# Project Estimation With Use Case Points

Roy K. Clemmons
*Diversified Technical Services, Inc.*

*Software developers frequently rely on use cases to describe the business processes of object-oriented projects. Since use cases consist of the strategic goals and scenarios that provide value to a business domain, they can also provide insight into a project's complexity and required resources. This article provides an introduction to the Use Case Points method that employs a project's use cases to produce a reasonable estimate of a project's complexity and required man-hours.*

Use case modeling is an accepted and widespread technique to capture the business processes and requirements of a software application project. Since use cases provide the functional scope of the project, analyzing their contents provides valuable insight into the effort and size needed to design and implement a project. In general, projects with large, complicated use cases take more effort to design and implement than small projects with less complicated use cases. Moreover, the time to complete the project is affected by the following:

- The number of steps to complete the use case.
- The number and complexity of the actors.
- The technical requirements of the use case such as concurrency, security, and performance.
- Various environmental factors such as the development teams' experience and knowledge.

An estimation method that took into account the above factors early in a project's life cycle, and produced an estimate within 20 percent of the actual completion time would be very helpful for project scheduling, cost, and resource allocation.

The Use Case Points (UCP) method provides the ability to estimate the man-hours a software project requires from its use cases. Based on work by Gustav Karner [1], the UCP method analyzes the use case actors, scenarios, and various technical and environmental factors and abstracts them into an equation. Readers familiar with Allan Albrecht's "Function Point Analysis" [2] will recognize its influence on UCP; function point analysis inspired UCP.

The UCP equation is composed of three variables:

1. Unadjusted Use Case Points (UUCP).
2. The Technical Complexity Factor (TCF).
3. The Environment Complexity Factor (ECF).

Each variable is defined and computed separately using weighted values, subjective values, and constraining constants. The weighted values and constraining constants were initially based on Albrecht, but subsequently modified by people at Objective Systems, LLC, based on their experience with *Objectory* – a methodology created by Ivar Jacobson for developing object-oriented applications [3]. The subjective values are determined by the development team based on their perception of the project's technical complexity and efficiency.

Additionally, when productivity is included as a coefficient that expresses time, the equation can be used to estimate the number of man-hours needed to complete a project. Here is the complete equation with a *Productivity Factor* (PF) included:

**UCP = UUCP * TCF * ECF * PF**

The necessary steps to generate the estimate based on the UCP method are the following:

1. Determine and compute the UUCPs.
2. Determine and compute the TCFs.
3. Determine and compute the ECFs.
4. Determine the PF.
5. Compute the estimated number of hours.

Table 1: *Use Case Categories*

| Use Case Category | Description | Weight |
|---|---|---|
| Simple | Simple user interface. Touches only a single database entity. Its success scenario has three steps or less. Its implementation involves less than five classes. | 5 |
| Average | More interface design. Touches two or more database entities. Between four and seven steps. Its implementation involves between five and 10 classes. | 10 |
| Complex | Complex user interface or processing. Touches three or more database entities. More than seven steps. Its implementation involves more than 10 classes. | 15 |

## Sample Case Study

In the sections that follow, the UCP method is retroactively applied to a Web application developed by the author. This after-the-fact approach provides a practical way to establish a baseline PF for projects already completed. As described later, the PF helps determine the number of man-hours needed to complete the project.

### *UUCPs*

UUCPs are computed based on two computations:

1. The *Unadjusted Use Case Weight (UUCW)* based on the total number of activities (or steps) contained in all the use case scenarios.
2. The *Unadjusted Actor Weight (UAW)* based on the combined complexity of all the actors in all the use cases.

### UUCW

The UUCW is derived from the number of use cases in three categories: *simple*, *average*, and *complex* (see Table 1). Each use case is categorized by the number of steps its scenario contains, including alternative flows.

Keep in mind the number of steps in a scenario affects the estimate. A large number of steps in a use case scenario will bias the UUCW toward complexity and increase the UCPs. A small number of steps will bias the UUCW toward simplicity and decrease the UCPs. Sometimes, a

large number of steps can be reduced without affecting the business process.

The UUCW is calculated by tallying the number of use cases in each category, multiplying each total by its specified weighting factor, and then adding the products. For example, Table 2 computes the UUCW for the sample case study.

### UAW

In a similar manner, the Actor Types are classified as simple, average, or complex as shown in Table 3.

The UAW is calculated by totaling the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the products. Table 4 computes the UAW for the sample case study.

The UUCP is computed by adding the UUCW and the UAW. For the data used in Tables 2 and 4, the UUCP = 210 + 12 = 222.

The UUCP is *unadjusted* because it does not account for the TCFs and ECFs.

### TCFs

Thirteen standard technical factors exist to estimate the impact on productivity that various technical issues have on a project (see Table 5, page 20). Each factor is weighted according to its relative impact.

For each project, the technical factors are evaluated by the development team and assigned a *perceived complexity* value between zero and five. The perceived complexity factor is subjectively determined by the development team's perception of the project's complexity – concurrent applications, for example, require more skill and time than single-threaded applications. A perceived complexity of 0 means the technical factor is irrelevant for this project, 3 is average, and 5 is strong influence. When in doubt, use 3.

Each factor's weight is multiplied by its perceived complexity factor to produce the *calculated factor*. The calculated factors are summed to produce the *Technical Total Factor*. Table 6 (see page 20) calculates the technical complexity for the case study.

Two constants are computed with the Technical Total Factor to produce the TCF. The constants constrain the effect the TCF has on the UCP equation from a range of 0.60 (perceived complexities all zero) to a maximum of 1.30 (perceived complexities all five).

TCF values less than one reduce the UCP because any positive value multiplied by a positive fraction decreases in magnitude: 100 * 0.60 = 60 (a reduction of 40 percent).

TCF values greater than one increase the UCP because any positive value multiplied by a positive mixed number increases in magnitude: 100 * 1.30 = 130 (an increase of 30 percent).

*Since the constants constrain the TCF from a range of 0.60 to 1.30, the TCF can impact the UCP equation from -40 percent (.60) to a maximum of +30 percent (1.30).*

For the mathematically astute, the complete formula to compute the TCF is:

$$TCF = C_1 + C_2 \sum_{i=1}^{13} W_i * F_i$$

where,

**Constant 1 ($C_1$) = 0.6**
**Constant 2 ($C_2$) = .01**
**W = Weight**
**F = Perceived Complexity Factor**

For the rest of us, a more digestible equation is:

**TCF = 0.6 + (.01 * Technical Total Factor)**

For Table 6, the TCF = 0.6 + (0.01 * 19.5) = 0.795, *resulting in a reduction of the UCP by 20.5 percent.*

### ECFs

The ECF (see Table 7, page 21) provides a concession for the development team's experience. More experienced teams will have a greater impact on the UCP computation than less experienced teams.

The development team determines each factor's *perceived impact* based on its perception the factor has on the project's

Table 2: *Computing UUCW*

| Use Case Type | Description | Weight | Number of Use Cases | Result |
|---|---|---|---|---|
| Simple | Simple user interface. Touches only a single database entity. Its success scenario has three steps or less. Its implementation involves less than five classes. | 5 | 7 | 35 |
| Average | More interface design. Touches two or more database entities. Between four and seven steps. Its implementation involves between five and 10 classes. | 10 | 13 | 130 |
| Complex | Complex user interface or processing. Touches three or more database entities. More than seven steps. Its implementation involves more than 10 classes. | 15 | 3 | 45 |
| | | | Total UUCW | 210 |

Table 3: *Actor Classifications*

| Actor Type | Description | Weight |
|---|---|---|
| Simple | The actor represents another system with a defined application programming interface. | 1 |
| Average | The actor represents another system interacting through a protocol, like Transmission Control Protocol/Internet Protocol. | 2 |
| Complex | The actor is a person interacting via a graphical user interface. | 3 |

Table 4: *Computing UAW*

| Actor Type | Description | Weight | Number of Actors | Result |
|---|---|---|---|---|
| Simple | The actor represents another system with a defined application programming interface. | 1 | 0 | 0 |
| Average | The actor represents another system interacting through a protocol, like Transmission Control Protocol/Internet Protocol. | 2 | 0 | 0 |
| Complex | The actor is a person interacting via an interface. | 3 | 4 | 12 |
| | | | Total UAW | 12 |

| Technical Factor | Description | Weight |
|---|---|---|
| T1 | Distributed System | 2 |
| T2 | Performance | 1 |
| T3 | End User Efficiency | 1 |
| T4 | Complex Internal Processing | 1 |
| T5 | Reusability | 1 |
| T6 | Easy to Install | 0.5 |
| T7 | Easy to Use | 0.5 |
| T8 | Portability | 2 |
| T9 | Easy to Change | 1 |
| T10 | Concurrency | 1 |
| T11 | Special Security Features | 1 |
| T12 | Provides Direct Access for Third Parties | 1 |
| T13 | Special User Training Facilities Are Required | 1 |

Table 5: *Technical Complexity Factors*

success. A value of 1 means the factor has a strong, negative impact for the project; 3 is average; and 5 means it has a strong, positive impact. A value of zero has no impact on the project's success. For example, team members with little or no motivation for the project will have a strong negative impact (1) on the project's success while team members with strong object-oriented experience will have a strong, positive impact (5) on the project's success.

Each factor's weight is multiplied by its perceived impact to produce its *calculated factor*. The calculated factors are summed to produce the *Environmental Total Factor*.

*Larger values for the Environment Total Factor will have a greater impact on the UCP equation.*

Table 8 calculates the environmental factors for the case study project.

To produce the final ECF, two constants are computed with the Environmental Total Factor. The constants, "based on interviews with experienced Objectory users at Objective Systems" [1], constrain the impact the ECF has on the UCP equation from 0.425 (Part-Time Workers and Difficult Programming Language = 0, all other values = 5) to 1.4 (perceived impact all 0). Therefore, the ECF can reduce the UCP by 57.5 percent and increase the UCP by 40 percent.

*The ECF has a greater potential impact on the UCP count than the TCF.* The formal equation is:

$$ECF = C_1 + C_2 \sum_{i=1}^{8} W_i * F_i$$

where,

**Constant 1 ($C_1$) = 1.4**
**Constant 2 ($C_2$) = -0.03**
**W = Weight**
**F = Perceived Impact**

Informally, the equation works out to be:

**ECF = 1.4 + (-0.03 * Environmental Total Factor)**

Table 6: *Calculating the Technical Total Factor*

| Technical Factor | Description | Weight | Perceived Complexity | Calculated Factor (Weight* Perceived Complexity) |
|---|---|---|---|---|
| T1 | Distributed System | 2 | 1 | 2 |
| T2 | Performance | 1 | 3 | 3 |
| T3 | End User Efficiency | 1 | 3 | 3 |
| T4 | Complex Internal Processing | 1 | 3 | 3 |
| T5 | Reusability | 1 | 0 | 0 |
| T6 | Easy to Install | 0.5 | 0 | 0 |
| T7 | Easy to Use | 0.5 | 5 | 2.5 |
| T8 | Portable | 2 | 0 | 0 |
| T9 | Easy to Change | 1 | 3 | 3 |
| T10 | Concurrency | 1 | 0 | 0 |
| T11 | Special Security Features | 1 | 0 | 0 |
| T12 | Provides Direct Access for Third Parties | 1 | 3 | 3 |
| T13 | Special User Training Facilities Are Required | 1 | 0 | 0 |
| | | | Technical Total Factor | 19.5 |

For the sample case study, the author's software development experience resulted in a high ETF. The most significant negative factor was the author's lack of experience in the application domain.

For Table 8, the ECF = 1.4 + (-0.03 *26) = 0.62, *resulting in a decrease of the UCP by 38 percent.*

### Calculating the UCP

As a reminder, the UCP equation is:

**UCP = UUCP * TCF * ECF**

From the above calculations, the UCP variables have the following values:

**UUCP = 222**
**TCF = 0.795**
**ECF = 0.62**

For the sample case study, the final UCP is the following:

**UCP = 222 * 0.795 * 0.62**

**UCP = 109.42 or 109 use case points**

*Note for the sample case study, the TCF and ECF reduced the UUCP by approximately 49 percent (109/222*100).*

By itself, the UCP value is not very useful. For example, a project with a UCP of 222 may take longer than one with a UCP of 200, but we do not know by how much. Another factor is needed to estimate the number of hours to complete the project.

### PF

The PF is the *ratio of development man-hours needed per use case point.* Statistics from past projects provide the data to estimate the initial PF. For instance, if a past project with a UCP of 120 took 2,200 hours to complete, divide 2,200 by 120 to obtain a PF of 18 man-hours per use case point.

### Estimated Hours

The total estimated number of hours for the project is determined by multiplying the UCP by the PF.

**Total Estimate = UCP * PF**

If no historical data has been collected, consider two possibilities:

1. Establish a baseline by computing the UCP for previously completed projects (as was done with the sample case study in this article).

2. Use a value between 15 and 30 depending on the development team's overall experience and past accomplishments (Do they normally finish on time? Under budget? etc.). If it is a brand-new team, use a value of 20 for the first project.

After the project completes, divide the number of actual hours it took to complete the project by the number of UCPs. The product becomes the new PF.

*Since the sample case study presented in this article actually took 990 hours to complete, the PF for the next project is: 990/109 = 9.08*

## Industry Case Studies

From the time Karner produced his initial report in 1993, many case studies have been accomplished that validate the reasonableness of the UCP method.

In the first case study in 2001, Suresh Nageswaran published the results of a UCP estimation effort for a product support Web site belonging to large North American software company [4]. Nageswaran, however, extended the UCP equation to include testing and project management coefficients to derive a more accurate estimate.

While testing and project management might be considered non-functional requirements, nevertheless they can significantly increase the length of the project. Testing a Java 2 Enterprise Edition implementation, for example, may take longer than testing a Component Object Model component; it is not unusual to spend significant time coordinating, tracking, and reporting project status.

*Nageswaran's extensions to the UCP equation produced an estimate of 367 man-days, a deviation of 6 percent of the actual effort of 390 man-days.*

In a recent e-mail exchange with this author, Nageswaran said he had also applied the UCP method to performance testing, unit-level testing, and white box testing.

In the second case study, research scientist Dr. Bente Anda [5] evaluated the UCP method in case studies from several companies and student projects from the Norwegian University of Science and Technology that varied across application domains, development tools, and team size. The results are shown in Table 9.

*For the above studies, the average UCP estimate is 19 percent; the average expert estimate is 20 percent.*

Additionally, at the 2005 International Conference on Software Engineering, Anda, et al. [6] presented a paper that described the UCP estimate of an incremental, large-scale development project that was within 17 percent of the actual effort.

In the third case study, Agilis Solutions and FPT Software partnered to produce an estimation method, based on the UCP method that produces very accurate estimates. In an article that was presented at the 2005 Object-Oriented, Programming, Systems, Languages, and Applications conference, Edward R. Carroll of Agilis Solutions stated:

> After applying the process across hundreds of sizable (60 man-months average) software projects, we have demonstrated metrics that prove an estimating accuracy of less than 9 percent deviation from actual to estimated cost on 95 percent of our projects. Our process and this success factor are documented over a period of five years, and across more than 200 projects. [7]

*To achieve greater accuracy, the Agilis Solutions/FPT Software estimation method includes a risk coefficient with the UCP equation.*

## Conclusion

An early project estimate helps managers, developers, and testers plan for the resources a project requires. As the case studies indicate, the UCP method can produce an early estimate within 20 percent of the actual effort, and often, closer to the actual effort than experts and other estimation methodologies [7].

Moreover, in many traditional estima-

| Environmental Factor | Description | Weight |
|---|---|---|
| E1 | Familiarity With UML* | 1.5 |
| E2 | Part-Time Workers | -1 |
| E3 | Analyst Capability | 0.5 |
| E4 | Application Experience | 0.5 |
| E5 | Object-Oriented Experience | 1 |
| E6 | Motivation | 1 |
| E7 | Difficult Programming Language | -1 |
| E8 | Stable Requirements | 2 |

*Note: Karner's original factor, "Familiar with Objectory," was changed to reflect the popularity of UML.

Table 7: *Environmental Complexity Factors*

Table 8: *Calculating the Environmental Total Factor*

| Environmental Factor | Description | Weight | Perceived Impact | Calculated Factor (Weight*Perceived Complexity) |
|---|---|---|---|---|
| E1 | Familiarity With UML | 1.5 | 5 | 7.5 |
| E2 | Part-Time Workers | -1 | 0 | 0 |
| E3 | Analyst Capability | 0.5 | 5 | 2.5 |
| E4 | Application Experience | 0.5 | 0 | 0 |
| E5 | Object-Oriented Experience | 1 | 5 | 5 |
| E6 | Motivation | 1 | 5 | 5 |
| E7 | Difficult Programming Language | -1 | 0 | 0 |
| E8 | Stable Requirements | 2 | 3 | 6 |
| | Environmental Total Factor | | | 26 |

Table 9: *Use Case Studies*

| Company | Project | Use Case Estimate | Expert Estimate | Actual Effort | Deviation Use Case Estimate | Deviation Expert Estimate |
|---|---|---|---|---|---|---|
| Mogel | A | 2,550 | 2,730 | 3,670 | -31% | -26% |
| Mogel | B | 2,730 | 2,340 | 2,860 | -5% | -18% |
| Mogel | C | 2,080 | 2,100 | 2,740 | -24% | -23% |
| CGE and Y | A | 10,831 | 7,000 | 10,043 | +8% | -30% |
| CGE and Y | B | 14,965 | 12,600 | 12,000 | +25% | +5% |
| IBM | A | 4,086 | 2,772 | 3,360 | +22% | -18% |
| Student Project | A | 666 | | 742 | -10% | |
| Student Project | B | 487 | | 396 | +23% | |
| Student Project | C | 488 | | 673 | -25% | |

tion methods, influential technical and environmental factors are not given enough consideration. The UCP method quantifies these subjective factors into equation variables that can be tweaked over time to produce more precise estimates.

Finally, the UCP method is versatile and extensible to a variety of development and testing projects. It is easy to learn and quick to apply.

The author encourages more projects to use the UCP method to help produce software on time and under budget.◆

## References

1. Karner, Gustav. "Resource Estimation for Objectory Projects." Objective Systems SF AB, 1993.
2. Albrecht, A.J. Measuring Application Development Productivity. Proc. of IBM Applications Development Symposium, Monterey, CA, 14-17 Oct. 1979: 83.
3. Jacobson, I., G. Booch, and J. Rumbaugh. The Objectory Development Process. Addison-Wesley, 1998.
4. Nageswaran, Suresh. "Test Effort Estimation Using Use Case Points." June 2001 <www.cognizant.com/cogcommunity/presentations/Test_Effort_Estimation.pdf>.
5. Anda, Bente. "Improving Estimation Practices By Applying Use Case Models." June 2003 <www.cognizant.com/cogcommunity/presentations/Test_Effort_Estimation.pdf>.
6. Anda, Bente, et al. "Effort Estimation of Use Cases for Incremental Large-Scale Software Development." 27th International Conference on Software Engineering, St Louis, MO, 15-21 May 2005: 303-311.
7. Carroll, Edward R. "Estimating Software Based on Use Case Points." 2005 Object-Oriented, Programming, Systems, Languages, and Applications (OOPSLA) Conference, San Diego, CA, 2005.

## About the Author

**Roy K. Clemmons** is an employee of Diversified Technical Services, Inc. He has more than 20 years experience in software design and development. Currently, he is contracted to the Retrieval Applications Group at Randolph Air Force Base, Texas, where he works on the Virtual Military Personnel Flight system and the Retrieval Applications Web site.

**Diversified Technical Services, Inc.**
**403 E Ramsey STE 202**
**San Antonio, TX 78216**
**Phone: (210) 565-1119**
**E-mail: roy.clemmons.ctr@randolph.af.mil**

## MORE ONLINE FROM CROSSTALK

CROSSTALK is pleased to bring you this additional article with full text at <www.hill.af.mil/crosstalk/2006/02/index.html>.

### Hard Skills Simulations: Tackling Defense Training Challenges Through Interactive 3-D Solutions

Josie Simpson
*NGRAIN Corporation*

The defense industry today faces a number of challenges around skills training, primarily driven by an increased pace of operations, the growing need to cross-train technical personnel to meet mission objectives, and ever-shrinking training budgets. Combined, these challenges can be daunting; but they can be overcome through the insertion of advanced technologies in instructional programs. Until recently, the use of three-dimensional (3-D) in hard skills training was limited to high-end applications such as flight simulators. Today, new technologies have been introduced that remove the traditional barriers to 3-D, allowing interactive 3-D to be used in lower-end applications, including maintenance training. Hard skills simulations, most notably 3-D virtual equipment, provide an innovative new way to cost-effectively train students to standard in less time on maintenance procedures and repair tasks, while simultaneously helping to improve performance in the field through on-the-job training aids. The result is reduced costs and a higher level of preparedness, ultimately saving lives.