SISE: A Novel Approach to Individual Effort Estimation

Russell Thackston, Georgia Southern University
David Umphress, Auburn University

Abstract. The SISE estimation model presents a novel alternative to PSP's PROBE by combining expert judgment with empirical data to create reasonably accurate predictions. The model uses a simple, four-steps process in which a future activity is compared to a developer's completed tasks to identify two known effort values that define a prediction interval. Initial validation of the SISE model by researchers at Auburn University was completed in July 2013 and development of supporting tools is currently underway.

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 focused on individuals as opposed to teams. For example, recent surveys of the microISV industry have shown that time management and related issues topped their founders' list of "pain points"[1]. 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[2].

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 – such as the Personal Software Process (PSP) PROxy-Based Estimation (PROBE) model – which rely on complex software, algorithms, or mathematical calculations.

Traditional Models

A variety of established models exist for estimating the size of a future task. The vast majority – Planning Poker, Wideband Delphi, COCOMO, etc. – focus on team- or project-level estimation. Most of these models have demonstrated an ability to accurately predict future effort through the combined efforts of a team. However, the individual software engineer is often left to his or her own devices when it comes to planning personal activities on a daily basis.

Currently, only one formal estimation model exists that is specifically targeted at the individual software engineer: PSP's PROBE[3]. The PROBE model relies on a proxy-based approach in which estimators:

- Develop their conceptual design;
- Identify and size proxies for the actual work;
- Estimate other elements;
- Estimate the program size using one of four methods;
- Calculate prediction intervals.

PROBE forecasts effort through a set of rules that determine the statistical relationship between size estimates and actual effort across past projects. The approach falls back on "engineering judgment" when no demonstrable relationship exits.

The disadvantage to PROBE is in its perceived complexity and inflexibility. The rules for determining which pieces of historical data provide the strongest statistical mapping of size to effort niggle developers, especially in the absence of any tool. The approach requires a substantial amount of bookkeeping. It relies on statistically significant amounts of historical data from relatively stable development efforts to avoid consistent use of the "engineering judgment" rule. On the plus side, PROBE does have the advantage of being the sole prescriptive model for costing effort at the level of the individual engineer.

When pressed for a quick, familiar approach to estimation -- especially in informal settings -- individual developers rely on expert judgment. While expert judgment is recognized as a valid and sometime accurate approach to effort estimation, it lacks formal, quantitative methods. Formal methods that may be incorporated into expert judgment include estimation by analogy, case-based estimation, and work breakdown, but the greatest challenge to expert judgment is in what to change to improve accuracy if the approach leading to the estimate is undefined or vague.

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 detectable 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.

The goal of the SISE model is to provide a viable alternative to traditional models such as PROBE. The SISE model defines an agile approach to estimation by offering a lighter-weight method than those found in models such as PROBE and PCSE; the model employs fewer steps, which are demonstrably less complex. Additionally, the SISE model improves upon simple guesswork by relying on a foundation of empirical data.

The SISE Steps

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."

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.

The exact manner in which tasks are chosen as smaller and larger is left to the practitioner, as is the determination of confidence. In this instance, SISE views the practitioner's decision-making process as a black box; the model relies on the software engineer's intuition and experience to make complex, yet relatively accurate, judgment calls. The only other important consideration is consistency from estimate to estimate.

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[4].

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.

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 determining if the estimates are typically low or high. If a consistent bias can be demonstrated, then the future estimate is adjusted to account for the bias. The Removing Shift Bias section describes a sample scenario.

If adjusting for shift bias is not possible or has not produced the desired results, the practitioner may adjust for width bias. Width bias results from historical prediction intervals that are too narrow or too wide. Large prediction intervals create a high level of confidence, but make the estimate less useful for planning purposes through reduced precision. By comparing a future task's prediction interval width and confidence level to those of historical tasks, the practitioner may choose to increase or decrease the prediction interval width, and, therefore, achieve a more desirable precision/confidence level for the future task's estimate. The Removing Width Bias section describes a sample scenario.

This last step 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.

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.

Framework vs. Methodology

The SISE model outlines a general framework for constructing estimates. However, the model is not prescriptive in terms of the lower level details of the process. The manner in which a software engineer tracks his or her time, identifies the smaller and larger tasks, and determines an acceptable prediction interval width/confidence level, are left to the practitioner. In these instances, the only important consideration is consistency; time tracking and relative task sizing must follow a similar method from day-to-day and week-to-week.

Additionally, the SISE model allows the practitioner to select an appropriate level of granularity for his or her work tasks. Typically, a software engineer must plan each day's activities. SISE facilitates such planning and, indirectly, supports longer term planning (e.g. weeks and months), by providing a quantitative estimate at

the lowest levels. Therefore, individual software engineers must determine the smallest unit of work that is appropriate for them personally for planning purposes. These units of work will define scope and frequency of each SISE estimate. Again, the most important aspect is consistency; the granularity of estimates should not change dramatically from week-to-week or month-to-month.

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.

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 section 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% confidence 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 bias (see the Adjusting for Width Bias and Adjusting for Shift Bias sections). 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.

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, ∞]. Once the first activity is completed and the actual value is recorded, the hit rate is calculated to be 100% (see Table 1).

Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
Design security model	0	∞	10

Table 1: One completed activity (Hit rate = 100%)

Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
Design user model	0	10	8
Design security model	0	∞	10

Table 2: Two completed activities (Hit rate = 100%)

Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
Design user model	0	10	8
Design security model	0	∞	10
Design content model	8	10	11

Table 3: Three completed activities (Hit rate = 67%)

Task	Low Est. (hours)	High Est. (hours)	Actual (hours)
Design database tables	0	11	6
Design user model	0	10	8
Design security model	0	∞	10
Design content model	8	10	11

Table 4: Four completed activities (Hit rate = 75%)

Task	Low Est. (hours)	High Est. (hours)	Actual (hours)	Missed Prediction Interval?
Design FAQ 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 class	0	11	14	Yes

Table 5: Ten completed activities (Hit rate = 70%)

Task	Low Est. (hours)	Adj. Low	High Est. (hours)	Adj. High	Actual (hours)	Missed Prediction Interval?
Design FAQ 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 class	0	0	11	12	14	Yes

Table 6: Adjusting for width bias (Hit rate = 90%)

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%.

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 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).

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].

The confidence level is assumed to be 67%, based on the historical hit rate. After referring to the sections 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 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 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.

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 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.

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.

Additional validation of the SISE model within an industrial setting is planned for the near future. This will provide an opportunity to view the performance of SISE in a less structured (i.e. non-academic) environment over a longer term. In addition, an industrial environment will provide critical feedback on SISE's ability to integrate into a team environment, as the individual estimates are rolled up into team and project-level estimates.

Removing Shift

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 predominantly 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 in Table 7.

Table 7:

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

Table 8 shows an example of how widening the prediction interval may allow for an increase in the hit rate from 60% to 80%.

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%.

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:

Table 8:

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

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 list of work activities in Table 9.

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%.

Table 9:

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

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.

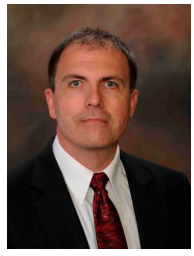
NOTES

1. M Jorgensen, K H Teigen, and K J Molokken-Ostvold. Better sure than safe? Over confidence in judgment based software development effort prediction intervals. Journal of Systems and Software, 70(1-2):79–93, 2004.
2. 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.
3. Pomeroy-Huff, Marsha, et al. "The Personal Software ProcessSM (PSPSM) Body of Knowledge, Version 2.0." Software Engineering Institute. Carnegie Mellon, August 2009. Web. 30 October 2013.
4. Russell Thackston and David Umphress. Micropreneurs: The rise of the microISV. IT Professional, 15(2):50–56, 2013.

ABOUT THE AUTHORS

Russell Thackston is an Assistant Professor of IT at Georgia Southern University. He earned his Ph.D. in computer science and software engineering from Auburn University. His research interests focus on software process and effort estimation, specifically with regard to individual software engineers and microISVs. His work on SISE -- an effort estimation model tailored to individuals -- has been published in Crosstalk, the Journal of Defense Software Engineering, and his research into microISVs has been published in IEEE's IT Pro. Russell served in the U.S. Air Force during Desert Storm and Desert Calm and has been happily married for more than 25 years.

Phone: 912-478-4218
E-mail: rthackston@georgiasouthern.edu



David A. Umphress, Ph.D., is an associate professor of computer science and software engineering at Auburn University, where he specializes in software development processes. He has worked over the past 30 years in various software and system engineering capacities in military, industry, and academia settings. Umphress is an Institute of Electrical and Electronics Engineers (IEEE) certified software development professional.

Phone: 334-844-6335
E-mail: david.umphress@auburn.edu