Individual Effort Estimating

Not Just for Teams Anymore

Russell Thackston, Auburn University
David A. Umphress, Auburn University

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.

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 effort estimation methods can be employed for one-person endeavors and those methods can benefit teams, but those 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.

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 ecosystems, such as Apple's 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 in the Apple App Store requires an investment of around $100—not including a device—as the only requirement is the $99 per year to join the Apple iOS Developer Program [2]. 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 [4].

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 to 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).

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 overwhelmingly 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 [6], for example, exemplifies this philosophy of guessing effort individually then attenuating the range of guesses through team dialog. 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 (heavyweight) or it produces low-quality results (lightweight or guessing). In either case, the ROI does not 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.

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 half of all published research on 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 [7]. 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, the Constructive Cost Model (COCOMO), one of the most well known models, uses 15 cost drivers or "attributes" to estimate the size of the product to be created [8]. 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 estimate, the typical software project is on time, on budget, and fully functional only about one-third of the time [9]. 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 [10]. 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 "lightweight" and suitable to the agile environments of one-person software development teams.

Estimation for the One-person Team

In 1995, Watts Humphrey introduced the Personal Software ProcessSM (PSPSM), which defined the first published software process tailored for the individual [11]. 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 Proxy-based Estimating (PROBE). In general, a proxy is a generic stand-in for something else, where the two objects' 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 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 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 [12]. 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 ProcessSM (TSPSM) [13] corroborates the fusing of 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 Principle-Centered Software Engineering (PCSE) [14]. PCSE uses a proxy-based approach to estimating SLOC, 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 SLOC 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 [15]. 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 estimate 20-30% lower than actual effort [16], this would explain why the gut feel approach is typically heavily padded.

A "Better Than Guessing" Agile Approach

A void exists in the spectrum of effort estimating tools, specifically focused on lightweight tools for the individual. Sort Insert Size Estimate (SISE) 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 ad-

dressed 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.

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.

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.

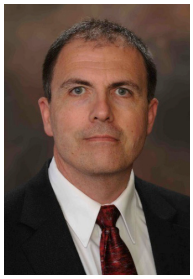
Disclaimer:

PSPSM and TSPSM are service marks of Carnegie Mellon University.

ABOUT THE AUTHORS

Russell Thackston, MSCIS, is a graduate student in Auburn University's Department of Computer Science and Software Engineering. He has spent more than 15 years developing business solutions as a software developer and consultant in the retail, commercial, and open source environments. He served in the U.S. Air Force during Desert Storm and Desert Calm and has been happily married for 25 years.

Phone: 334-246-2600
Fax: 334-844-6329
E-mail: Russell.Thackston@Auburn.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 IEEE certified software development professional.

Phone: 334-844-6335
Fax: 334-844-6329
E-mail: David.Umphress@Auburn.edu

REFERENCES

1. "July 2009 Metrics Report." AdMob Metrics. Web. 14 Aug. 2011. <http://metrics.admob.com/2009/08/july-2009-metrics-report/>.
2. "Apple Developer Programs 2011." Apple Developer. Web. 14 Aug. 2011. <http://developer.apple.com/programs/>.
3. "Android Market Developer Signup." Android Market. Web. 14 Aug. 2011. <https://market.android.com/publish/signup/>.
4. "Gartner Says Worldwide Software as a Service Revenue Is Forecast to Grow 21 Percent in 2011." Technology Research | Gartner Inc. Web. 14 Aug. 2011. <http://www.gartner.com/it/page.jsp?id=1739214>.
5. "ASP Member Forum." Association of Software Professionals. Web. 14 Aug. 2011. <http://members.asp-software.org/newsgroups/showthread.php?t=25806>.
6. Cohen, Mike. "Succeeding with Agile." Reading, MA: Addison-Wesley. 2010. Print
7. Jorgensen, Magne, and Martin Shepperd. "A Systematic Review of Software Development Cost Estimation Studies." IEEE Transactions on Software Engineering 33.1 (2007): 33-53. Print.
8. Boehm, Barry W. "Software Engineering Economics." IEEE Transactions on Software Engineering SE-10.1 (1984): 4-21. Print.
9. "Standish Chaos Report 2009," available at <https://secure.standishgroup.com/reports/reports.php>
10. Jorgensen, Magne, and Martin Shepperd. "A Systematic Review of Software Development Cost Estimation Studies." IEEE Transactions on Software Engineering 33.1 (2007): 33-53. Print.
11. Humphrey, Watts S. A Discipline for Software Engineering. Reading, MA: Addison-Wesley, 1995. Print.
12. Philip M. Johnson and Anne M. Disney, "The Personal Software Process: A Cautionary Case Study," IEEE Software, November/December 1998: 85.
13. Humphrey, Watts S. "Introduction to the Team Software Process". Reading, MA: Addison-Wesley. 2000. Print.
14. "Principle-Centered Software Engineering" Auburn University Web. 12 Oct. 2011. <http://swemac.cse.eng.auburn.edu/~umphrda>.
15. "ASP Member Forum." Association of Software Professionals. Web. 14 Aug. 2011. <http://members.asp-software.org/newsgroups/showthread.php?t=25806>.
16. Michiel Van Genuchten, "Why is Software Late? An Empirical Study of Reasons for Delay in Software Development," IEEE Transactions on Software Engineering, Vol. 17, No. 6 (June 1991): 582. Print.