**Estimating With Use Case Points- Example**

If you’ve worked with use cases, you’ve probably felt there should be an easy way to estimate the overall size of a project from all the work that went into writing the use cases. There’s clearly a relationship between use cases and code in that complicated use cases generally take longer to code than simple use cases. Fortunately, there is an approach for estimating and planning with *use case points.* Similar in concept to function points, use case points measure the size of an application. Once we know the approximate size of an application, we can derive an expected duration for the project if we also know (or can estimate) the team’s rate of progress.

Use case points were first described by Gustav Karner, but his initial work on the subject is closely guarded by Rational Software. This article, therefore, primarily documents Karner’s work as describer by Schneider and Winters (1998) and Ribu (2001).

**Use Case Points**

The number of use case points in a project is a function of the following:

* the number and complexity of the use cases in the system
* the number and complexity of the actors on the system
* various non-functional requirements (such as portability, performance, maintainability) that are not written as use cases
* the environment in which the project will be developed (such as the language, the team’s motivation, and so on)

The basic formula for converting all of this into a single measure, use case points, is that we will "weigh" the complexity of the use cases and actors and then adjust their combined weight to reflect the influence of the nonfunctional and environmental factors.

Fundamental to the use of use case points is the need for all use cases to be written at approximately the same level. Alistair Cockburn (2001) identifies five levels for use cases: very high summary, summary, user goal, subfunction, and too low. Cockburn’s very high summary and summary use cases are useful for setting the context within which lower-level use cases operate. However, they are written at too high of a level to be useful for estimating. Cockburn recommends that user goal-level use cases form the foundation of a well thought through collection of use cases. At a lower level, subfunction use cases are written to provide detail on an as-needed basis.

If a project team wishes to estimate with use case points, they should write their use cases at Cockburn’s user goal level. Each use case (at all levels of Cockburn’s hierarchy) has a goal. The goal of a user goal-level use case is a fundamental unit of business value. There are two tests for the whether a user goal use case is written at the proper level: First, the more often the user achieves the goal, the more value is delivered to the business; Second, the use case is normally completed within a single session and after the goal is achieved, the user may go on to some other activity.

A sample user goal use case is shown in Figure 1. This use case is from a job posting and search site. It describes the situation in which a third-party recruiter has already posted a job opening on the site and now needs to submit payment for placing that ad.

Since this is not an article on use cases, I won’t fully cover all the details of the use case shown in Figure 1; however, it is worth reviewing the meaning of the Main Success Scenario and Extensions sections. The Main Success Scenario is a description of the primary successful path through the use case. In this case, success is achieved after completing the five steps shown. The Extensions section defines alternative paths through the use case. Often, extensions are used for error handling; but extensions are also used to describe successful but secondary paths, such as in extension 3a of Figure 1. Each path through a use case is referred to as a *scenario*. So, just as the Main Success Scenario represents the sequence of steps one through five, an alternate scenario is represented by the sequence 1, 2, 2a, 2a1, 2, 3, 4, 5.

Use Case Title: Pay for a job posting

Primary actor: Recruiter

Level: Actor goal

Precondition: The job information has been entered but is not viewable.

Minimal Guarantees: None

Success Guarantees: Job is posted; recruiter’s credit card is charged.

Main Success Scenario:

1. Recruiter submits credit card number, date, and authentication information.

2. System validates credit card.

3. System charges credit card full amount.

4. Job posting is made viewable to Job Seekers.

5. Recruiter is given a unique confirmation number.

Extensions:

2a: The card is not of a type accepted by the system:

2a1: The system notifies the user to use a different card.

2b: The card is expired:

2b1: The system notifies the user to use a different card.

2c: The card number is invalid:

2c1: The system notifies the user to re-enter the number.

3a: The card has insufficient available credit to post the ad.

3a1: The system charges as much as it can to the current credit card.

3a2: The user is told about the problem and asked to enter a second credit card for the remaining charge. The use case continues at Step 2.

Figure 1. A sample use case for pay for a job posting

**Unadjusted Use Case Weight**

If all of a project’s use cases are written at approximately the level of detail shown in Figure 1, it’s possible to calculate use case points from them. Unlike an expert opinion-based estimating approach where the team discusses items and estimates them, use case points are assigned by a formula. In Karner’s original formula, each use case is assigned a number of points based on the number of transactions within the use case. A transaction (at least when working with user goal-level use cases) is equivalent to a step in the use case. Therefore we can determine the number of transactions by counting the steps in the use case. Karner originally proposed ignoring transactions in the extensions part of a use case. However, this was probably largely because extensions were not as commonly used in the use cases he worked with during the era when he first proposed use case points (1993). Extensions clearly represent a significant amount of work and need to be included in any reasonable estimating effort.

Counting the number of transactions in a use case with extensions requires a small amount of caution. That is, you cannot simply count the number of lines in the extension part of the template and add those to the lines in the main success scenario.

In Figure 1, each extension starts with a result of a transaction, rather than a new transaction itself. For example, extension 2a ("The card is not of a type accepted by the system") is the result of the transaction described by step 2 of the main success scenario ("System validates credit card"). So, item 2a in the extensions section of Figure 1 is not counted. The same, of course, is true for 2b, 2c, and 3a. The transaction count for the use case in Figure 1 is then ten. You may want to count 2b1 and 2b2 only once but that is more effort than is worthwhile, and they may be separate transactions sharing common text in the use case.

Table 1 shows the points assigned to each simple, average, and complex use case based on the number of transactions. Since the use case we’re considering contains more than seven transactions it is considered complex.

|  |  |  |
| --- | --- | --- |
| **Use case complexity** | **Number of transactions** | **Weight** |
| Simple | 3 or fewer | 5 |
| Average | 4 to 7 | 10 |
| Complex | More than 7 | 15 |

Table 1. Use case weights based on the number of transactions

Repeat this process for each use case in the project. The sum of the weights for each use case is known as the *Unadjusted Use Case Weight*, or UUCW. Table 2 shows how to calculate UUCW for a project with 40 simple use cases, 21 average, and 10 complex.

|  |  |  |  |
| --- | --- | --- | --- |
| **Use case complexity** | **Weight** | **Number of use cases** | **Product** |
| Simple | 5 | 40 | 200 |
| Average | 10 | 21 | 210 |
| Complex | 15 | 10 | 150 |
| Total |  |  | 560 |

Table 2. Calculating Unadjusted Use Case Weight (UUCW) for a simple project