Improving Consistency of Use Case Points Estimates

Dr. David J. Coe
University of Alabama, Huntsville

Use cases document key product functionality and have been used as the basis for estimating software product development efforts. Presented here is experimental data on the use of the Use Case Points (UCP) method to estimate development effort for a semester-scale software product. From a common set of initial requirements, six student teams refined these requirements and produced independent effort estimates from their own use case models. The sources of estimation inconsistency are examined and recommendations for improving the consistency of the estimation process are presented.

Use cases are a common technique for documenting software requirements. A *use case* is often defined as a step-by-step description of the interaction between the software and one or more *actors* (the people or other systems that utilize the software product to complete a given task). A use case model of a software product is thus the set of all use cases that describe the product's desired functionality. Well-written use cases concisely summarize product functionality in a way that is easy for customers to understand. They may also provide early insight on project complexities and give software developers a starting point from which to estimate total development effort. The UCP estimation method was proposed by Gustav Karner as a way of estimating resources for projects developed using the Objectory methodology (which later evolved into the Unified Process) [1-2].

The UCP method takes into account the complexities of the use cases themselves and the complexities of the users (or actors) that will interact with the software product. A weighted equation based upon the number of steps in each use case and the complexities of the actors determines an initial UCP estimate. This initial estimate is then scaled by a technical factor, to adjust the estimate by the product's perceived technical challenges, and by an environmental factor, to adjust the estimate for people-related factors such as skill levels, experience, motivation, etc. The scaled UCP estimate may then be used to estimate development effort by multiplying it by an experimentally derived Productivity Factor (PF). The mechanics of the UCP calculation resemble that of Albrecht's Function Point method from which this method was derived [2-3].

Students in an introductory software engineering course were required to use Karner's UCP method to estimate the overall effort required to implement a simple hotel reservation system. An overview of the team project is presented, followed by a step-by-step review of the UCP method. The PFs achieved by the student teams are determined using the UCP estimates and time sheet data, and sources of discrepancies in the estimates are analyzed. Finally, the lessons learned from this experience are summarized.

Team Project Requirements

Six, four-person student teams were each given one semester to implement a simplified hotel reservation system in C++ using a common set of initial requirements. The hotel reservation system was to allow hotel employees to make reservations for customers and to generate reports that summarized the current state of the hotel's room reservations. The base rate charged for each hotel room would be set in advance and could be changed to reflect variations in demand throughout the year. The actual rate charged, however, could vary depending upon which of the four types of reservations (prepaid, advance, conventional, or incentive) was selected. The reservation system also had to apply specified penalties for no-show guests. At check out, guests were to receive a bill that summarized the details of their reservation and the total charges.

The reservation system was also required to maintain records of all transactions, including reservation changes and cancellations, and archive those records to disk. Finally, employees were to have the option of generating one of the five types of reports that summarized various aspects of hotel operation, including lists of expected arrivals and current occupancy lists. Additional details on the hotel reservation system project may be found in Appendix A of [3].

Teams were required to practice object-oriented analysis and design techniques learned in the course. Given the time constraints of the semester, a graphical user interface for the reservation system was optional. All teams were required to follow an iterative and incremental development process based upon the five core disciplines of the Unified Process (requirements, analysis, design, implementation, and test). During the project, teams were required to generate and refine seven different deliverables as follows:

- Software Project Management Plan.
- Use Case Model.
- Unified Modeling Language (UML) Class Diagrams.
- UML Sequence Diagrams.
- Test and Integration Plan.
- C++ Source Code and Makefile.
- User Manual.

Five intermediate milestones were established to encourage teams to start the project early and avoid the last minute heroic efforts at the end of the semester. At each milestone, teams were also required to submit timesheets that summarized the total time spent by each team member on the submitted artifacts at that milestone. This data on actual effort would later be used to compute the PF for each team.

I first came across the UCP estimation method in [2] as the teams were nearing completion of their Use Case models. Since my previous software estimation experience utilized line-of-code (LOC) based methods, I added a UCP estimate requirement to the team project to see how well this estimation technique would work, especially since all six teams would be producing estimates for the same product. Students were asked to use their existing use case model, as is, when completing the UCP estimate.

Brief Review of the UCP Method

This overview of the UCP estimation method is derived from [1, 2]. Included in the discussion is an example UCP calculation, utilizing values reported by one of the student teams. Following this review of the UCP method, the estimates prepared by the six student teams are presented and analyzed.

Step 1: Unadjusted Use Case Weight

Starting with the use case model, the first step in the UCP estimation method is to calculate the Unadjusted Use Case Weight (UUCW), which is a weighted sum of the total number of steps identified in all of the use cases. The use cases are sorted into three different categories (Simple, Average, or Complex) depending upon the number of steps (or transactions) identified or the perceived complexity of implementation (estimated number of objects required). Table 1 describes the use case category criteria.

Consider the data reported by Team A. Team A identified a total of 14 use cases for the team project. Seven of these use cases were determined to be *Simple* in that the use cases consist of three or fewer transactions, and seven were considered to be *Average* in that they consisted of four to seven transactions. Team A identified no *Complex* use cases. Thus, the weighted sum UUCW for Team A is computed as follows:

Team A UUCW = 7 * 5 + 7 * 10 + 0 * 15 = 105

Step 2: Unadjusted Actor Weight

As with the use cases, the actors are also categorized by their perceived complexity (Simple, Average, or Complex). The Unadjusted Actor Weight (UAW) is a weighted sum of all actors appearing in the use case model. Table 2 summarizes the actor category criteria and category weights. Team A reported a single Complex actor in their use case model and no Simple or Average actors. The weighted sum UAW for Team A is calculated as follows:

Team A UAW = 0 * 1 + 0 * 2 + 1 * 3 = 3

Step 3: Unadjusted UCP

The Unadjusted UCP (UUCP) is computed as the sum of the UUCW and the UAW. For Team A, the UUCP is computed as follows:

Team A UUCP = UUCW + UAW = 105 + 3 = 108

Step 4: Technical Complexity Factor

The Technical Complexity Factor (TCF) is used to adjust the UCP estimate based upon the perceived technical complexities of the project. The influences of 13 technical factors on the development effort are estimated using a scale from 0 (irrelevant) to 5 (essential) with the value 3 used if the factor's influence is unknown. The technical factors and their relative weights are described in Table 3. For each technical factor, the influence estimate is multiplied by the corresponding factor weight and summed across all thirteen factors to compute the Technical Total Factor (TTF). The TCF is computed from the TTF as follows:

TCF = 0.60 + 0.01 * TTF

Using Team A's influence estimates as

Use Case Category	Category Description		Category Weight
	Transactions T per Use Case	Objects Required for Implementation R	
Simple	$T < 4$	$R < 5$	5
Average	$4 \le T \le 7$	$5 \le R \le 10$	10
Complex	$T > 7$	$R > 10$	15

Table 1: *Use Cases Complexity Categories and Weights* [1-2]

Actor Category	Category Description	Category Weight
Simple	Actor represents another system interacting through a defined application programming interface	1
Average	An actor which represents interaction with another system via a protocol or human interaction through a text interface	2
Complex	An actor which interacts through a graphical user interface	3

Table 2: *Actor Complexity Categories and Weights* [1-2]

Table 3: *Technical factors that influence complexity as described in* [1, 2] *along with their relative weights. Also included are influence estimates of each factor made by Team A. Influence estimates range from* 0 *(irrelevant) to* 5 *(essential).*

Technical Factor	Factor Description	Factor Weight	Team A Influence Estimates
T_1	Distributed systems	2	0
T_2	Response time or throughput performance	1	4
T_3	End user efficiency	1	3
T_4	Complex internal processing	1	2
T_5	Reusability of code in other applications	1	0
T_6	Ease of installation	0.5	4
T_7	Ease of use	0.5	4
T_8	Portability	2	0
T_9	Changeability	1	3
T_{10}	Concurrency	1	0
T_{11}	Special security features	1	0
T_{12}	Provide direct access for third parties	1	0
T_{13}	Special user training facilities	1	2

Table 4: *Environmental factors that influence complexity as described in* [1-2] *along with their relative weights. Also included are influence estimates of each factor made by Team A. Influence estimates range from* 0 *(irrelevant) to* 5 *(essential).*

Environmental Factor	Factor Description	Factor Weight	Team A Influence Estimates
E_1	Familiarity with Unified Process*	1.5	1
E_2	Part time workers	-1	4
E_3	Analyst capability	0.5	1
E_4	Application experience	0.5	1
E_5	Object-oriented experience	1	3
E_6	Motivation	1	4
E_7	Difficult programming language	-1	0
E_8	Stable requirements	2	3

*changed from original term Objectory

Project	Number of Actors (complexity)	Number of Use Cases (complexity)	TCF	ECF	UCPs	Person-Hours Required	PF (Hours per UCP)
A	5 (average)	10 (average)	1.00	0.975	107.25	2150	20.05
B	5 (average)	50 (average)	1.00	1.175	599.25	12500	20.86
C	5 (average)	15 (average)	1.00	1.175	188.00	5400	28.72

Table 5: *Project Metrics Used by Karner to Determine PF* [1]

shown in Table 3, Team A computed a TTF of 18 resulting in a 0.78 TCF.

Step 5: Environmental Complexity Factor

The Environmental Complexity Factor (ECF) is used to adjust the estimate for people-related factors such as skill levels, experience, motivation, etc. The influences of eight environmental factors on the development effort are estimated using a scale from 0 (irrelevant) to 5 (essential) with the value 3 used if the factor's influence is unknown. Table 4 (see previous page) lists the UCP environmental factors, their relative weights, and their estimated impact according to Team A. For each environmental factor, the influence estimate is multiplied by the corresponding factor weight and summed across all eight factors to compute the Environmental Total Factor (ETF). The ECF is computed from the ETF using the following equation:

$$ECF = 1.4 - 0.03 * ETF$$

Using Team A's influence estimates as shown in Table 4, Team A computed an ETF of 11.5 resulting in a 1.06 Environmental Complexity Factor.

Step 6: Compute UCPs and Estimated Labor

The UCP estimate for the product is computed as the product of the initial unadjusted use case points, the TCF, and the ECF. For Team A, the UCP computation is summarized next:

$$\text{Team A } UCP = UUCP * TCF * ECF = 108 * 0.78 * 1.06 = 89$$

The Estimated Labor is the product of the UCP estimate and an experimentally determined PF. For the three products presented by Karner in [1], the PF was experimentally determined to be in the range of 20-30 person-hours per use case point as shown in Table 5. Fitting a straight line to this data, Karner computed a nominal productivity of 20 person-hours per use case point.

In practice, there are a number of different methods that one might use to determine a PF. One could use UCP PFs documented in the software engineering literature. These values, however, may not produce an accurate estimate since they do not necessarily reflect the types of systems you are developing, the programming languages or development tools that you will be using, etc. Another approach would be to utilize your own organization's project metrics to calculate a PF. An advantage of this approach is that these metrics would account for your particular organization's software development process. Another advantage is that you could selectively include metrics from relevant projects only.

Having no experimental data on student teams using this method for a semester-sized student project, students were asked to complete their estimates using the nominal value of 20 person-hours per use case point to demonstrate that they understood the mechanics of the calculation. Students were warned that this would likely result in an overestimate of the actual labor required on the assigned project since Karner's nominal productivity was derived from data collected from larger commercial products. I planned to use this first classroom experience with UCP estimation to gather metrics that I might use to compute a more realistic PF for subsequent classroom use. My goal was that each student would need to average at minimum five hours per person per week to satisfactorily complete the team software project. Assuming four students per team working 12 weeks at five hours per person per week, my nominal labor goal was 240 hours total per team. The calendar spacing of the milestones was intended to force the students to distribute this labor uniformly across the span of the project.

Table 6: *Summary of Team UCP Estimates, Reported Effort, and PF*

Student Team Project Data	Student Teams						Mean All Teams
	A	B	C	D	E	F	
UCP Estimates	89	74	56	275	53	123	111.6
Reported Effort (hours)	297	262	216	170	171	384	249.7
Actual PF (hours/UCP)	3.34	3.52	3.89	0.62	3.22	3.12	2.95

Student UCP Estimates

The UCP estimates prepared by the student teams are summarized in Table 6 along with reported labor hours required to complete the project. Even though all teams were estimating the same project starting with the same initial list of requirements, a wide range of UCP estimates were produced with the largest estimate of 275 UCPs being approximately five times larger than the smallest estimate of 56 UCPs. Reported effort also varied over a wide range from a low of 170 hours to a high of 384 hours. Calculated productivities ranged from 0.62 to 3.89 hours per use case point, which is significantly lower than Karner's nominal value. To determine the source of the discrepancies, we examine the intermediate calculations below.

Analysis of Estimate Discrepancies

Table 7 summarizes the intermediate UCP calculations for all six teams. The subsequent discrepancy analysis examines the UCP calculations and identifies factors that contributed to the observed variations across the student teams. Recommendations for improving the consistency and quality of the estimates are also presented.

UUCW Calculations

The majority of teams identified between nine and 14 distinct use cases though Team B and Team D identified 18 and 32 use cases respectively. The primary source of discrepancy in the number of use cases identified was the consolidation or expansion of related use cases, that is, a use case with multiple scenarios may have been split and counted as multiple distinct uses cases. An example of this would be the *Reserve Room* use case where each of the four types of room reservations is treated as a separate use case. Splitting of such a use case could inflate the UUCW if the complexity of the resulting use cases did not decrease. Team D's estimate is an extreme example of this situation.

Another source of UUCW discrepancy is in the number of actor-product interactions identified. Some teams described additional interactions for a given use case than did other teams due to either an alternate interpretation of the initial project requirements or feature creep, the incorporation of extra features by the developers. For example, one team included a user authentication interchange with every use case. Other teams included confirmation interchanges to verify user inputs. In other instances, a team would split an interaction into multiple interchanges. For example, the hotel manager should be able to set the base room rate on a given date. Some teams documented the two inputs, base rate and date, as a single use case step while other teams split the input of the two values into multiple steps.

It was also clear that on some teams, no effort was made to make the level of use case detail included uniform across the entire use case model. Instead, use cases were assigned to individual team members and prepared as individual assignments with little or no peer review. Since the UUCW is a measure of the number of steps or interactions between an actor and the system, additional interactions can result in a given use case being classified as more complex during the UCP estimation process, inflating the UUCW.

The first step in determining a reasonable UUCW estimate must be *communication with the customer*. The use case model documents the developer's vision of the product's functionality, yet I, acting as the customer, received surprisingly few questions from the students regarding clarification of specific project requirements. Students developed their products in a vacuum, by choice, and as a result the teams were not really estimating the exact same product due to their respective interpretations of the initial requirements and occasional addition of *nice-to-have* features that were not explicitly required.

A *use case preparation standard* could also improve the UUCW estimation procedure. Such a standard should include specific criteria for combining or splitting of use cases for estimation purposes. Examples must be included in the standard to illustrate the desired level of detail to include in the model for estimation purposes. This level of detail is likely to vary with the scope of the product under consideration and is an area of research that I am currently pursuing.

One should also *perform a reality check on the initial UUCW estimate* derived from the use case model. Karner's UCP method allows you to determine the complexity of a use case from its text description, but as shown in Table 1, he also specified for each complexity category the number of objects required for implementation [1]. As a reality check, one can revisit the initial complexity assessment of each use case by examining the number of objects required in its implementation. If the two assessments disagree, you will have to make some decisions as to how to proceed. The brief summary of Karner's estimation technique presented in [1] provides no guidance on how to deal with this discrepancy. One approach would be to complete the calculation under the worst-case assumption, always sorting use cases into the most complex category identified by either of the two methods. One could also average the complexity weights indicated for those use cases in which the two methods produced different complexity assessments. You could also compare your UUCW estimate to that for any similar products developed by your organization to see if it seems reasonable.

“Some students felt it was easiest to rate a product for a particular factor if it happened to fall at one of the extremes, irrelevant, essential, or unknown, but they expressed a need for some criteria to differentiate the intermediate influence levels.”

UAW Calculations

Two types of errors were observed in the student UAW calculations: incorrect identification of the set of actors, and duplicate counting of actors in the calculation. In my view, the system had two actors, a hotel employee and a hotel manager, who directly interacted with the reservation system, and a Guest, who interacted indirectly with the system via the hotel employee or manager. Team B argued that the guest could be omitted for estimation purposes since the guest did not directly interact with the reservation system. Team C included all three actors in their estimate. Team A identified a single actor but classified that actor as complex since they implemented a graphical user interface. While inexperience at use case modeling contributed to these discrepancies, additional guidance was needed on the inclusion or exclusion of indirect actors in the estimate. Teams D-F, however, counted the same actor multiple times for each use case in which that actor was involved, inflating their resulting UAW. Several commercial and freeware UCP estimation software tools have reduced the possibility of this particular error by the way their developers have chosen to guide users through the UCP estimation process.

TCF and ECF Calculations

Both the TCF and ECF calculations evaluate to approximately 1.0 if the influence of all of their corresponding factors are unknown. Recall that the influence values are integers that range from 0 (irrelevant) to 5 (essential) with an influence value of 3 used to indicate that the impact of a particular technical or environmental factor is unknown. Some students felt it was easiest to rate a product for a particular factor if it happened to fall at one of the *extremes*, *irrelevant*, *essential*, or *unknown*, but they expressed a need for some criteria to differentiate the intermediate influence levels.

Fortunately, the TCF proved to be relatively insensitive to variations on the perceived impact of a single technical factor since the weights associated with each factor are small. The computed TCFs were all within 0.08 of the overall mean value 0.79. Some variation in ECF is to be

Table 7: *Summary of Supporting UCP Estimate Calculations*

Metric	Student Team Estimates						Mean All Teams
	A	B	C	D	E	F	
Total Use Cases	14	18	12	32	9	14	16.5
UUCW	105	120	70	310	90	160	142.5
Total Actors	1	2	3	10	9	14	6.5
UAW	3	6	6	18	18	28	13.17
TCF	0.78	0.71	0.80	0.87	0.72	0.87	0.79
ECF	1.06	0.83	0.92	0.97	0.68	0.76	0.87

COMING EVENTS

April 1-3
FOSE 2008
Washington, D.C.
www.fose.com

April 7-11
RSA Conference 2008
San Francisco, CA
www.rsaconference.com/2008/US/home.aspx

April 14-18
DoD Enterprise Architecture
Orlando, FL
www.afei.org/brochure/8a05/index.cfm

April 15-17
MRO Military 2008 Conference and Exhibition
Fort Lauderdale, FL
www.aviationweek.com conferences/milmain.htm

April 15-17
Software Test and Performance Conference
San Mateo, CA
www.stpcon.com

April 28-May 1
SATURN 2008
Pittsburgh, PA
www.sei.cmu.edu/architecture/saturn/2008

April 29-May 2

2008 Systems and Software Technology Conference
Las Vegas, NV
www.sstc-online.org

COMING EVENTS: **Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.**

expected since the ECF gauges people-related factors such as object-oriented development experience, application domain experience, motivation, etc.

PF Calculations

The average productivity, excluding Team D, was approximately 3.5 hours per use case point, which was significantly lower than Karner's nominal value of 20 hours per use case point. Karner's nominal value was determined from real-world products, which are likely to be significantly larger and more robust than those developed by the student teams (see reported labor in Table 5). Given the time constraints of the semester, the student products are essentially prototypes that lack adequate error handling, and in some cases, omit functionality identified in their use case model.

It is also important to remember that these PFs are derived from the labor hours reported by the student teams. Inaccurate reporting of labor hours on student projects does occur. In some cases, students forget to record their hours for the week and report an estimated labor value for that week. Students may also choose to inflate their reported hours in hopes of achieving a better grade on the assignment.

Mathematical Errors

While the mathematical computations required by the UCP estimation method are not conceptually difficult, errors did occur. The best way to minimize such errors is to *provide a software tool* that implements the UCP calculations correctly. While a spreadsheet estimation template would suffice, a dedicated UCP estimation tool that interfaced with your modeling tool would help prevent errors in execution of the estimation method itself, such as duplicate counting of actors in the UAW calculation, in addition to preventing the purely numeric errors.

Conclusions

The student team estimation data illustrates both the potential of the UCPs estimation method and some of the pitfalls that can be avoided. Excluding Teams D-F due to calculation errors, team productivity averaged approximately 3.5 hours per use case point on this semester-sized, four-person project. While this was significantly lower than that reported by Karner [1], the time constraints of the semester prevent the students from developing product-quality code.

Since the PF is determined experimentally, it is important to have a consistent method of producing a use case points estimate. The consistency of the presented estimates could have been improved by *better communication with the customer*, to prevent inclusion of unnecessary features, and by the use of a *use case preparation standard*. Such a standard must address the level of detail to be included in the description of each use case and include specific criteria for splitting or combining use cases. Commercial UCP *software tools* also help to prevent errors by automating many of the UCP calculations. One must also perform a *reality check* on any UCP estimate, using either the required objects count of the UCP method, a best-case/worst-case analysis, or your own historical data, to determine if the method has produced a reasonable estimate. Finally, one must keep in mind that the UCP estimation method is the focus of ongoing research worldwide so your estimation procedures should be reviewed periodically to incorporate the latest lessons learned.◆

References

1. Karner, Gustav. "Resource Estimation for Objectory Projects." Objective Systems SF AB, September 17, 1993.
2. Clemmons, Roy K. "Project Estimation With Use Case Points." CROSSTALK Feb. 2006.
3. Schach, Steven R. Object-Oriented & Classical Software Engineering. 6th ed. Boston: McGraw Hill, 2005.

About the Author

David J. Coe, Ph.D., is an assistant professor in the Department of Electrical and Computer Engineering at the University of Alabama in Huntsville. He currently teaches undergraduate and graduate courses in C++ programming, data structures, and software engineering, and he has consulted locally in the areas of software engineering and software process. Coe has an undergraduate degree in computer science from Duke University, a masters of science in electrical engineering, and a doctorate in electrical engineering from the Georgia Institute of Technology.

The University of Alabama, Huntsville
Department of Electrical and Computer Engineering
217-F Engineering BLDG
Huntsville, AL 35899
Phone: (256) 824-3583
E-mail: coe@ece.uah.edu