conclusions

Overall, the survey results indicate that software engineers are generally capable of identifying the larger of two tasks when the size difference is statistically significant. When the two tasks are not significantly different in size, software engineers tend to choose based on their perceptions of the task complexity, as influenced by the problem description. Lastly, individual responses indicate that a significant proportion of software engineers are capable of selecting a simple majority of correct answers.

Decision: Reject the null hypothesis $H0_1$ in favor of the alternate Ha_1 , concluding that software engineers are generally capable of identifying the larger of two tasks.

4.5 Accuracy

Direct validation of the accuracy of the SISE model began in the Software Process classroom in Fall 2012 and continued through Spring 2013. Prior to beginning their last assignment of the semester, the students were asked to review the requirements and size it relative to their past assignments; this relative sizing formed the basis for a SISE estimate. The accuracy of these estimates was then compared to the accuracy of PROBE estimates, collected from the Software Process course over a period from 2001 to 2008.

4.5.1 Hypothesis

The accuracy of the SISE model should, at a minimum, equal the accuracy of existing, validated models, tailored to the individual. To confirm the relative accuracy of SISE, an experiment was constructed to test the following hypothesis:

H_{a2}: The SISE model produces estimates that are equally or more accurate than PROBE ($accuracy_{SISE} \geq accuracy_{PROBE}$).

with the null hypothesis as

H₀₂: The SISE model produces estimates that are less accurate than PROBE ($accuracy_{SISE} < accuracy_{PROBE}$).

4.5.2 Experiment

Students enrolled in the Software Process course were taught and required to follow a personal software process. Each coding assignment was accompanied by a spreadsheet, completed by the student, which detailed his or her adherence to an individual software process and activities as they related to the completion of the assignment. The components of the spreadsheet that are relevant to this research include the description of the assignment's requirements, a detailed time log, and various estimates. Each student was instructed to keep a careful record of time spent during each aspect of construction and project management.

From 2001 through 2008, the students in the Software Process course completed a portion of their assignments by providing a PROBE effort estimate. A total of 34 distinct assignments that included PROBE estimates were completed by students across twelve semesters. Within that sample, 406 assignments produced valid estimates utilizing a prediction interval. These estimates formed the basis for the accuracy measurement of the PROBE model.

In the Fall 2012 and Spring 2013 classes, students compiled a historical record of their actual effort values by completing a series of six to seven coding assignments. Prior to beginning work on their last assignment, the students completed a brief survey on the relative

sizing for the future assignment as compared to the completed ones; the results were used to construct a SISE-style estimate.

From 2001 through 2012, for all assignments that did not include a PROBE or PCSE estimate, the students were required to construct an estimate prior to beginning development work based on their expert judgment. This estimate was included in their spreadsheets and turned in with their completed work. No mechanism was imposed to prevent these estimates from being constructed after development had begun or after it had been completed. Due to this fact, the estimates based on expert judgment were not considered reliable to use for accuracy comparisons.

The accuracy values for the PROBE and SISE models were compared to determine if a significant difference in their respective accuracies existed.

4.5.3 Metrics

Both the SISE and PROBE models produced an estimate containing a prediction interval. In addition, the PROBE model produced a single, *planned effort* value. The SISE model, however, does not employ a prescriptive method for translating its prediction interval into a single value estimate. Therefore, to construct a valid comparison between the two models, the metric employed must rely on a comparison of prediction intervals, as opposed to single value estimates.

Two methods are commonly used to evaluate estimation model accuracy. The first method involves the calculation of the *mean absolute relative error* (MARE) for a set of estimates [11]. The MARE value (see Equation 4.1) is an average of the magnitude of all estimate errors and requires single value estimates to calculate.

$$MARE = \frac{1}{n} \sum_{i=1}^n \left| \frac{(actual_i - estimate_i)}{actual_i} \right| \quad (4.1)$$

where n is the number of individual tasks in the set.

The second method of comparison involves a calculation of the estimation model’s hit rate and requires that the original estimates be phrased in terms of a prediction interval.

A *prediction interval* (PI) represents the low and high bounds of the estimate. For example, an estimate of “90 to 110 person hours” for an activity has a prediction interval of [90,110]. The *prediction interval width* (PI width) is the difference between the high and low bounds. The *hit rate* is defined as the percentage of time the actual effort falls within the prediction interval [23]; Equation 4.2 shows the formula for calculating the hit rate. Logically, the hit rate may be greatly affected by the PI width, which, in turn, relates to the confidence level.

$$HitRate = \frac{1}{n} \sum_i h_i, h_i = \begin{cases} 1, & min_i \leq actual_i \leq max_i \\ 0, & actual_i > max_i \vee actual_i < min_i \end{cases} \quad (4.2)$$

where min_i and max_i are the minimum and maximum values, respectively, of the PI for the estimate of task i ; $actual_i$ is the actual effort of the task i ; and n is the number of the estimated task.

Due to the fact that SISE only produces estimates in terms of a prediction interval, the hit rate was chosen as the method for comparison to PROBE. In addition, the PI widths were compared to give perspective to the relative hit rates.

4.5.4 Participants

The students involved in this experiment were, at the time, enrolled in Auburn University’s Software Process course (COMP 5700). The Software Process course is designed to provide: insight into process-oriented software development; exposure to common engineering processes; and experience with a software process [12].

Size Estimate									
Base Parts		Estimated			Actual				
		Base (B)	Deleted (D)	Modified (M)	Added (BA)	Base (B)	Deleted (D)	Modified (M)	Added (BA)
Totals		0	0	0	0	0	0	0	0
Parts Additions		Type	Operations	Estimated Rel Size	Size (PA)	Reusable?	Actual		
main	logic	1	M	9	no	1	7	no	
readFile	I/O	1	L	27	no	1	11	no	
parseFile	I/O	1	M	9	no	1	9	no	
ChiSquared	calculation	5	L	135	no	6	29	no	
Totals				180			56		0
Reused Parts		Estimated Size (R)	Actual Size						
userPromptFileName		6	6						
Totals		6	6						
Calculations		r(estimated, actual)		Size	Time				
Calculation A				0.939	0.999				
Intercept	B0			-47.677	-9.273				
Slope	B1			1.670	2.258				
Calculation B				0.938	0.999				
Intercept	B0			-44.857	-5.632				
Slope	B1			1.630	2.206				
Calculation C				1.176	2.163				
Intercept	B0			0.000	0.000				
Slope	B1			1.176	2.163				
Calculation D				1.650	2.540				
PROBE estimating basis used:	A, B, C, or D			A	B				
Added size (A)	A = BA + PA			180					
Estimated proxy size (E)	E = A + M			180					
Planned added & modified size (P)	P = E * B1size + B0size			253					
Planned total size (T)	T = P + B - D - M + R			259					
Reusable size (NR)				0					
Estimated time	Time = E * B1time + B0time				552				
Lower prediction interval (LPI)	LPI			128	521				
Upper prediction interval (UPI)	UPI			368	555				

Figure 4.5: Sample PROBE calculation using the assignment spreadsheet.

4.5.5 Questions and Presentation

Students preparing a PROBE estimate utilized a custom spreadsheet, which guided and facilitated the model's calculations. Figure 4.5 shows an example.

In Fall 2012 and Spring 2013, the Software Process course had moved beyond the PROBE estimation model and was employing PCSE estimates. In these semesters, prior to the last assignment, each student was provided a summary of his or her actual effort values from past assignments (see Table 4.13). The students were asked to review the summary data and answer the following questions before beginning work on the assignment (approximately a week prior).

Place a check next to each past assignment that you are confident is smaller than CA07 in terms of required effort to complete.

Assignment	Actual Effort	Description
CA06	272 min.	Calculate effort based on historical data.
CA04	298 min.	Extract design components from a file containing Python source code.
CA03	433 min.	Determine the size of a software component relative to a list of components.
CA02	765 min.	Analyze a time log.

Table 4.13: Sample assignment summary.

- CA02
- CA03
- CA04
- CA06

Place a check next to each past assignment that you are confident is larger than CA07 in terms of required effort to complete.

- CA02
- CA03
- CA04
- CA06

Based on the results of these questions, the researchers constructed a SISE-style estimate for each student. The accuracy of the SISE estimates was calculated and compared to the historical accuracy of PROBE estimates.

4.5.6 Results

A total of 406 estimates were constructed using the PROBE model. Of that number, 176 assignments were completed within the estimate's prediction interval and 230 fell outside. The average PI width was 758 minutes.

A total of 77 estimates were constructed using the SISE model. Of that number, 26 assignments were completed within the estimate's prediction interval and 51 fell outside. The average PI width was 292 minutes.

The hit rates were compared using 2-sample test for equality of proportions with continuity correction in R. Output Listing 22 shows the results. The p-value of 0.07535 prevents the rejection of the test's null hypothesis that the SISE proportion is less than the PROBE proportion. Therefore, it cannot be concluded that the accuracy of the PROBE model out-performs the SISE model.

The PI width values for each model were analyzed and determined to follow a non-normal distribution. Therefore, the comparison of the PI widths was conducted using a Mann-Whitney U test (see Output Listing 23). At a 95% confidence interval, the p-value of 0.008988 indicated that the PI width values from the two data sets were significantly different. A second Mann-Whitney U test was conducted (see Output Listing 24) to determine if the PROBE PI widths were significantly larger than the SISE PI widths; the p-value of 0.004494 confirmed they were.

4.5.7 Conclusions

Overall, the analysis revealed that no provable statistical difference existed between the accuracy of the PROBE and SISE models' hit rates. In addition, the SISE model demonstrated significantly smaller PI widths than that of the PROBE model, which may – subjectively – be interpreted as more useful for project planning purposes.

Decision: *Reject the null hypothesis H_0 in favor of the alternate H_a , concluding that the SISE model estimates are equally or more accurate than PROBE.*

4.6 Time Investment and Perceived Value

The next steps in validation of the SISE model involved demonstrating the model's required time investment and perceived value. To accomplish this, an attitudinal survey was

conducted in the Spring 2013 Software Process class, which had constructed estimates using both expert judgment and PCSE in project assignments.

The PCSE model is based on a subset of the activities composing the PROBE model; it follows that the complexity and time investment required to complete a PCSE estimate is equal to or less than that of the PROBE model. Therefore, a comparison of the SISE model to the PCSE model was determined valid in demonstrating the SISE model's complexity and required time investment as (indirectly) compared to the PROBE model.

4.6.1 Hypothesis

The amount of time spent constructing an estimate directly impacts a practitioner's perception of the model's value and may influence the decision to use the model. The direct comparison of SISE time investments to PCSE, which uses a subset of the PROBE model, indirectly demonstrates SISE's relationship to PROBE. To confirm the relative time investment in a SISE estimate is less than that of a PCSE estimate, a survey was constructed to test the following hypothesis:

H_{a3}: An estimator using the SISE model will invest less time in producing an estimate as the time required using PCSE ($timeSISE < timePCSE$).

with the null hypothesis as

H₀₃: An estimator using the SISE model will invest as much or more time in producing an estimate as the time required to use PCSE ($timeSISE \geq timePCSE$).

In addition, the perceived value of an estimate may influence a practitioner's usage of a particular model or approach. A comparison of the value of a SISE estimate to guesswork, a commonly used approach, demonstrates a significant factor in adoption. To confirm that a software engineer's perception of value for a SISE estimate is greater than that of guesswork, a survey was constructed to test the following hypothesis:

Ha₄: An estimator introduced to the SISE model and underlying approach will perceive the output of the model as equally or more useful than guesswork (valueSISE \geq valueGuess).

with the null hypothesis as

H₀₄: An estimator introduced to the SISE model and underlying approach will perceive the output of the model as less useful than guesswork (valueSISE < valueGuess).

4.6.2 Survey

A survey was conducted of software engineering students to determine their attitudes and opinions regarding the relationships between SISE, PCSE, and guesswork with respect to their relative time investment and expected value. The survey presented twenty-six questions covering PCSE and SISE comprehension; model usage as compared to risk; and time investment, complexity, and value comparisons.

4.6.3 Metrics

The answers to the survey questions followed either a Likert scale or categorical model. The scaled responses allowed the student to choose along a rating scale such as “much more,” “somewhat more,” “same,” “somewhat less,” and “much less.” The categorical responses typically followed a pattern of “choice 1,” “choice 2,” or “neither.”

Once the survey responses were tabulated, a proportion was assigned to each response. The responses were analyzed to determine if the number of people choosing a particular response was significantly greater than the others by calculating a 1-sample proportions test with continuity correction in R. In addition, the scaled responses were categorized into more general “agree,” “disagree,” or “neither” values to determine if a particular category demonstrated a significantly greater proportion than the others.

4.6.4 Participants

The students involved in this experiment were, at the time, enrolled in Auburn University's Software Process course (COMP 5700). The Software Process course is designed to provide: insight into process-oriented software development; exposure to common engineering processes; and experience with a software process [12].

4.6.5 Questions and Presentation

Appendix D – Attitudinal Survey Questions contains the complete list of survey questions. The questions relevant to this research include: contrasting the *time investment* required for SISE to that of PCSE and contrasting the *perceived value* of a SISE estimate to that of expert judgment.

21. The PCSE and SISE effort estimation models take two distinct approaches to constructing an estimate.

(Descriptions of models omitted for brevity.)

*Based on the descriptions of each model, select the statement below that best described your impression of the PCSE model as compared to the SISE model in terms of **time investment**.*

I believe the PCSE model would require a larger time investment than the SISE model.

I believe neither model is more time-consuming to utilize than the other.

I believe the SISE model would require a larger time investment than the PCSE model.

*26. Based on the description of each the SISE model and your experience with expert judgement (i.e. guesswork), select the statement below that best described your impression of the SISE model as compared to expert judgement in terms of the **value** of the estimate produced.*

I believe the SISE model will produce much more valuable estimates as compared to expert judgement.

I believe the SISE model will produce somewhat more valuable estimates as compared to expert judgement.

I believe the SISE model will produce estimate of equal value as compared to expert judgement.

I believe the SISE model will produce somewhat less valuable estimates as compared to expert judgement.

I believe the SISE model will produce much less valuable estimates as compared to expert judgement.

In addition, students were asked to provide their opinion of the SISE and PCSE models' relative complexities.

*22. Based on the descriptions of each model, select the statement below that best described your impression of the PCSE model as compared to the SISE model in terms of **complexity**.*

I believe the PCSE model is much more complex as compared to the SISE model.

I believe the PCSE model is somewhat more complex as compared to the SISE model.

I believe the PCSE model is the same complexity as the SISE model.

I believe the PCSE model is somewhat less complex as compared to the SISE model.

I believe the PCSE model is much less complex as compared to the SISE model.

The survey was made available to the students near the end of the semester and participation was voluntary.

Reason	Count	Prop.
Time Investment		
PCSE requires a larger time investment than SISE	31	88.6%
Neither model is more time consuming	3	8.6
PCSE requires a smaller time investment than SISE	1	2.9
Total	35	
Perceived Value		
SISE much more than expert judgment	18	51.4%
SISE somewhat more than expert judgment	16	45.7
No difference	1	2.9
SISE somewhat less than expert judgment	0	0.0
SISE much less than expert judgment	0	0.0
Total	35	
Perceived Complexity		
PCSE much more than SISE	20	57.1%
PCSE somewhat more than SISE	13	37.1
No difference	0	0.0
PCSE somewhat less than SISE	2	5.7
PCSE much less than SISE	0	0.0
Total	35	

Table 4.14: Summary of survey results.

4.6.6 Results

A total of 35 responses to the survey were received. Table 4.14 lists the questions, the responses, the count of each response, and the relative proportion.

In terms of time investment, the responses demonstrate that 31 out of 35 students believe the PCSE model requires a larger time investment as compared to the SISE model, based on the provided descriptions. The 95% confidence interval for this proportion is 72.3-96.2% (see Output Listing 25).

In terms of perceived value, the responses demonstrate that 34 out of 35 students believe the SISE model provides greater value – either “much more” or “somewhat more” – as compared to expert judgment, based on the provided descriptions. The 95% confidence interval for this proportion is 83.4-99.9% (see Output Listing 26).

In terms of complexity, the responses demonstrate that 33 out of 35 students believe the PCSE model is more complex – either “much more” or “somewhat more” – as compared to the SISE model, based on the provided descriptions. The 95% confidence interval for this proportion is 79.5-99.0% (see Output Listing 26).

4.6.7 Conclusions

In summary, a significant proportion of the respondents indicated their belief that SISE requires a smaller time investment as compared to PCSE; SISE provides a higher level of perceived value as compared to expert judgment; and PCSE is a more complex model as compared to SISE.

Decision: *Reject the null hypothesis $H0_3$ in favor of the alternate Ha_3 , concluding that an estimator using the SISE model will invest less time in producing an estimate as required using PCSE.*

Decision: *Reject the null hypothesis $H0_4$ in favor of the alternate Ha_4 , concluding that an estimator introduced to the SISE model and underlying approach will perceive the output of the model as equally or more useful than guesswork.*

4.7 Summary

This work addresses the viability of the SISE estimation model as a reasonable option for individuals wishing to construct effort estimates. Specifically, SISE attempts to fill the gap between expert judgment (guesswork) and other individual estimation models, such as PROBE.

It has been demonstrated that software engineers are generally capable of identifying the larger of two tasks. In terms of estimation accuracy, this research had demonstrated that the SISE model is no more or less useful than PROBE, its nearest, validated competitor. In fact, this research has demonstrated that the SISE model, with no provable difference in accuracy, produces estimates with a narrower, and arguably more useful, prediction interval.

This research has also demonstrated that the SISE model is perceived as less complex and less time consuming than the PCSE model, and by relative comparison, the PROBE model. Lastly, it has been demonstrated that software engineers view the output of the SISE model as more valuable than that of expert judgment, a factor that may influence adoption and continued usage.

Obviously, further research is required to determine the extent to which these results apply to an industrial environment. Furthermore, additional research into the benefits and effectiveness of the SISE model should be conducted to determine how the model behaves when calibrated with larger, more extensive historical data sets. Lastly, the relative sizing survey may produce different and more interesting results if it is modified to allow the respondents to specify a third option of “unknown,” for the relative size differences.

Chapter 5

Conclusions and Additional Research

5.1 Summary

The software engineering discipline is filled with many varied examples of software process methods and tools focused on the team or organization. In recent years, the agile approach to software engineering has increased the focus of software process on small teams and individuals; however, not all aspects of software process have been deeply or fully addressed.

The majority of effort estimation models – traditional and agile – focus on teams or groups of software engineers. The discipline is ripe with various examples of team-based models including Wideband Delphi, Planning Poker, function point analysis, COCOMO, etc. The few examples of effort estimation models focused on the lone software engineer are limited to tradition mathematical models with (relatively) substantial complexity and required time investment. The discipline lacks a truly agile model based on a minimal combination of empirical data and expert judgment.

The SISE model under development at Auburn University’s microISV Research Lab is a simple-to-understand, lightweight, and agile effort estimation model that specifically targets individual software engineers. SISE combines an individual’s personal, empirical data with expert judgment and experiences to produce relatively accurate estimates with a minimal investment of training and time.

The SISE model rests on two foundational principles. First, software engineers are capable of identifying the largest of a pair of tasks based solely on their descriptions. Second, a software engineer who is presented with a future work activity is capable of identifying

two historical tasks – one larger, one smaller – which may serve as a prediction of the future activity’s size.

The name “SISE” is an acronym for the model’s four basic steps: Sort, Identify, Size, and Evaluate. The first step – Sort – involves the ordering of historical data by the actual effort required to complete the activity. The second step – Identify – involves choosing two tasks from the historical data set: one confidently known to be smaller, one confidently known to be larger, and both relatively close in size to the future work. Once the practitioner has chosen a pair of tasks, the third step – Size – produces a rough prediction interval of the future activity’s size using the actual effort values for the two completed tasks. The final step – Evaluate – involves shifting or resizing the prediction interval to account for any historical bias. This last step is optional and is only applied if the estimator is dissatisfied with the precision, accuracy, or confidence level of his or her estimate.

Validation of the SISE model included two major steps. First, the foundational principle that relative tasks sizing by software engineers is suitably accurate was validated. The validation occurred in the form of a survey, presented to over 100 software engineering students, which presented the respondents with a series of task pairs from which they were to identify the larger. Some of the pairs had a known, verifiable size difference based on ten years of time logs provided by students in the Software Process course, while some of the pairs did not. The results indicated that, on average, a majority of software engineers were able to identify the larger task, while not typically misidentifying the smaller. When presented with tasks demonstrating no significant difference in size, the respondents were typically swayed by the wording, format, or word count.

The second phase of validation involved a series of Software Process students who were asked to identify where a future activity should be placed in the ordered list of their completed tasks. In addition, the students were asked to construct a PCSE estimate. The results indicated that SISE predictions were no more or less accurate than the PCSE model’s estimates. In addition, the students indicated that SISE, in their opinion, took less time and

was based on less a complex model. In summary, SISE appears capable of producing results of equal quality, in less time, and with less training.

5.2 Conclusions

Several conclusions may be drawn from the results of this research. First, it should be noted that a lightweight, agile effort estimation mode – SISE – has been proven effective as a tool for individual software engineers. From this, several other conclusions may be drawn. For example, this research has reinforced the notion that effort estimation, in general, does not need to be a heavy weight activity. Approaches such as Planning Poker and SISE provide valuable results with minimal cost. In fact, it is possible that many data-driven activities within the software engineer discipline may benefit from a lightweight version based on expert judgment and backed by empirical data, in much the same way SISE is built. In other words, new models may be constructed to build upon the intuitive knowledge and experience of software engineers while grounding the activities in solid, fact-based data.

Lastly, this research has demonstrated that individual software engineers possess skillsets that are unpredictable and likely very difficult to quantify. Nowhere has this been more apparent than in this research’s attempts to identify “common” rank orderings of tasks by effort. Actual effort values demonstrated that one person may struggle on a simple task, whereas another person may finish quickly. In fact, the wide variety of responses from students on why they believed one task might be larger than another demonstrated a plethora of subtle experiences, skills, and talents, each capable of affecting personal productivity in significant ways.

5.3 Additional Research

Building upon the conclusions drawn from this research, the authors have identified several topics of future of research.

Individual software developers – whether they are labeled as team members, consultants, or micropreneurs – represent fertile ground for future research activities. A wide variety of tools exist for facilitating team software development activities; however, the tools and techniques specific to the lone software engineer are few and far between. For example, the study of effort estimation approaches and models has been underway for decades, however, the software engineering industry still lacks for agile, reasonably accurate tools for individuals to size their own personal work efforts.

Another obvious area of research remaining is the method or methods by which SISE may be integrated into a team environment. Team integration should include a basic methodology for combining individual and team estimates, calculating and incorporating overhead costs, and creating synergy between team members.

Once the role and behaviors of SISE model have been defined with the larger context of a team environment, a sample implementation plan must be formulated. The implementation plan will cover the basics steps in introducing SISE into a team environment, training the participants, equipping team members with appropriate tools, measuring the model's effectiveness, and adjusting for quality, as necessary.

A key factor in support of the SISE model will be development of supporting tools. Although the use of tools is not mandatory for the successful use of the SISE model, several areas of the model may benefit from their creation. Such areas include data pruning, historical data management, and relative sizing. For example, a tool supporting the data pruning process, based on a variety of algorithms, would reduce the preparation time for each estimate. A data repository for historical activities – descriptions, dates, estimated effort, and actual effort values – would streamline the data pruning process. Lastly, a relative sizing tool – designed to present a list of activities and assist in the process of inserting a new, future task into the list – would be useful in both training new estimators and assisting experienced ones. As such tools are developed, methods must be developed for integrating these tools

into a team environment (e.g. integration with time tracking tools). Such integration will encourage SISE adoption into team-based development environments.

Another area of research can be found in the development of pruning algorithms to support the SISE model. Such research would be based in the exploration of the factors involved in the pruning of historical data (e.g. age of activities, sizes of activities, data distribution types, etc.). Depending on the factors involved, algorithms may be developed to identify and remove extraneous historical data, while leaving sufficient data to establish a reliable estimate within the desired confidence levels.

In instances where the SISE model is utilized for long periods of time, research should be undertaken to determine if the model is self-correcting, or if a combination of pruning and output tuning are necessary. It is possible that the model will not require specific actions, other than historical pruning, to maintain an acceptable level of quality. However, it may also be possible that the practitioner will eventually need to tune the estimates based on historical performance. Additional research into such feedback mechanisms may benefit the overall quality of the model.

During the process of gathering and analyzing data related to relative task sizing, a moderate-to-strong correlation was noted between individual students and groups, such as a class. Additional research into such correlations may provide insight into task sizing in general. Questions that may be explored include: How strong are these correlations? How homogeneous must the groups be to maintain a strong correlation to the individuals' sizings? Can relative sizing techniques be combined with rank correlation techniques to enhance existing estimation models?

Surprisingly, this research noted that early estimates, within the context of the software process course, tended to outperform formal estimation models, such as PCSE and SISE. While this may be due to a lack of data to calibrate the models, further research into the relationship between expert judgment, formal models, and data set sizes may reveal some interesting trends that can be used to improve existing estimation approaches.

One last area of potential research, beyond the “software engineering” focused practices, is the underlying psychological factors involved in the use of the SISE model (and sizing in general). For example, research indicates that word choice, word count, and grammatical structure during requirements definition affects a reader’s perception of complexity and required effort. In addition, the attitudinal survey revealed a tendency, on the part of software engineers, to equate higher complexity with more value, which may not always be the case. Techniques for identifying and addressing these psychological influences would positively affect both the implementation of the SISE model and other estimation models.

Bibliography

- [1] AdMob Metrics. July 2009 metrics report. <http://metrics.admob.com/2009/08/july-2009-metrics-report/>, August 2011. Accessed August 14, 2011.
- [2] Alan Albrecht. Function points: A new way of looking at tools. IBM, 1979.
- [3] Android Market. Android market developer signup. <https://market.android.com/publish/signup/>, August 2011. Accessed August 14, 2011.
- [4] Apple. Apple Developer Programs 2011. <http://developer.apple.com/programs/>, August 2011. Accessed August 14, 2011.
- [5] Association of Software Professionals. ASP member forum. <http://members.asp-software.org/newsgroups/showthread.php?t=25806>, August 2011. Accessed August 14, 2011.
- [6] Auburn University. COMP 2210 course description. http://www.eng.auburn.edu/files/acad_depts/csse/syllabi/comp2210.pdf, 2013. Accessed May 13, 2013.
- [7] Barry W. Boehm. Software engineering economics. *Software Engineering, IEEE Transactions on*, SE-10(1):4–21, jan. 1984.
- [8] Barry W. Boehm, Chris Abts, A Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece. *Software Cost Estimation with Cocomo II with CD-ROM*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 2000.
- [9] M. Cohn. *Succeeding with Agile: Software Development Using Scrum*. Addison-Wesley signature series. Addison-Wesley, 2009.
- [10] Construx. Software development practices. <http://www.construx.com/Page.aspx?nid=68>, 2012. Accessed January 26, 2012.
- [11] S. D. Conte, H. E. Dunsmore, and V. Y. Shen. *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA, 1986.
- [12] David Umphress. COMP 5700/6700/6706 software process. <http://www.eng.auburn.edu/users/umphress/comp6700/index.html>, 2013. Accessed May 13, 2013.
- [13] Iris Fabiana de Barcelos Tronto, José Demisio Simões da Silva, and Nilson Sant Anna. Comparison of artificial neural network and regression models in software effort estimation. In *IJCNN*, pages 771–776, Brazil, 2007. IEEE.

- [14] Gartner Incorporated. Gartner says worldwide software as a service revenue is forecast to grow 21 percent in 2011. <http://www.gartner.com/it/page.jsp?id=1739214>, August 2011. Accessed August 14, 2011.
- [15] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Ltd, Amsterdam, May 1977.
- [16] M. Host and C. Wohlin. An experimental study of individual subjective effort estimations and combinations of the estimates. In *Software Engineering, 1998. Proceedings of the 1998 International Conference on*, pages 332–339, Sweden, apr 1998. IEEE, IEEE.
- [17] Watts S. Humphrey. *A Discipline for Software Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [18] W.S. Humphrey. *Introduction to the team software process(sm)*. SEI series in software engineering. Addison-Wesley, 2000.
- [19] D.R. Jeffery and G. Low. Calibrating estimation tools for software development. *Software Engineering Journal*, 5(4):215–221, jul 1990.
- [20] Philip M. Johnson and Anne M. Disney. The personal software process: A cautionary case study. *IEEE Software*, 15(6):85–88, November 1998.
- [21] M. Jorgensen, B. Boehm, and S. Rifkin. Software development effort estimation: Formal models or expert judgment? *Software, IEEE*, 26(2):14–19, march-april 2009.
- [22] M. Jorgensen and M. Shepperd. A systematic review of software development cost estimation studies. *Software Engineering, IEEE Transactions on*, 33(1):33–53, jan. 2007.
- [23] M Jorgensen, K H Teigen, and K J Molokken-Ostvold. Better sure than safe? overconfidence in judgment based software development effort prediction intervals. *Journal of Systems and Software*, 70(1-2):79–93, 2004.
- [24] Magne Jørgensen. A critique of how we measure and interpret the accuracy of software development effort estimation. In Jacky Keung, editor, *1st International Workshop on Software Productivity Analysis and Cost Estimation*, pages 15–22, Tokyo, Japan, 2007. Information Processing Society of Japan.
- [25] Chris F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30(5):416–429, May 1987.
- [26] Steve McConnell. *Software Estimation: Demystifying the Black Art (Best Practices (Microsoft))*. Microsoft Press, Redmond, WA, USA, kindle edition edition, 2006.
- [27] Mirriam-Webster. Mirriam-webster dictionary. <http://www.merriam-webster.com/>, 2012. Accessed January 26, 2012.
- [28] J.J. Moder, C.R. Phillips, and E.W. Davis. *Project management with CPM, PERT, and precedence diagramming*. Van Nostrand Reinhold, New York, NY, USA, 1983.

- [29] Mountain Goat Software. Planning poker cards. <http://store.mountaingoatsoftware.com>, 2013. Accessed May 30, 2013.
- [30] Tridas Mukhopadhyay, Steven S. Vicinanza, and Michael J. Prietula. Examining the feasibility of a case-based reasoning model for software effort estimation. *MIS Q.*, 16:155–171, June 1992.
- [31] L. H. Putnam. A general empirical solution to the macro software sizing and estimating problem. *IEEE Trans. Softw. Eng.*, 4:345–361, July 1978.
- [32] M. Ruhe, R. Jeffery, and I. Wiczorek. Using web objects for estimating software development effort for web applications. In *Software Metrics Symposium, 2003. Proceedings. Ninth International*, pages 30 – 37, sept. 2003.
- [33] Melanie Ruhe, Ross Jeffery, and Isabella Wiczorek. Cost estimation for web applications. In *Proceedings of the 25th International Conference on Software Engineering, ICSE '03*, pages 285–294, Washington, DC, USA, 2003. IEEE Computer Society.
- [34] R. Schoedel. *PROxy Based Estimation (PROBE) for Structured Query Language (SQL)*. Technical note. Carnegie Mellon University, Software Engineering Institute, 2006.
- [35] Scrum Methodology. Scrum effort estimation and story points. <http://scrummethodology.com/scrum-effort-estimation-and-story-points>, 2008. Access January 30, 2012.
- [36] Martin Shepperd, Chris Schofield, and Barbara Kitchenham. Effort estimation using analogy. In *Proceedings of the 18th international conference on Software engineering, ICSE '96*, pages 170–178, Washington, DC, USA, 1996. IEEE Computer Society.
- [37] Standish Group. Standish chaos report 2009. <https://secure.standishgroup.com/reports/reports.php>, August 2011. Accessed August 14, 2011.
- [38] Russell Thackston and David Umphress. Individual effort estimating: Not just for teams anymore. *CrossTalk: The Journal of Defense Software Engineering*, 25(3):4–7, May/June 2012.
- [39] Russell Thackston and David Umphress. Micropreneurs: The rise of the microisv. *IT Professional*, 15(2):50–56, 2013.
- [40] David Umphress. Principle-centered software engineering. <http://swemac.cse.eng.auburn.edu/~umphrda/PCSE>, October 2011. Accessed October 12, 2011.
- [41] M. van Genuchten. Why is software late? an empirical study of reasons for delay in software development. *Software Engineering, IEEE Transactions on*, 17(6):582 –590, jun 1991.
- [42] Fiona Walkerden and Ross Jeffery. An empirical study of analogy-based software effort estimation. *Empirical Softw. Engg.*, 4:135–158, June 1999.

- [43] Gerhard E. Wittig and Gavin R. Finnie. Using artificial neural networks and function points to estimate 4gl software development effort. *Australasian J. of Inf. Systems*, 1(2):87–94, 1994.
- [44] S. Yenduri, S. Munagala, and L.A. Perkins. Estimation practices efficiencies: A case study. In *Information Technology, (ICIT 2007). 10th International Conference on*, pages 185 –189, dec. 2007.

Appendices

Appendix A

Output Listings

The following output listings were produced using R. Listings 1 - 10 compare the average construction times for assignment pairs used in the relative sizing survey. Listings 11 - 20 compare the proportion of correct answers against the probability of random chance (proportion = 0.5). Output Listing 21 compares the proportion of correct survey answers to the total number of answers. Output Listing 22 compares the hit rates of SISE and PROBE via the number of hits for each model versus the total number of assignments. Output Listings 23 and 24 compare the PROBE and SISE prediction interval widths. Output Listings 25 - 27 compare the respondents' perceptions of time investment, value, and complexity for SISE, PCSE, and expert judgment.

```
> wilcox.test(TDist[,1], MPPS[,1], conf.int="T" alternative="g")
Wilcoxon rank sum test with continuity correction

data:  TDist[, 1] and MPPS[, 1]
W = 9923.5, p-value = 1.007e-09
alternative hypothesis: true location shift is greater than 0
95 percent confidence interval:
 75.00005 Inf
sample estimates:
difference in location
105.0001
```

(1)

```
> wilcox.test(CriticalPath[,1], T-Dist[,1], conf.int="T"
alternative="g")
Wilcoxon rank sum test with continuity correction
data: CriticalPath[, 1] and T-Dist[, 1]
W = 19022.5, p-value = 0.0001667
alternative hypothesis: true location shift is greater than 0 (2)
95 percent confidence interval:
36.99997 Inf
sample estimates:
difference in location
66.99999
```

```
> wilcox.test(TDist[,1], FiveSlot[,1], conf.int="T"
alternative="g")
Wilcoxon rank sum test with continuity correction
data: TDist[, 1] and FiveSlot[, 1]
W = 4701, p-value = 0.01516
alternative hypothesis: true location shift is greater than 0 (3)
95 percent confidence interval:
14.99994 Inf
sample estimates:
difference in location
62.00005
```

```
> wilcox.test(Text[,1], MPPS[,1], conf.int="T" alternative="g")
Wilcoxon rank sum test with continuity correction
data: Text[, 1] and MPPS[, 1]
W = 1133, p-value = 0.004951
alternative hypothesis: true location shift is greater than 0 (4)
95 percent confidence interval:
15.99999 Inf
sample estimates:
difference in location
47.40739
```

```
> wilcox.test(TDist2[,1], CalcCorr[,1], conf.int="T"
alternative="g")
Wilcoxon rank sum test with continuity correction
data:  TDist2[, 1] and CalcCorr[, 1]
W = 451, p-value = 0.06837
alternative hypothesis: true location shift is greater than 0 (5)
95 percent confidence interval:
-5.000015 Inf
sample estimates:
difference in location
45.99993
```

```
> wilcox.test(FiveSlot[,1], MPPS[,1], conf.int="T"
alternative="g")
Wilcoxon rank sum test with continuity correction
data:  FiveSlot[, 1] and MPPS[, 1]
W = 764, p-value = 0.05171
alternative hypothesis: true location shift is greater than 0 (6)
95 percent confidence interval:
-5.4093e-05 Inf
sample estimates:
difference in location
43.78432
```

```
> wilcox.test(ComponentInfo[,1], FiveSlot[,1], conf.int="T"
alternative="g")
Wilcoxon rank sum test with continuity correction
data: ComponentInfo[, 1] and FiveSlot[, 1]
W = 397, p-value = 0.1399
alternative hypothesis: true location shift is greater than 0 (7)
95 percent confidence interval:
-33.99998 Inf
sample estimates:
difference in location
37.99995
```

```
> wilcox.test(Text[,1], AMS[,1], conf.int="T" alternative="g")
Wilcoxon rank sum test with continuity correction
data: Text[, 1] and AMS[, 1]
W = 1739.5, p-value = 0.1012
alternative hypothesis: true location shift is greater than 0 (8)
95 percent confidence interval:
-6.000062 Inf
sample estimates:
difference in location
22.00008
```

```
> wilcox.test(Text[,1], CalcCorr[,1], conf.int="T" alternative="g")
Wilcoxon rank sum test with continuity correction
data: Text[, 1] and CalcCorr[, 1]
W = 570.5, p-value = 0.1689
alternative hypothesis: true location shift is greater than 0 (9)
95 percent confidence interval:
-20.00005 Inf
sample estimates:
difference in location
21.00005
```

```

> wilcox.test(AMS[,1], MPPS[,1], conf.int="T" alternative="g")
Wilcoxon rank sum test with continuity correction
data:  AMS[, 1] and MPPS[, 1]
W = 2184, p-value = 0.06977
alternative hypothesis: true location shift is greater than 0
95 percent confidence interval:
-2.999972 Inf
sample estimates:
difference in location
26.99997

```

(10)

```

M-P-P-S vs. T-Dist
> prop.test(95,113, alternative="g")
1-sample proportions test with continuity correction
data:  95 out of 113, null probability 0.5
X-squared = 51.115, df = 1, p-value = 4.355e-13
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
0.7712947 1.0000000
sample estimates:
p
0.840708

```

(11)

```

T-Dist vs. CriticalPath
> prop.test(80,113, alternative="g")
1-sample proportions test with continuity correction
data:  80 out of 113, null probability 0.5
X-squared = 18.7257, df = 1, p-value = 7.547e-06
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
0.6287827 1.0000000
sample estimates:
p
0.7079646

```

(12)

5-Slot vs. T-Dist

```
> prop.test(56,113, alternative="g")
1-sample proportions test with continuity correction
data: 56 out of 113, null probability 0.5
X-squared = 0, df = 1, p-value = 0.5
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
0.4149116 1.0000000
sample estimates:
p
0.4955752
```

(13)

M-P-P-S vs. Text

```
> prop.test(71,113, alternative="g")
1-sample proportions test with continuity correction
data: 71 out of 113, null probability 0.5
X-squared = 6.9381, df = 1, p-value = 0.004219
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
0.546867 1.000000
sample estimates:
p
0.6283186
```

(14)

CalcCorr vs. T-Dist2

```
> prop.test(51,113)
1-sample proportions test with continuity correction
data: 51 out of 113, null probability 0.5
X-squared = 0.885, df = 1, p-value = 0.3468
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
0.3584833 0.5475238
sample estimates:
p
0.4513274
```

(15)

M-P-P-S vs. 5-Slot

```
> prop.test(91,113)
```

1-sample proportions test with continuity correction

data: 91 out of 113, null probability 0.5

X-squared = 40.9204, df = 1, p-value = 1.586e-10

alternative hypothesis: true p is not equal to 0.5

(16)

95 percent confidence interval:

0.7179120 0.8714467

sample estimates:

p

0.8053097

ComponentInfo vs. 5-Slot

```
> prop.test(63,112)
```

1-sample proportions test with continuity correction

data: 63 out of 112, null probability 0.5

X-squared = 1.5089, df = 1, p-value = 0.2193

alternative hypothesis: true p is not equal to 0.5

(17)

95 percent confidence interval:

0.4656548 0.6550020

sample estimates:

p

0.5625

Text vs. A-M-S

```
> prop.test(76,113)
```

1-sample proportions test with continuity correction

data: 76 out of 113, null probability 0.5

X-squared = 12.7788, df = 1, p-value = 0.0003506

alternative hypothesis: true p is not equal to 0.5

(18)

95 percent confidence interval:

0.5770539 0.7561623

sample estimates:

p

0.6725664

CalcCorr vs. Text

```
> prop.test(77,113)
```

1-sample proportions test with continuity correction

data: 77 out of 113, null probability 0.5

X-squared = 14.1593, df = 1, p-value = 0.000168

alternative hypothesis: true p is not equal to 0.5

(19)

95 percent confidence interval:

0.5861820 0.7641181

sample estimates:

p

0.6814159

M-P-P-S vs. A-M-S

```
> prop.test(107,113)
```

1-sample proportions test with continuity correction

data: 107 out of 113, null probability 0.5

X-squared = 88.4956, df = 1, p-value < 2.2e-16

alternative hypothesis: true p is not equal to 0.5

(20)

95 percent confidence interval:

0.8832757 0.9782325

sample estimates:

p

0.9469027

Proportion of respondents answering correctly (Tasks 1-4)

```
> prop.test(72,113,alternative="g")
1-sample proportions test with continuity correction
data: 72 out of 113, null probability 0.5
X-squared = 7.9646, df = 1, p-value = 0.002385
alternative hypothesis: true p is greater than 0.5 (21)
95 percent confidence interval:
0.5558603 1.0000000
sample estimates:
p
0.6371681
```

```
> hitMiss <- matrix(c(26,176,51,230), ncol=2)
> colnames(hitMiss) <- c('Hit','Miss')
> rownames(hitMiss) <- c('SISE','PROBE')
> hitMiss
Hit Miss
SISE 26 51
PROBE 176 230
> prop.test(hitMiss, alternative="l")
2-sample test for equality of proportions with continuity correction (22)
data: hitMiss
X-squared = 2.0652, df = 1, p-value = 0.07535
alternative hypothesis: less
95 percent confidence interval:
-1.000000000 0.009330862
sample estimates:
prop 1 prop 2
0.3376623 0.4334975
```

```

> wilcox.test(probePI[,1],sisePI[,1], conf.int="T")
Wilcoxon rank sum test with continuity correction
data:  probePI[, 1] and sisePI[, 1]
W = 14635, p-value = 0.008988
alternative hypothesis:  true location shift is not equal to 0
95 percent confidence interval:
18.00009 162.00005
sample estimates:
difference in location
78.00007

```

(23)

```

> wilcox.test(probePI[,1],sisePI[,1], conf.int="T" alternative="g")
Wilcoxon rank sum test with continuity correction
data:  probePI[, 1] and sisePI[, 1]
W = 14635, p-value = 0.004494
alternative hypothesis:  true location shift is greater than 0
95 percent confidence interval:
26.00004 Inf
sample estimates:
difference in location
78.00007

```

(24)

```

> prop.test(31,35)
1-sample proportions test with continuity correction
data:  31 out of 35, null probability 0.5
X-squared = 19.3143, df = 1, p-value = 1.109e-05
alternative hypothesis:  true p is not equal to 0.5
95 percent confidence interval:
0.7232023 0.9627436
sample estimates:
p
0.8857143

```

(25)

```
> prop.test(34,35)
1-sample proportions test with continuity correction
data: 34 out of 35, null probability 0.5
X-squared = 29.2571, df = 1, p-value = 6.338e-08
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
0.8338216 0.9985068
sample estimates:
p
0.9714286
```

(26)

```
> prop.test(33,35)
1-sample proportions test with continuity correction
data: 33 out of 35, null probability 0.5
X-squared = 25.7143, df = 1, p-value = 3.959e-07
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
0.7947722 0.9900403
sample estimates:
p
0.9428571
```

(27)

Appendix B

Themes in Relative Sizing Rationale

In performing a linguistic analysis of each respondent's rationale for selecting a particular task as larger, the following common themes were detected.

Data – Referenced specific data structures, specific data types, or problems inherent in dealing with a particular data set. *“Again, task number two only needs to have values dumped into an array while task number one will need to process an indeterminate amount of data.”*

Familiarity – Referenced prior experience with the problem space. *“I have done a project using this task and it seems more involved that basically an array with basic math applied to it. (Task) 2 would require less time to start/understand.”*

File – Referenced the ease or difficulties involved with file I/O, processing file contents, etc. *“Sometimes unexpected errors can occur when scanning text files, not to mention task number one is very simple.”*

Math – Referenced either the ease or difficulty of creating math-based software or algorithms. *“From my experience with math a t-distribution is more complicated to calculate.”*

Methods – Referenced the number of methods/functions, amount of code or lines of code required to complete the assignment, number of operations to be performed by the software, steps involved in the algorithms, etc. *“Requires more equations to implement which means more functions, more logic to code.”*

Planning – Referenced the amount of time that would be spent in planning and/or design. *“It requires more of analysis and design work before coding.”*

Problem – Directly referenced actual aspects of the problem space as defined by the assignment. *“Both require degrees of freedom and probability / x , but task number two also requires number of tails functionality to be implemented.”*

Reuse – Referenced either the ability to reuse code the respondent has already written, reuse code within the assignment, or standard libraries available in the programming language. *“Dependent on which language you wrote in, but task 1 is accomplished with built-in packages- which makes it quicker to finish.”*

Simplicity – Referenced either the simplicity of the smaller assignment or the complexity of the larger assignment. Based on the general tone of the responses, this theme appears most closely linked with a non-empirical, unstructured, and intuitive guess. *“Task 2 is more complex and will require more effort to implement.”*

Testing – Referenced the ease or difficulty of testing the assignment. *“Writing test code that would cover this code would be quite extensive.”*

Text – Referenced the ease or difficulty inherent in dealing with text-based problems. *“In my experience, parsing of English/text documents and has always led to more work, task 2 includes parsing, but it is less complex and mathematics usually are too hard to do.”*

Appendix C
Relative Sizing Survey Questions

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write software to calculate the median, product, penultimate, and standard deviation of a list of numbers.

The software should consist of four standalone functions that follow the specification below.

- median(list) --> numeric
Returns the median of a list of numeric values. The median for an even number of items is the average of the two midpoint values when the items are arranged in sorted order; the median for an odd number of items is the middle value when the items are arranged in sorted order. Raises an error if the list is empty.
- product(list) --> numeric
Returns the product of the values in the list. Raises an error if the list is empty.
- penultimate (list) --> numeric
Returns the second largest value of a list of numeric values. Raises an error if the list contains one or fewer elements.
- stdev(list) --> numeric
Returns the standard deviation of a list of numeric values using the formula below. Raises an error if the list contains one or fewer elements.

$$\sigma(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{avg})^2}{n - 1}}$$

where $\sigma(x_1, \dots, x_n)$ is the standard deviation of the x values and x_{avg} is the average of these n values.

Task 2

Write a class to calculate the probability value of a t-distribution given the number of tails and the values of t and n (i.e., write a program to find the area under the t-distribution).

The software should follow an object-oriented design and implement the specification below for the class named *TCurve*.

- *TCurve*(n) --> *TCurve*
Constructor. Instantiates an instance of a t-distribution with n degrees of freedom. Raises the predefined "ValueError" exception if n is invalid. The exception has the description "Non-integer n " if n not an integer; "Invalid value of n " if $2 < n > 50$; and "Missing n " if n is omitted.
- *myTCurve.p*(t , tails) -> float
Finds the area under the t-curve, given a value of t and the number of tails. Raises the "ValueError" exception with a string of "Invalid t " if t is missing, not numeric, or less than zero; raises the "ValueError" exception with a string of "Invalid tails" if the number of tails is not 1 or 2.
- *myTCurve.n*() -> integer
Returns the number of degrees of freedom of the t-curve.

The formula for integrating a T-Distribution can be found on Attachment 1.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

A

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write software to calculate the average, median, and standard deviation of a list of numbers.

The software should consist of four standalone functions that follow the specification below.

- `average(list) --> numeric`
Returns the average of a list of numeric values. Raises an error if the list is empty.
- `median(list) --> numeric`
Returns the median of a list of numeric values. The median for an even number of items is the average of the two midpoint values when the items are arranged in sorted order; the median for an odd number of items is the middle value when the items are arranged in sorted order. Raises an error if the list is empty.
- `stdev(list) --> numeric`
Returns the standard deviation of a list of numeric values using the formula below. Raises an error if the list contains one or fewer elements.

$$\sigma(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{avg})^2}{n - 1}}$$

where $\sigma(x_1, \dots, x_n)$ is the standard deviation of the x values and x_{avg} is the average of these n values.

Task 2

Write software to calculate the median, product, penultimate, and standard deviation of a list of numbers.

The software should consist of four standalone functions that follow the specification below.

- `median(list) --> numeric`
Returns the median of a list of numeric values. The median for an even number of items is the average of the two midpoint values when the items are arranged in sorted order; the median for an odd number of items is the middle value when the items are arranged in sorted order. Raises an error if the list is empty.
- `product(list) --> numeric`
Returns the product of the values in the list. Raises an error if the list is empty.
- `penultimate(list) --> numeric`
Returns the second largest value of a list of numeric values. Raises an error if the list contains one or fewer elements.
- `stdev(list) --> numeric`
Returns the standard deviation of a list of numeric values using the formula below. Raises an error if the list contains one or fewer elements.

$$\sigma(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{avg})^2}{n - 1}}$$

where $\sigma(x_1, \dots, x_n)$ is the standard deviation of the x values and x_{avg} is the average of these n values.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

C

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write a program, which reads a text file and displays the name of the file as well as the total number of words, sentences, and lines in the file.

The program should be capable of sequentially processing multiple files by repeatedly prompting the user for file names until the user enters "stop". The program should issue the message, "I/O error", if the file is not found or if any other I/O error occurs.

A "word" is defined as a collection of non-blank characters surrounded by some sort of visual delimiter such as a blank or newline character. A "sentence" is defined as one or more words ending with a period. A "line" is defined as a physical line of text containing at least one non-newline character.

Task 2

Write a program to calculate chi, given probability and degrees of freedom.

Your program should accept as input the name of a file that contains pairs of data from which to perform the calculations. Each line of the file will contain a value for p (probability, where $p > 0$) and n (the number of degrees of freedom, where $n > 0$). The values p and n will be separated by one or more spaces.

Attachment 2 contains an in-depth explanation of a chi-squared distribution and the associated equations.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

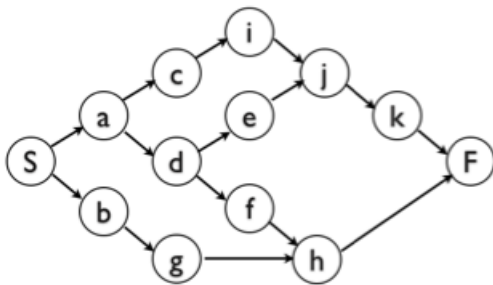
E

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write a program to identify the critical activities of a project described by a dependency network. Your program should accept as input the relevant information from a network chart. You may assume that your chart will contain no more than 100 activities and that each activity will have no more than 10 successors or 10 predecessors.

Submit your software with the following test data:



Activity	t
S	0
a	3
b	5
c	6
d	2
e	3
f	3

Activity	t
g	4
h	5
i	4
j	2
k	3
F	0

Your output should follow the general format below:

Task	Succ-essors	Pred-ecessors	ES	EF	LS	LF	TS	Critical (Y/N)
S								
a								
b								
...								

Length of the critical path is XX

ES - Earliest Start; EF - Earliest Finish; LS - Latest Start; LF - Latest Finish; TS - Total Slack

Task 2

Write a class to calculate the probability value of a t-distribution given the number of tails and the values of t and n (i.e., write a program to find the area under the t-distribution).

The software should follow an object-oriented design and implement the specification below for the class named *TCurve*.

- *TCurve*(n) -> *TCurve*
Constructor. Instantiates an instance of a t-distribution with n degrees of freedom. Raises the predefined "ValueError" exception if n is invalid. The exception has the description "Non-integer n" if n not an integer; "Invalid value of n" if 2 < n > 50; and "Missing n" if n is omitted.
- *myTCurve.p*(t, tails) -> float
Finds the area under the t-curve, given a value of t and the number of tails. Raises the "ValueError" exception with a string of "Invalid t" if t is missing, not numeric, or less than zero; raises the "ValueError" exception with a string of "Invalid tails" if the number of tails is not 1 or 2.
- *myTCurve.n*() -> integer
Returns the number of degrees of freedom of the t-curve.

The formula for integrating a T-Distribution can be found on Attachment 1.

Write the Larger Task Number Here

Briefly state why you selected this task as the larger task?

G

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write a program to calculate chi, given probability and degrees of freedom.

Your program should accept as input the name of a file that contains pairs of data from which to perform the calculations. Each line of the file will contain a value for p (probability, where $p > 0$) and n (the number of degrees of freedom, where $n > 0$). The values p and n will be separated by one or more spaces.

Attachment 2 contains an in-depth explanation of a chi-squared distribution and the associated equations.

Task 2

Write a program to calculate the probability of a t-distribution given the number of tails, the value of x , and the degrees of freedom (i.e., write a program to find the area under a t-distribution).

Your program should accept as input the name of a file that contains the 3-tuples of data from which to perform the calculations. Each line of the file will contain a value of x (the upper limit on the probability distribution integral, where $x > 0$), n (the number of degrees of freedom, where $n > 0$), and tails (the number of tails, where tails = 1 | 2).

The formula for integrating a T-Distribution can be found on Attachment 1.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

I

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write software to calculate the median, product, penultimate, and standard deviation of a list of numbers.

The software should consist of four standalone functions that follow the specification below.

- `median(list) --> numeric`
Returns the median of a list of numeric values. The median for an even number of items is the average of the two midpoint values when the items are arranged in sorted order; the median for an odd number of items is the middle value when the items are arranged in sorted order. Raises an error if the list is empty.
- `product(list) --> numeric`
Returns the product of the values in the list. Raises an error if the list is empty.
- `penultimate(list) --> numeric`
Returns the second largest value of a list of numeric values. Raises an error if the list contains one or fewer elements.
- `stdev(list) --> numeric`
Returns the standard deviation of a list of numeric values using the formula below. Raises an error if the list contains one or fewer elements.

$$\sigma(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{avg})^2}{n - 1}}$$

where $\sigma(x_1, \dots, x_n)$ is the standard deviation of the x values and x_{avg} is the average of these n values.

Task 2

When faced with a large list of values, we often want to deal with characteristics of the values rather than the values themselves. Write a class that characterizes data in such a way that you can quantify data as "very small", "small", "medium", "large", and "very large".

The software should follow an object-oriented design and implement the specification below for the class named `Buckets`.

- `Buckets()` --> `Buckets`
Constructor. Creates an instance of `Buckets`.
- `myBuckets.avg()` --> float
Represents the average of the natural log of a set of values.
$$avg(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^n \ln(x_i)$$
- `myBuckets.std()` --> float
Represents the average of the natural log of a set of values.
$$std(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (\ln(x_i) - avg)^2}{n - 1}}$$
- `myBuckets.vs()` --> float
Represents the average of the natural log of a set of values.
$$vs = [e^{avg - 2std}]$$
- `myBuckets.s()` --> float
Represents the average of the natural log of a set of values.
$$s = [e^{avg - std}]$$
- `myBuckets.m()` --> float
Represents the average of the natural log of a set of values.
$$m = [e^{avg}]$$
- `myBuckets.l()` --> float
Represents the average of the natural log of a set of values.
$$l = [e^{avg + std}]$$
- `myBuckets.vl()` --> float
Represents the average of the natural log of a set of values.
$$vl = [e^{avg + 2std}]$$
- `myBuckets.buildLogNormal()` --> None
Sets the `avg`, `std`, `vs`, `s`, `m`, `l`, and `vl` attributes to values that statistically characterize in a log normal fashion a list of input integers. Returns None. Raises a `ValueError` if the length of the list is less than two (2). Raises a `ValueError` if a list item is less than or equal to zero (0).

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

K

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

When faced with a large list of values, we often want to deal with characteristics of the values rather than the values themselves. Write a class that characterizes data in such a way that you can quantify data as "very small", "small", "medium", "large", and "very large".

The software should follow an object-oriented design and implement the specification below for the class named *Buckets*.

- `Buckets()` --> `Buckets`
Constructor. Creates an instance of *Buckets*.
- `myBuckets.avg()` -> float
Represents the average of the natural log of a set of values.

$$avg(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^n \ln(x_i)$$

- `myBuckets.std()` -> float
Represents the average of the natural log of a set of values.

$$std(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (\ln(x_i) - avg)^2}{n - 1}}$$

- `myBuckets.vs()` -> float
Represents the average of the natural log of a set of values.
 $vs = \lfloor e^{avg - 2std} \rfloor$

- `myBuckets.s()` -> float
Represents the average of the natural log of a set of values.
 $s = \lfloor e^{avg - std} \rfloor$

- `myBuckets.m()` -> float
Represents the average of the natural log of a set of values.
 $m = \lfloor e^{avg} \rfloor$

- `myBuckets.l()` -> float
Represents the average of the natural log of a set of values.
 $l = \lfloor e^{avg + std} \rfloor$

- `myBuckets.vl()` -> float
Represents the average of the natural log of a set of values.
 $vl = \lfloor e^{avg + 2std} \rfloor$

- `myBuckets.buildLogNormal()` -> None
Sets the *avg*, *std*, *vs*, *s*, *m*, *l*, and *vl* attributes to values that statistically characterize in a log normal fashion a list of input integers. Returns None.
Raises a *ValueError* if the length of the list is less than two (2). Raises a *ValueError* if a list item is less than or equal to zero (0).

Task 2

Write a class to calculate the probability value of a t-distribution given the number of tails and the values of *t* and *n* (i.e., write a program to find the area under the t-distribution).

The software should follow an object-oriented design and implement the specification below for the class named *TCurve*.

- `TCurve(n)` --> `TCurve`
Constructor. Instantiates an instance of a t-distribution with *n* degrees of freedom. Raises the predefined "ValueError" exception if *n* is invalid. The exception has the description "Non-integer *n*" if *n* not an integer; "Invalid value of *n*" if $2 < n > 50$; and "Missing *n*" if *n* is omitted.
- `myTCurve.p(t, tails)` -> float
Finds the area under the t-curve, given a value of *t* and the number of tails. Raises the "ValueError" exception with a string of "Invalid *t*" if *t* is missing, not numeric, or less than zero; raises the "ValueError" exception with a string of "Invalid tails" if the number of tails is not 1 or 2.
- `myTCurve.n()` -> integer
Returns the number of degrees of freedom of the t-curve.

The formula for integrating a T-Distribution can be found on Attachment 1.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

M

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write software to calculate the median, product, penultimate, and standard deviation of a list of numbers.

The software should consist of four standalone functions that follow the specification below.

- median(list) --> numeric
Returns the median of a list of numeric values. The median for an even number of items is the average of the two midpoint values when the items are arranged in sorted order; the median for an odd number of items is the middle value when the items are arranged in sorted order. Raises an error if the list is empty.
- product(list) --> numeric
Returns the product of the values in the list. Raises an error if the list is empty.
- penultimate (list) --> numeric
Returns the second largest value of a list of numeric values. Raises an error if the list contains one or fewer elements.
- stdev(list) --> numeric
Returns the standard deviation of a list of numeric values using the formula below. Raises an error if the list contains one or fewer elements.

$$\sigma(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{avg})^2}{n - 1}}$$

where $\sigma(x_1, \dots, x_n)$ is the standard deviation of the x values and x_{avg} is the average of these n values.

Task 2

Write a program, which reads a text file and displays the name of the file as well as the total number of words, sentences, and lines in the file.

The program should be capable of sequentially processing multiple files by repeatedly prompting the user for file names until the user enters "stop". The program should issue the message, "I/O error", if the file is not found or if any other I/O error occurs.

A "word" is defined as a collection of non-blank characters surrounded by some sort of visual delimiter such as a blank or newline character. A "sentence" is defined as one or more words ending with a period. A "line" is defined as a physical line of text containing at least one non-newline character.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

0

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

Write a program, which reads a text file and displays the name of the file as well as the total number of words, sentences, and lines in the file.

The program should be capable of sequentially processing multiple files by repeatedly prompting the user for file names until the user enters "stop". The program should issue the message, "I/O error", if the file is not found or if any other I/O error occurs.

A "word" is defined as a collection of non-blank characters surrounded by some sort of visual delimiter such as a blank or newline character. A "sentence" is defined as one or more words ending with a period. A "line" is defined as a physical line of text containing at least one non-newline character.

Task 2

Write software to calculate the average, median, and standard deviation of a list of numbers.

The software should consist of four standalone functions that follow the specification below.

- `average(list)` --> numeric
Returns the average of a list of numeric values.
Raises an error if the list is empty.
- `median(list)` --> numeric
Returns the median of a list of numeric values.
The median for an even number of items is the average of the two midpoint values when the items are arranged in sorted order; the median for an odd number of items is the middle value when the items are arranged in sorted order.
Raises an error if the list is empty.
- `stdev(list)` --> numeric
Returns the standard deviation of a list of numeric values using the formula below.
Raises an error if the list contains one or fewer elements.

$$\sigma(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (x_i - x_{avg})^2}{n - 1}}$$

where $\sigma(x_1, \dots, x_n)$ is the standard deviation of the x values and x_{avg} is the average of these n values.

Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

Q

Instructions: Read the descriptions for the two tasks below. Based on your knowledge and experience, select which of the two tasks you believe will require the most effort to complete; this is the "larger" task. In the box at the bottom of the page, write the number of the larger task. In the space provided, write a one-sentence justification for selecting that task as the larger task.

Task 1

When faced with a large list of values, we often want to deal with characteristics of the values rather than the values themselves. Write a class that characterizes data in such a way that you can quantify data as "very small", "small", "medium", "large", and "very large".

The software should follow an object-oriented design and implement the specification below for the class named *Buckets*.

- `Buckets()` --> `Buckets`
Constructor. Creates an instance of *Buckets*.
- `myBuckets.avg()` -> float
Represents the average of the natural log of a set of values.

$$avg(x_1, \dots, x_n) = \frac{1}{n} \sum_{i=1}^n \ln(x_i)$$

- `myBuckets.std()` -> float
Represents the average of the natural log of a set of values.

$$std(x_1, \dots, x_n) = \sqrt{\frac{\sum_{i=1}^n (\ln(x_i) - avg)^2}{n - 1}}$$

- `myBuckets.vs()` -> float
Represents the average of the natural log of a set of values.
 $vs = \lfloor e^{avg - 2std} \rfloor$
- `myBuckets.s()` -> float
Represents the average of the natural log of a set of values.
 $s = \lfloor e^{avg - std} \rfloor$
- `myBuckets.m()` -> float
Represents the average of the natural log of a set of values.
 $m = \lfloor e^{avg} \rfloor$
- `myBuckets.l()` -> float
Represents the average of the natural log of a set of values.
 $l = \lfloor e^{avg + std} \rfloor$
- `myBuckets.vl()` -> float
Represents the average of the natural log of a set of values.
 $vl = \lfloor e^{avg + 2std} \rfloor$
- `myBuckets.buildLogNormal()` -> None
Sets the *avg*, *std*, *vs*, *s*, *m*, *l*, and *vl* attributes to values that statistically characterize in a log normal fashion a list of input integers. Returns None.
Raises a *ValueError* if the length of the list is less than two (2). Raises a *ValueError* if a list item is less than or equal to zero (0).

Task 2

Using a completed implementation of Task 1, write a class that extracts component (classes and methods) information from actual source code then categorizes it by size using the "Buckets" class.

The software should follow an object-oriented design and implement the specification below for the class named *Script*.

- `Script(filename=string)` --> `Script`
Constructor. Creates an instance of *Script*.
"FileName" is a character sequence of non-zero length. Required. Arrives unvalidated.
Raises an *IOError* if the file cannot be opened.
Raises an *IOError* if the fileName is not of the form ".py" (where "+" connotes one or more characters).
- `myScript.extractComponents()` -> float
Builds an instance of a *Component* object (provided to you) for each class in the file. The instance contains the name of the class, the number of methods in the class, and LOC count for the class. For this assignment, the LOC count is defined as the number of physical lines the class, starting with the class definition and ending with the last non-blank line of the class. Returns a list of *Component* instances for each class in the file. Returns an empty list if no classes are present.
- `myScript.getFileName()` -> string
Returns the name of the file used to construct the instance.

Note: This will be the first use of your *Buckets* class, outside your initial testing.


Write the
Larger Task
Number
Here

Briefly state why you selected this task as the larger task?

S

Appendix D


Attitudinal Survey Questions

 qualtrics.com[®]

Select the statement that best describes you.

- I am currently an undergraduate student at Auburn University.
- I am currently a graduate student at Auburn University.
- I am a graduate of Auburn University.
- I have not attended Auburn University.


Survey Powered By [Qualtrics](#)

 qualtrics.com[®]

Select the statement that best describes you in relation to the **Software Process course (COMP 5700/6700/6706)** at Auburn University.

- I have not taken the course.
- I am currently enrolled in the course.
- I have successfully completed the course.
- I am not familiar with the course.

Survey Powered By [Qualtrics](#)

 qualtrics.com[®]

Select the statement that best describes you with regard to your **software development experience**.

- My experience is limited to coursework at a school or university.
- My experience includes a co-op or internship in addition to coursework.
- My experience includes some professional experience (part or full-time).
- I am currently working full-time in the software development industry.
- None of the above.

Survey Powered By [Qualtrics](#)

Select the statement below that best represents your attitude toward effort estimation and software engineering.

- Estimating future effort is an essential part of the software engineering discipline.
- Estimating future effort has a role in software engineering and should be implemented, when necessary.
- Estimating future effort is sometimes useful, but is unnecessary in most software engineering environments.
- Estimating future effort is an unnecessary and time consuming practice that holds little or no value in software engineering.

>>

You indicated you have either completed the Software Process course or are currently enrolled.

In this course, you have applied the **Principle-Centered Software Process (PCSE)** to software development assignments. One aspect of PCSE is **effort estimation**, a prediction of the future effort required to complete a software development assignment.

Select the statement below that best describes your understanding of the PCSE effort estimation model.

- I am very familiar with the model and fully understand how it predicts future effort.
- I am familiar with the model and have a general understanding of how it predicts future effort.
- I am somewhat familiar with the model and have a vague understanding of how it predicts future effort.
- I do not recall the details of the model and I do not understand how it predicts future effort.

>>

(A quick refresher of the PCSE model, in case you've forgotten...)

The **PCSE effort estimation model** may be described by the following high-level concepts/steps:

- Determining the lines of code per method (*LOC/m*) from past work.
- Normalizing the *LOC/m* of each component as $\ln(\text{LOC}/m)$.
- Calculating the average and standard deviation of the $\ln(\text{LOC}/m)$ values.
- Constructing a "size matrix" that characterizes past work as *very small, small, medium, large, or very large*, based on the average of the $\ln(\text{LOC}/m)$ values and their standard deviation.
- Each component of past work is then labeled with a relative size (*VS, S, M, L, VL*) by looking up the actual *LOC* in the size matrix.
- For each component of future work, a similar component of past work is identified.
- Since the past and future work components are believed to be similar, it can be assumed that their effort attributes are similar, therefore...
- The *LOC/m* value of the past work can be used to calculate the *total LOC* for the future work (i.e. $\text{total LOC} = \text{LOC}/m \times \text{number of methods}$).
- This raw *LOC* value is then adjusted based on the software engineer's historical accuracy in estimating *LOC*.
- The *adjusted LOC* value is multiplied by the software engineer's productivity (*LOC/hour*) to predict the effort in terms of time (hours).
- The confidence in this effort estimate is calculated based on the software engineer's worst overestimate and worst underestimate.

Select the statement that best describes your understanding of the PCSE effort estimation model.

- I am very familiar with the model and fully understand how it predicts future effort.
- I am familiar with the model and have a general understanding of how it predicts future effort.
- I am somewhat familiar with the model and have a vague understanding of how it predicts future effort.
- I still do not recall the details of the model and I still do not understand how it predicts future effort.

>>

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **There are no repercussions for being wrong in your estimate.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another estimation model. (Please specify the model)

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **If your estimate is wrong, your paycheck will be reduced by \$5.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another formal estimation model. (Please specify the model)

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **If your estimate is wrong, your paycheck will be reduced by \$50.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another formal estimation model. (Please specify the model)

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **If your estimate is wrong, your paycheck will be reduced by \$500.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another formal estimation model. (Please specify the model)

>>

A new effort estimation model named "SISE" has been developed, which constructs estimates based on a software engineer's perception of the size of future work as compared to the size of past work. Unlike PCSE, the SISE model is *not* based on finding similar past activities; instead, the SISE model is based on finding two past activities which are perceived to be smaller and larger, respectively, than the future work.

SISE follows four general steps:

- *Sort* your past work activities from smallest to largest by the actual effort that was required to complete the work. Hide all numeric values; estimates are based solely on work descriptions.
- *Identify* two past activities: one confidently known to be smaller, one confidently known to be larger, and both relatively close in size to the future work.
- *Size* the future activity with a rough prediction interval defined by the actual efforts of the selected smaller and larger past activities.
- *Evaluate* if dissatisfied with the precision, accuracy, or confidence level of the estimate, then shift or resize the prediction interval based on historical bias.

The confidence in the effort estimate is calculated based on the size of the prediction interval and the software engineer's historical accuracy in constructing effort estimates.

Select the statement below that best describes your understanding of the SISE effort estimation model based on the above description.

- I have a clear understanding of the SISE model and how it produces effort estimates.
- I have a general understanding of the SISE model and how it produces effort estimates.
- I have a vague understanding of the SISE model and how it produces effort estimates.
- I have no idea how the SISE model works and do not know how it produces effort estimates.

Select the statement below that best describes your belief in the value of an effort estimate produced by the SISE model.

- I believe the SISE model uses a solid approach and will produce highly accurate estimates.
- I believe the SISE model uses a mostly valid approach and should produce relatively accurate estimates.
- I believe the SISE model has some validity and may produce useful estimates.
- I do not believe the SISE model to be valid and I doubt it will produce any meaningful estimates.

>>

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **There are no repercussions for being wrong in your estimate.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the SISE model.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another estimation model. (Please specify the model)

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **If your estimate is wrong, your paycheck will be reduced by \$5.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the SISE model.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another estimation model. (Please specify the model)

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **If your estimate is wrong, your paycheck will be reduced by \$50.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the SISE model.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another estimation model. (Please specify the model)

Assume you are working for a software development firm. You have been asked to predict how much time it will take to develop a new software component. **If your estimate is wrong, your paycheck will be reduced by \$500.**

Select the statement that best describes your attitude concerning this scenario.

- I would spend as little time as possible and just make a guess as to the estimated effort.
- I would take a little time, consider the problem, and formulate an estimate based on my knowledge and experience.
- I would construct an estimate using the SISE model.
- I would construct an estimate using the PCSE model.
- I would construct an estimate using another estimation model. (Please specify the model)

>>

Many software development organizations rely on expert judgement (i.e. guesswork) to construct effort estimates.

Select the statement below that best describes your understanding of expert judgement as it relates to estimating the effort of a future task.

- Expert judgement is a completely accurate approach.
- Expert judgement is a highly accurate approach.
- Expert judgement is a moderately accurate approach.
- Expert judgement is a slightly accurate approach.
- Expert judgement is an inaccurate approach.

Select the statement below that best describes your understanding of how an expert uses their judgement to construct an effort estimate.

- I completely understand how the knowledge and experiences of a software engineer can be used to predict the future effort required to complete an activity.
- I have a relatively clear idea of how the knowledge and experiences of a software engineer can be used to predict the future effort required to complete an activity.
- I have a vague idea of how the knowledge and experiences of a software engineer can be used to predict the future effort required to complete an activity.
- I have no idea how the knowledge and experiences of a software engineer can be used to predict the future effort required to complete an activity.

Select the statement below that best describes your confidence in the use of expert judgement as it relates to effort estimation.

- I am completely confident in estimates based solely on expert judgement.
- I have strong confidence in estimates based solely on expert judgement.
- I have some confidence in estimates based solely on expert judgement.
- I have little confidence in estimates based solely on expert judgement.
- I have no confidence in estimates based solely on expert judgement.

Select the statement below that best describes your perspective on using expert judgement as it relates to effort estimation.

- I often use or I am highly likely to use expert judgement.
- I sometimes use or would possibly use expert judgement.
- I seldom use or likely would not use expert judgement.
- I do not use expert judgement.



The PCSE and SISE effort estimation models take two distinct approaches to constructing an estimate. You may click these links to expand/collapse a description of each model:

- [\[Show/hide PCSE model description\]](#)
- [\[Show/hide SISE model description\]](#)

Based on the descriptions of each model, select the statement below that best described your impression of the PCSE model as compared to the SISE model in terms of time investment.

- I believe the *PCSE model* would require a larger time investment than the SISE model.
- I believe neither model is more time-consuming to utilize than the other.
- I believe the *SISE model* would require a larger time investment than the PCSE model.

Based on the descriptions of each model, select the statement below that best described your impression of the PCSE model as compared to the SISE model in terms of complexity.

- I believe the PCSE model is much more complex as compared to the SISE model.
- I believe the PCSE model is somewhat more complex as compared to the SISE model.
- I believe the PCSE model is the same complexity as the SISE model.
- I believe the PCSE model is somewhat less complex as compared to the SISE model.
- I believe the PCSE model is much less complex as compared to the SISE model.

Based on the descriptions of each model, select the statement below that best described your impression of the PCSE model as compared to the SISE model in terms of the value of the estimate produced.

- I believe the PCSE model will produce much more valuable estimates as compared to the SISE model.
- I believe the PCSE model will produce somewhat more valuable estimates as compared to the SISE model.
- I believe the PCSE model will produce estimate of equal value as compared to the SISE model.
- I believe the PCSE model will produce somewhat less valuable estimates as compared to the SISE model.
- I believe the PCSE model will produce much less valuable estimates as compared to the SISE model.

>>

The PCSE and SISE effort estimation models take two distinct approaches to constructing an estimate. You may click these links to expand/collapse a description of each model:

- [\[Show/hide PCSE model description\]](#)

The PCSE effort estimation model may be described by the following high-level concepts/steps:

- Determining the lines of code per method (*LOC/m*) from past work.
- Normalizing the *LOC/m* of each component as $\ln(\text{LOC}/m)$.
- Calculating the average and standard deviation of the $\ln(\text{LOC}/m)$ values.
- Constructing a "size matrix" that characterizes past work as *very small, small, medium, large, or very large*, based on the average of the $\ln(\text{LOC}/m)$ values and their standard deviation.
- Each component of past work is then labeled with a relative size (*VS, S, M, L, VL*) by looking up the actual *LOC* in the size matrix.
- For each component of future work, a similar component of past work is identified.
- Since the past and future work components are believed to be similar, it can be assumed that their effort attributes are similar, therefore...
- The *LOC/m* value of the past work can be used to calculate the *total LOC* for the future work (i.e. $\text{total LOC} = \text{LOC}/m * \text{number of methods}$).
- This raw *LOC* value is then adjusted based on the software engineer's historical accuracy in estimating *LOC*.
- The *adjusted LOC* value is multiplied by the software engineer's productivity (*LOC/hour*) to predict the effort in terms of time (hours).
- The confidence in this effort estimate is calculated based on the software engineer's worst overestimate and worst underestimate.

- [\[Show/hide SISE model description\]](#)

The SISE model follows four general steps:

- Sort your past work activities from smallest to largest by the actual effort that was required to complete the work. Hide all numeric values; estimates are based solely on work descriptions.
- Identify two past activities: one confidently known to be smaller, one confidently known to be larger, and both relatively close in size to the future work.
- Size the future activity with a rough prediction interval defined by the actual efforts of the selected smaller and larger past activities.
- Evaluate If dissatisfied with the precision, accuracy, or confidence level of the estimate, then shift or resize the prediction interval based on historical bias.

The confidence in the effort estimate is calculated based on the size of the prediction interval and the software engineer's historical accuracy in constructing effort estimates.

The SISE effort estimation model and expert judgement take two distinct approaches to constructing an estimate. You may click these links to expand/collapse a description of the SISE model:

- [\[Show/hide SISE model description\]](#)
- [\[Show/hide Expert Judgment description\]](#)

- I believe the *SISE model* would require a larger time investment than expert judgement.
- I believe neither approach is more time-consuming to utilize than the other.
- I believe *expert judgement* would require a larger time investment than the SISE model.

Based on the description of the SISE model and your experience with expert judgement (i.e. guesswork), select the statement below that best described your impression of the SISE model as compared to expert judgement in terms of complexity.

- I believe the SISE model is much more complex as compared to expert judgement.
- I believe the SISE model is somewhat more complex as compared to expert judgement.
- I believe the SISE model is the same complexity as expert judgement.
- I believe the SISE model is somewhat less complex as compared to expert judgement.
- I believe the SISE model is much less complex as compared to expert judgement.

Based on the description of each the SISE model and your experience with expert judgement (i.e. guesswork), select the statement below that best described your impression of the SISE model as compared to expert judgement in terms of the value of the estimate produced.

- I believe the SISE model will produce much more valuable estimates as compared to expert judgement.
- I believe the SISE model will produce somewhat more valuable estimates as compared to expert judgement.
- I believe the SISE model will produce estimate of equal value as compared to expert judgement.
- I believe the SISE model will produce somewhat less valuable estimates as compared to expert judgement.
- I believe the SISE model will produce much less valuable estimates as compared to expert judgement.

>>

The SISE effort estimation model and expert judgement take two distinct approaches to constructing an estimate. You may click these links to expand/collapse a description of the SISE model:

- [\[Show/hide SISE model description\]](#)

The SISE model follows four general steps:

- Sort your past work activities from smallest to largest by the actual effort that was required to complete the work. Hide all numeric values; estimates are based solely on work descriptions.
- Identify two past activities: one confidently known to be smaller, one confidently known to be larger, and both relatively close in size to the future work.
- Size the future activity with a rough prediction interval defined by the actual efforts of the selected smaller and larger past activities.
- Evaluate if dissatisfied with the precision, accuracy, or confidence level of the estimate, then shift or resize the prediction interval based on historical bias.

The confidence in the effort estimate is calculated based on the size of the prediction interval and the software engineer's historical accuracy in constructing effort estimates.

- [\[Show/hide Expert Judgment description\]](#)

Expert judgement relies on an individual software engineer's ability to assign a numeric value to the effort estimate based on their knowledge, experiences, and intuition. Expert judgement is not a prescriptive model and estimators may use a variety of conscious or subconscious approaches to construct the estimate, such as analogy or work breakdown.