Technique: Use-case points

Extracted from Chapter 6: Managing time

Philippe Kruchten
May 2010

Origins

Use-case points are a very simple alternative to function points, where the proxy is the use-case model. The method was originally developed by Gustav Karner (1993) for the *Objectory* method, and then subsequently integrated in the *Rational Unified Process* (IBM, 2003; Kruchten, 1998; Smith, 1999) after Rational Software acquired the Objectory company in 1995.

Overview

From a use-case model, a simple counting method derives an initial, unadjusted, use-case point number UUCP, which is then adjusted to take into account:

(a) the *technical complexity* of the system: quality attributes, inherent complexity, and

(b) the development *environment*: process, tools, people.

giving an (adjusted) use-case points number UCP, from which effort is then derived.

So:

$$UCP = UUCP \times TCF \times EF$$
$$Effort = UCP \times PF$$

where

UCP: Use-case points, estimate of the size of the software system
UUCP: Unadjusted Use Case Points, derived form the use-case model
TCF: Technical Complexity factor
EF: Environment Factor
PF: Productivity Factor

Step1. Unadjusted Use Case Points

First we will compute the number of Unadjusted Use-Case Points, UUCP, which has one major component deriving from the use cases UUCW, and a minor component from the actors involved, UUAW.

UUCP = UUCW + UUAW

Use cases are not all equal in terms of effort, so we sort them in 3 categories, with different weights, as show on table 6-1.

.

Table 6-1: Three categories of use cases and their weight

Use-Case Category	Category description	Weight
Simple	Simple UI. Main flow of event has 3 steps or less. Less than 5 classes involved in implementation. Affects only one database entity.	5
Average	More elaborate interface. Affects two or more database entities. Main flow of events has 4 to 7 steps. Implementation involves between 5 and 10 classes.	10
Complex	Complex UI, or complex processing. Three or more database entities are affected. More than 7 steps. More than 10 classes involved.	15

Example:
Let us illustrate the technique with simple system for which we have a use-case model.

This use-case model has 23 use cases; 7 simple ones, 13 average, and 3 complex ones.

Table 6-2: uses cases in our example

UC Category	Weight	Number	Points
Simple	5	7	35
Average	10	13	130
Complex	15	3	45
		UUW =	210

We proceed similarly for the Actors, where we distinguish 3 main types. See table 6-3.

Table 6-3: Three categories of actors and their weight

Actor type	Description	Weight
Simple	The actor represents another system with a defined application programming interface (API)	1
Average	The actor represents another system interacting through a protocol such as TCP/IP	2
Complex	The actor is a person interacting via a graphical user interface (GUI).	3

Our example use-case model involves 4 actors, all humans; UUAW will therefore be 12
We can now compute the total Unadjusted Use-Case Point for this system:

UUCP = UUCW + UUAW = 210 + 12 = 222.

Step 2. Adjustments

Then we introduce the two *adjustment* factors, which play roughly the same role as the cost drivers in COCOMO. There are 13 Technical Complexity Factors; see table 6-4.

Table 6-4: Technical complexity factors

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	Direct access to 3rd parties	1
T13	Special user training facilities	1

For each of the 13 factors, you assign a value indicating the perceived impact of that aspect from zero (no impact on the project) to 5 (strong impact on the project), using the value 3 as default, and you compute the TCF using the formula:

$$TCF = 0.6 + 0.01 \times \Sigma W_i * F_i$$

For our example, let us assume that ease of use is paramount, while reusability, portability, ease of installation, etc. shall play little role. See 6.5.

Table 6-5: Complexity factors in our example

Technical factor	Description	Weight	Perceived impact	Impact
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	Portability	2	0	0
T9	Easy to change	1	3	3
T10	Concurrency	1	0	0
T11	Special security features	1	0	0
T12	Direct access to 3rd parties	1	3	3
T13	Special user training facilities	1	0	0
			Total technical factor	19.5

As a result the Technical Complexity Factor is:

$$TCF = 0.6 + 0.01 \times 19.5 = 0.795$$

Notice that the maximum range of TCF is 0.60 to 1.30, meaning it can affect the Use-Case Point from minus 40 % to plus 30% (Clemmons, 2006).

The Environment Factor proceeds similarly; see table 6-6. And the factor EF is given by the formula:

$$EF = 1.4 - 0.03 \times \Sigma W_i * F_i$$

Table 6-6: Environment factors

Environment factor	Description	Weight
E1	Familiarity with the process	1.5
E2	Use of part-time workers	-1
E3	Analyst capability	0.5
E4	Application experience	0.5
E5	OO Experience	1
E6	Motivation	1
E7	Difficult programming environment	-1
E8	Stable requirements	2

For our example, assuming a full dedicated and super-competent team, we will have the factors shown in table 6-7.

Table 6-7: Environment factors for our example

Environment factor	Description	Weight	Perceived impact	Factor
E1	Familiarity with the process	1.5	5	7.5
E2	Use of part-time workers	-1	0	0
E3	Analyst capability	0.5	5	2.5
E4	Application experience	0.5	0	0
E5	OO Experience	1	5	5
E6	Motivation	1	5	5
E7	Difficult programming environment	-1	0	0
E8	Stable requirements	2	3	6
				26

This gives us

$$EF = 1.4 - (0.03 \times 26) = 0.62$$

Now we can compute the size in Use-case points of this system

TCF = 0.6 + 0.1 x 19.5 = 0.795
EF = 1.4 – 0.03 * 26 = 0.62
UCP = UUCP * TCF * ECF

so

UCP = 222 * 0.795 * 0.62 = 109.4 use-case points.

Notice that EF has a range of 0.40 to 1.70 and therefore has a wider potential impact on the final UCP, from minus 60% to plus 70%.

Step 3. Effort estimate

To convert this adjusted size to an effort, we need now some estimate of the productivity, that is the effort it takes to develop one use-case point.

The best approach is certainly to calibrate this PF Productivity factor on *your own* projects, but how do you do without? Or to get started? Various authors have proposed, based on their data, some values, ranging from 15 to 30 person-hours per use-case point. Clemmons (2006) suggests to start with 20 for a first project.

Let us be conservative and use 28 person-hour per use-case point, so

Effort = 110 * 28 = 3,080 person-hours,

which in many places correspond to 305 person-days, or about 16 person-months.

This looks like magic, black art! There are issues though with this technique, beyond the calibration of the productivity factor noted above, which are well discussed by Mike Cohn (2005): “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. So if you wish to estimate with use-case points, you should have 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 whether a user goal- use case is written at the proper level:

1. the more often the user achieves the goal, the more value is delivered to the business;
2. 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.”

Bibliography

Clemmons, R. K. (2006). Project Estimation With Use Case Points. *CrossTalk: The Journal Of Defense Software, 19*(2), 18-22.

Cockburn, A. (2001). *Writing Effective Use Cases*. Boston: Addison-Wesley.

Cohn, M. (2005). Estimating With Use-Case Points. *Methods and Tools,* (Fall 2005), 2-13. Retrieved from http://www.methodsandtools.com/archive/archive.php?id=25

IBM. (2003). Rational Unified Process (Version 2003). Cupertino, CA: IBM Rational Software.

Karner, G. (1993). *Resource Estimation for Objectory Projects*. Kista, Sweden: Objective Systems AB.

Kruchten, P. (1998). *The Rational Unified Process--An Introduction* (1 ed.). Boston, MA: Addison-Wesley.

Smith, J. (1999). *The Estimation of Effort Based on Use Cases* (No. TP-171). Cupertino, CA: Rational Software.