

Software cost estimation using use case points: Getting use case transactions straight

Remi-Armand Collaris
Eef Dekker

March 15, 2009

from The Rational Edge: The use case points method is a useful model of estimating effort and cost on software development projects -- provided you can appropriately specify and count use case transactions. This article explains how (and how not) to count transactions for estimation purposes using this model.



An important part of the decision-making around starting a new

software development project is what it will cost. Estimating these costs has troubled system analysts, project managers, and software engineers for decades. The first hurdle is getting the scope of the project right. What should the system be able to do? Capturing functional requirements in use cases has helped considerably in communicating requirements in a form that is understandable for users and other domain experts. Early in the project a use case model is made, containing a list of all actors (users or external systems) and use cases in the system, their names, and a brief description. Capturing this information makes it easier to reach agreement on the size of the system early in the project.

The use case points method, which we'll sketch below, is a promising estimation method that fits in nicely with the use case approach to describing requirements. At its basis lies the concept of a use case transaction, the smallest unit of size measurement. There are, unfortunately, many diverging assumptions on the notion of a use case transaction.

In this article we take a look at some of these views and how well they work in practice. We start with a sketch of the use case points method, followed by a discussion on what definition of a use case transaction works best. We also show how use case transactions are connected to other concepts around use cases. We conclude with a discussion on how (not) to count them.

Use case points

The use case points method is a well-documented approach for estimating software development activities.¹ No estimation method, however, should be used just by itself, but needs to be balanced by other methods.² Here we focus on use case points. Figure 1 presents its main ideas.³ At its basis lies the use case model, which consists of actors and use cases. The number and weight of the use cases identified is the most important component in the calculation of the so-called unadjusted use case points. The size of a system is calculated from the unadjusted use case points by adjusting them with the technical complexity factor, obtained from a consideration of the system's technical properties.

Once you have an estimation of a system's size, you can start to think about calculating the effort. You do this by calculating the environmental factor (EF) from the team's qualifications and other environmental influences. A very important environmental factor is the stability of requirements. You should also look at how many hours per use case point (H) are needed. Finally, add the supplementary effort (SE) not accounted for in the use case points model (such as project management hours, integration testing) and the effort estimation is complete.

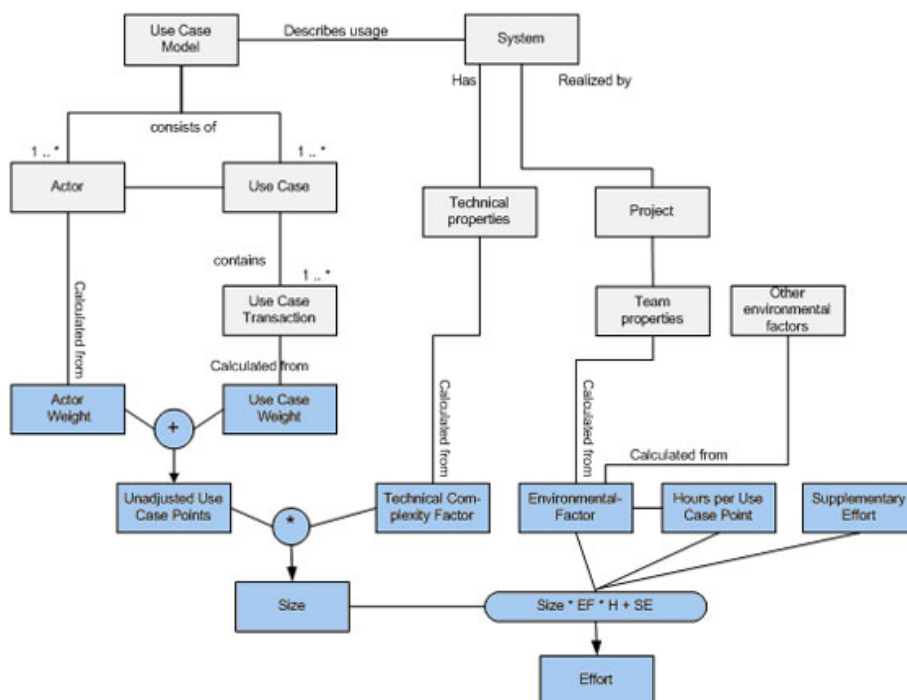


Figure 1: Main concepts of the use case points method

The weight of a use case is determined by the number of different use case transactions in the interaction between the actor and the system to be built.

According to the use case points method, the criteria to assign a weight to a use case are:

- Simple use case -- 1 to 3 transactions, weight = 5

- Average use case -- 4 to 7 transactions, weight = 10
- Complex use case -- more than 7 transactions, weight = 15

Hence, assumptions about the nature of a transaction and the strategy used to count transactions highly influence your estimation.

What is a use case transaction?

The concept of a (use case) transaction helps to deal with the variation in length and conciseness typical of use case descriptions. Use case specifications can be tersely written, or be rather verbose/detailed, depending on the use case template used, the approach adopted, the business context involved, or the personal taste of the Requirements Specifier. The number of steps in a use case flow, which describes the interaction between an actor and the system, can also vary widely both across and within scenarios. You can test for "sameness of size" by detecting and counting the use case transactions that are involved in your use case specifications. If two use case specifications have the same number of unique transactions, they have the same size.

A use case transaction is a "round trip"

Ivar Jacobson, inventor of the use case, describes a use case transaction as a "round trip" from the user to the system back to the user; a transaction is finished when the system awaits a new input stimulus.⁴ In other words, in one transaction the actor performs some action which is input for the system. Then the system reacts; i.e., it processes the input and returns the result to the Actor. A new transaction starts when the actor reacts to the result, which in turn is input for the system.

A use case transaction is not always a use case step

Jacobson's statement also implies that a use case transaction is not by definition "a step in the use case flow." This is only true if a step in a use case flow itself consists of a "round trip." Although some approaches on writing use cases prescribe this alternate method of delineating use case transactions, it is by no means the standard way.⁵

A use case transaction is not a "stimulus" as such

Some authors suggest that "the existence of a stimulus performed by an actor is what defines a transaction."⁶ Although a transaction starts with a stimulus (the actor does something that triggers the system), the stimulus itself is not the complete transaction. Suppose you have a use case description that reads:

```
The user selects an X.
...
(n)The user submits.
...
```

Then it is unclear whether the system reacts to the stimuli in steps (1) and (n) in one reaction, or whether the system instead reacts to steps (1) and (n) separately. Hence, two stimuli could make

up one transaction or two. This depends not on the stimuli but on the combination of stimulus and response.

A use case transaction is not a database activity

In many discussions on the Web, you find a use case transaction defined as "a set of activities, which is either performed entirely, or not at all."⁷ This definition sounds like that of a transactional mechanism in a database management system, in which a step can be rolled back if it is not performed correctly. In our experience, this is not a good way to isolate one piece of content from another in a use case description. It may give rise to the idea that a transaction is somehow related to a read or write action on a database. However, it is quite possible that in a round trip, the system does not have to consult the database at all. There may not even be a database involved, or data may come from outside the system. Thus it is not appropriate to conclude that use case transactions are necessarily linked to transactions on a database.


A use case transaction is not a system step

The response of the system in a use case transaction may be written as one step. On the surface, one could get the impression that a use case transaction just *is* a system step. A system step, however, is not a good basis for delineating a use case transaction, for it depends on the granularity of the description of a flow how many steps you count. Moreover, system steps alone don't say much about the interaction taking place between an Actor and the system. In other words, your estimation should be based on transactions being "round trips," not system steps.

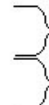
An example: complex user interfaces

The "round trip" approach to use case transactions shows its value in the case of estimating the complexity of user interfaces. Our example to follow stems from a job portal project, in which a job search machine was designed. In the early estimation based on the use case model (Survey), the job search interface was regarded as simple; it was expected that the user would select search items from a couple of drop-down menus and then fire his selection. However, in the user sessions dedicated to producing the use case specification it became obvious that the usability of the application would be enhanced if the system could respond to selections already made, and change the content of subsequent drop-down menus. In other words, what was regarded to be one transaction turned out to be two.

Here is the first draft of the use case specification:

- (1) The user selects an X and Ys and submits.
 - (2) The system searches for hits and shows the results.
- 
- Transaction 1

This text was expanded as follows:

- (1) The user selects an X.
 - (2) The system shows the Y s connected with this X.
 - (3) The user also selects one or more Y s and submits.
 - (4) The system searches for hits and shows the results.
- 
- Transaction 1
- Transaction 2

You see the two "round trips" clearly here. With the use case specification as testimony, it became easy to see the rationale for adjusting the initial estimation.

Keep use case transactions at the appropriate level

If a use case transaction is a stimulus followed by a system response, would then just about anything not count as a transaction? For example, if I type a character 'K' on my keyboard, this is a stimulus, with the system responding to it by producing a few pixels on the screen that are identifiable as 'K'. So isn't the definition we've been recommending then much too narrow?

No, it is not. This objection shows that you should interpret use case transactions at the same level at which the use case itself is meant to be interpreted. Nowadays, you typically would not be interested in the phenomenon of typing a character, which then appears somewhere on the screen. You just take it for granted; you don't have to build something in the system to produce this result. However, if your context is a description of the interaction of a keyboard module and a graphical renderer, such a use case transaction makes perfect sense.

How to count transactions

Now that you've seen a clear rationale for deciding what is and what is not a use case transaction, let's examine some of the challenges of counting transactions within use cases. As stated above, the weight of a use case is determined by the number of different use case transactions it contains. But exactly when does the system's reaction to a stimulus count as different?

Use case transactions and flows

Let us begin by examining the search flow of the job portal introduced above. If the actor looks for a job in the category of "Java," he selects Java and the system will search its database for all jobs in that category. When the actor looks for a job in the category of ".Net," he selects .Net and the system will search its database for all jobs in that category. Are these two different transactions? Clearly not. The use case specification itself is abstract or generic, in the sense that you don't expect different flows for different search terms. This is just a difference in instantiation. However, you would expect different flows for, say, a search using predefined categories or a free-format text search.

Handling exceptions, on the other hand, is a grey area. Suppose you have an input screen with seven fields, all of them with different constraints. You have a date field, a postal code, a field whose input is conditional upon the content of another, and so on. Each check may be described in a separate flow, and hence may count as at least one transaction. Alternatively, a generic exception flow may be provided. This presupposes that there is a framework in place in which several exception types are handled easily. In this case, you should count the flow as one transaction.

Use case transactions, being round trips, should be expected everywhere in the use case. Since a use case specification has at least a basic flow, it also has at least one transaction. A flow without a transaction is not meaningful, for then either the system would do something without stimulus, or the actor would provide one or more stimuli without it being clear what the system's reaction would be.

There are nearly always flows that describe handling an exception (hence, "exception flows"). Every exception flow contains at least one transaction. The same is true for an alternative flow; here you have at least one transaction per alternative flow. It may be the case that you have to look in the basic flow to see the stimulus of the transaction in your alternative flow; this depends on your specific guidelines for detailing a use case.

This gives us an indication of the minimum amount of use case transactions in any use case specification: there are at least as many transactions as there are flows.⁸

Showing and (not) counting

Given the ability to identify your use case transactions, do you have to take each and every transaction equally seriously? Our strategy is to *show* each and every transaction (if applicable), but sometimes not to *count* them in weight. This strategy is more straightforward to follow than just to ignore transactions if they seem "too light." It is also easier to adjust the original estimation, if necessary.

In this way, you are able to show the value of frameworks. If a use case counts ten transactions, but only seven of them need effort and three "follow" from the framework, that use case is average rather than complex. Table 1 shows an example.

Table 1: Transactions counted across hypothetical use cases

| Use case | # Transactions | # Counted | Reason | UC Weight |
|----------------------|----------------|-----------|-----------|-----------|
| 1 Apply for job | 4 | 3 | | Simple |
| 2 Find job | 3 | 3 | | Simple |
| 3 Assess application | 10 | 7 | framework | Average |

Many system steps could well be a new use case

Is there no way to account for the difference between many system steps implied by one use case transaction versus just one system step? Your intuition says that it takes more effort to build six system steps than one. Indeed, we completely agree. However, you should not try to solve this little puzzle by counting system steps as transactions, but by isolating the functionality involved in these extra system steps. If you have an obvious heap of functionality, then probably it is a use case in itself. Be careful not to promote just any heap of functionality to the status of 'use case' -- that would be functional decomposition -- but apply the following rule: Candidate use cases must always have a clear goal, matching the interest of at least one stakeholder (not necessarily identical with the actor).⁹

An example could be the use case "Generate annual balance." In the course of this use case several reports are generated, each being of interest to a specific stakeholder. There are several system steps involved in the generation of each report. Defining a separate use case for each report helps you to find the right stakeholder -- and not to bother other stakeholders. In this way, we are able to provide a more finegrained estimation.

Batch jobs

If it is true that a use case should also be used in the absence of user interaction (and we have good experiences in doing so), then how can you apply the concept of a transaction as a round trip? Well, frankly, it does not apply here. You need other ways to estimate the weight of such use cases. Mostly, this is done by expert estimation. Table 2 shows how this could look in your spreadsheet.

Table 2: Transaction count with batch job added

| use case | # Transactions | # Counted | Reason | UC Weight |
|--------------------------------|----------------|-----------|-----------------|-----------|
| 1 Apply for job | 4 | 3 | | Simple |
| 2 Find job | 3 | 3 | | Simple |
| 3 Assess application | 10 | 7 | framework | Average |
| 4 read flat file into database | -- | -- | Expert estimate | Complex |

If a batch job clearly is much bigger than a complex use case, it may well be that it serves more than one goal, and that the job should therefore be split up into more use cases, each serving its own goal of being of interest to at least one stakeholder. This is a mechanism applicable to any use case that seems to be significantly more complex than what you generally deem "Complex" (see Table 2). If you cannot find a good reason to split up the batch job, you can revert to the category of "supplementary effort" mentioned in Figure 1.

Very complex use cases

Some authors see a difficulty in the use case points method because there is no distinction between a complex use case of, say, eight transactions, and a complex use case of sixteen transactions. In our experience, use cases consisting of more than twelve transactions serve more than one goal. So, they are a sign of a problematic use case model. In other words, it is worthwhile to consider a new use case if at any point you have a use case of more than twelve transactions.¹⁰

Counting use case transactions early in the project

Counting transactions is easy when all use case specifications are written. On the other hand, estimation is expected early in the project. At that point, you only have the use case model, with a brief description per use case. In order to envision the flows that will make up the use cases and the use case transactions they involve, you need experience. If you don't have this experience yourself, don't hesitate to consult colleagues who have worked with similar systems and contexts. Start by creating a spreadsheet like the one in Table 2, and fill in the envisioned transactions. This will form a basis for managing the scope of the use cases, for you will be able to explain in detail which use cases involve more transactions than initially envisioned by the customer.

Conclusion

In order to apply the use case points method to the estimation of software effort estimation, it is important to have a good idea of its basic constituents. The notion of a use case transaction is such a constituent, and it is best taken to be a round trip, from the actor-initiated stimulus to the system's response. The transaction is finished if the system awaits a further stimulus.

Working with this notion, we have made some suggestions on how and when to count transactions. While this is more an art than a science, applying these recommendations along with common sense and experience can help you make more accurate effort and costs estimates earlier in a project.

References

- [1] Jacobson, Ivar et al., *Object-Oriented Software Engineering. A Use Case Driven Approach*, revised printing, Addison-Wesley 1993.
- [2] Cockburn, Alistair, *Writing Effective Use Cases*, Addison-Wesley, 2001.
- [3] Ribu, Kirsten, "Estimating Object-Oriented Software Projects with Use Cases", MSc Thesis Oslo 2001.
- [4] Övergaard, Gunnar and Karin Palmkvist, *Use Cases: Patterns and Blueprints*. Addison- Wesley 2005.
- [5] Mohagheghi, Parastoo, Bente Anda and Reidar Conradi, "Effort estimation of Use Cases for incremental large-scale software development", *International Conference on Software Engineering (ICSE)*, 2005, pp. 303 -- 31.
- [6] Laird, Linda M., and M. Carol Brennan, *Software Measurement and Estimation: A Practical Approach*. Wiley-Interscience 2006.
- [7] Robiolo, Gabriela and Ricardo Orosco, "Employing Use Cases to early estimate effort with simpler metrics", *Innovations in Systems and Software Engineering*, Volume 4, Number 1, April 2008 , pp. 31-43.
- [8] Issa, Ayman, Mohammed Odeh and David Coward, "Software Cost Estimation Using Use-Case Models: a Critical Evaluation", *Information and Communication Technologies*, 2006. ICTTA '06. 2nd Volume 2, pp. 2766-2771.
- [9] Vinsen, Kevin, Diane Jamieson and Guy Callender, "Use Case Estimation -- The Devil is in the Detail", *12th IEEE International Requirements Engineering Conference (RE'04)*, 2004, pp. 10-15.
- [10] Braz, Marcio Rodrigo and Silvia Regina Vergilio, "Software Effort Estimation Based on Use Cases", *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC '06)*, 2006, pp. 221-228.
- [11] Diev, Sergey, "Use cases modeling and software estimation: Applying Use Case Points", *ACM Software Engineering Notes*, Volume 31, Number 6, November 2006.
- [12] Anda, Bente, Endre Angelvik and Kirsten Ribu, "Improving Estimation Practices by Applying Use Case Models", *Profes 2002, LNCS 2259*, pp. 383-397.
- [13] Bittner, Kurt, and Ian Spence, *Use Case case Modeling*. Pearson Education 2003.

- [14] Kusumoto, Shinji, et al., "Estimating Effort by Use Case Points: Method, Tool and Case Study", *Proceedings of the 10th International Symposium on Software Metrics (METRICS'04)*, 2004.
- [15] Koirala, Shivprasad, "How to Prepare Quotation Using Use Case Points", The Code Project, December 2004.
- [16] Probasco, Leslee, "Dear Dr. Use Case: What About Function Points and Use Cases?", *The Rational Edge*, August 2002.

Notes

1. Detailed descriptions, spreadsheets, and tools are available on the Internet and in several publications; for example: [6], [3], [12].
2. See [6] for an overview of estimation methods.
3. Adapted from a similar figure found in Diev [11].
4. [1], p. 127; compare also [2], p. 93-94.
5. [2], p. 119-127.
6. [7], p. 35.
7. [3], p. 20, [14], section 2.1, [15].
8. Diev [11] sees the possibility of two (or more?) use case scenarios in one use case transaction. He says, "...the use case transaction 'Purchase financial product' may contain a scenario for successful purchase and another one for failed one [sic]." We do not think this is a good idea, for then the relation between transactions and scenarios becomes unclear. The "successful purchase" scenario consists of at least one stimulus and one response. The "failed purchase" scenario consists of the same stimulus as in the success scenario, but with a different system response. Therefore, this constitutes two transactions, not one.
9. See [4], p. 36-37.
10. Robiolo and Orosco try to eliminate the problem of how to count very complex use cases altogether. They do not relate use case transactions to complexity of use cases, but simply add all transactions found in use cases and directly calculate the size of an application from the amount of transactions [7], p. 35. This sounds promising, but as far as we know, substantial investigation of the applicable rules has yet to be done. At the moment, we prefer to use the use case points method. Further, in order to maintain interoperability, we do not wish to alter its basis, as is sometimes proposed; for example: changing the transaction/complexity ratio ([5], table 3); making additions to it (use case size points, Fuzzy use case size points [10]); or changing transactions for "key scenarios" [16].

© Copyright IBM Corporation 2009
(www.ibm.com/legal/copytrade.shtml)

Trademarks

(www.ibm.com/developerworks/ibm/trademarks/)