Metrics for Use Cases: A Survey of Current Proposals

Beatriz Bernárdez[1], Amador Durán[1] and Marcela Genero[2]

[1] Department of Computer Languages and Systems
University of Seville
Avda. Reina Mercedes, s/n. 41012 Sevilla (Spain)
{beat,amador}@us.es

[2]ALARCOS Research Group, Department of Computer Science
University of Castilla-La Mancha
Paseo de la Universidad, 4. 13071 Ciudad Real (Spain)
Marcela.Genero@uclm.es

1 Introduction

In this chapter, the current state–of–the–art of use case metrics is presented. Before describing the different proposals, the concept of *use case* itself is discussed. This discussion is necessary because of the different use case approaches that have been proposed since the original work by Jacobson et al. (1992) was published. These approaches vary from use cases expressed as informal, plain prose to extremely detailed message sequence diagrams, causing confusion about the concept and role of use cases in software development. This confusion has been increased by the ambiguity of the use–case specification in the UML standards (OMG, 2003), especially by the so–called *use case relationships*. Obviously, the concrete form in which use cases are used and expressed dramatically affects their measuring, as noted in Henderson-Sellers et al. (2002). Therefore, use case metrics must be always understood in the context of a specific use case format and purpose.

After introducing the reader to the concept of use case in the next section, several proposals for use case metrics are discussed. The proposals are grouped depending on their measurement goal. In section 3, use case metrics for project estimation are presented, including proposals by Karner (Schneider and Winters, 1998), Marchesi et al. (1998) and Smith (1999). In section 4 use case metrics for improving the requirements engineering process are discussed, including proposals by Saeki (2003) and Bernárdez et al. (2004). Finally, in section 5 the conclusions and a summary of the main proposals are presented.

The reader must always have in mind that the metrics proposals discussed in this chapter are still in a very early stage. Most of them are only initial proposals that have not been neither theoretically not empirically validated yet, as commented in the conclusions section.

2 The concept of use case

Use cases are a scenario–based technique initially proposed by Jacobson et al. (1992) that can be used for different purposes in software development, especially during requirements engineering. As a scenario–based technique, use cases tell *stories* describing interactions between some so–called *actors*, i.e. people or other systems with some goal to be achieved, and a *system under discussion* (SuD) providing some services needed to achieve actors' goals. The SuD has not necessarily to be a software system, it may be a computer–based information system encompassing hardware, software and people (see section *Dimensions of use cases* for details on *use case scoping*).

2.1 Roles of use cases in requirements engineering

As commented in Cockburn (2001), one of the best books on use cases, they can play different roles in the requirements engineering process, the following being the most usual:

- As a mean of understanding and describing current business processes, where they are called *business use cases* by some authors like Cockburn (2001) or Leffingwell and Widrig (2000), or simply *scenarios* as in Leite et al. (2000).

- As a mean of focusing discussion about the requirements of the system to be built, but not to be the requirements description, i.e. as a requirements elicitation technique but not as a requirement documentation or specification technique. In this case, use cases are eventually transformed into lists of *typical* functional requirements.

- As part of the functional requirements of the system to be built, which is probably the role they play more often. This is the main purpose of use cases as described in the latest UML specification (OMG, 2003) and in other publications (Cockburn, 2001; Leffingwell and Widrig, 2000; Schneider and Winters, 1998).

  Notice that, as stated by Cockburn (2001), "*they really are requirements but they are not all of the requirements*". Other kinds of requirements like information requirements, business rules or non–functional requirements cannot be expressed as use cases but must be part of any complete requirements specification.

At the moment of writing, use cases are the most popular requirements elicitation technique in software industry[1] and they are becoming an actual alternative to typical specifications of functional requirements composed of hundreds of sentences starting with "*the system shall*". As discussed in Cockburn (2001), use cases can be considered as *contracts for behavior*, thus rising Meyer's *contract* concept from programming to requirements.

All of the analyzed proposals for use case metrics assume that the measured use cases are part of a system or software requirements specification. We will also make the same assumption in the rest of this chapter for the sake of simplicity.

[1] The interested reader can see the article by Weidenhaput et al. (1998) for an excellent survey on how use cases are applied in European software industry.

2.2 Dimensions of use cases

Other criteria for classifying use cases apart from their purpose are their *scoping*, *goal level* and *visibility*, as proposed in Cockburn (2001) and graphically depicted in figure 1 as three orthogonal dimensions.

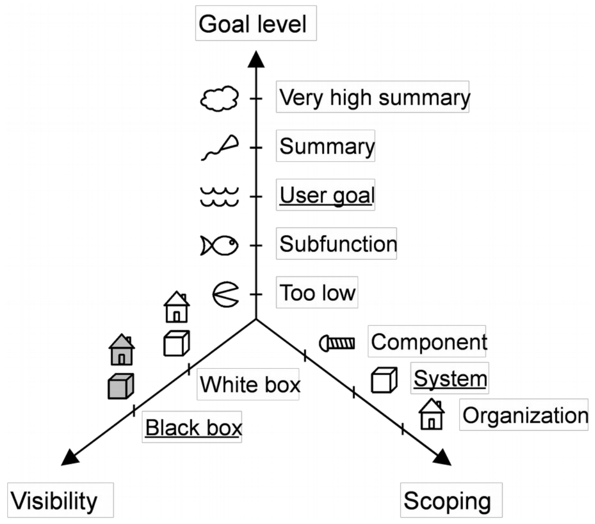


Figure 1. Cockburn's dimensions of use cases

2.2.1 Scoping

The scoping of a use case can be at an *organizational level* if describes *stories* between actors and organizations, which is the usual scope of the previously mentioned *business use cases*. The scoping is at a *system level* or *component level* if actors interact with a computer–based system or with a subsystem or a component of a computer–based system. The two latter scoping levels are the usual when writing system or software requirements specifications.

2.2.2 Goal level

The goal level of a use case indicates its level of abstraction with respect to user goals. For expressing the goal level, Cockburn proposes a metaphor in which height relative to sea level resembles goal level. The sea level corresponds to the *user goal* level, which is the level at which most use cases are usually described. Those higher–level use cases in which interactions are themselves user goals are said to be at a *summary* or *very high summary* level, depending on their level of abstraction. Since these use cases are *above the sea level*, they are represented by a kite (*summary*) or by a cloud (*very high summary*)[2]. User–goal–level use cases are usually performed in no more than a few minutes (what Cockburn calls a *single–sitting*), whereas summary use cases requires the completion of several user goals to

[2] A similar classification is described in Regnell et al. (1996), where *environment*, *structure* and *event* are proposed as levels of use case detail.

be performed and can take longer to complete.

2.2.3 Visibility

The visibility of a use case indicates whether it describes the internal structure of the SuD or only its external, observable behaviour. In the former case, the use case is said to have a white box visibility, whereas in the latter case it is said to have a black box visibility. Needless to say, the latter is the usual visibility when using use cases as specifications of functional requirements, following Davis recommendations (Davis et al., 1993).

Most of the metrics proposed for use cases focus on use cases with the underlined characteristics in figure 1, i.e. use cases at system scope, at user goal level and with a black box visibility. Because of that, these will be considered as the default values of Cockburn's dimensions for the rest of this chapter.

2.3 Specification of use cases

Jacobson et al. (1992) made use cases very popular, but they did not provide much guidance on how to specify them. As a result, a plethora of use case templates, notations and writing guidelines have bloomed in the last years. Use cases, as a scenario–based technique, are fundamentally text–based. Others formats like UML activity and sequence diagrams or Petri nets can be used for specifying use cases but, as recognized by Cockburn (2001), Kulak and Guiney (2000) and other practitioners, stakeholders without a software engineering background usually understand written *stories* using the vocabulary of the problem domain better than any other software–oriented diagrammatic notation.

Assuming that use cases are basically text, there are still many possible ways of specifying them, from plain prose to structured English. Regardless of the writing style, one commonly agreed point is that any use case specification must describe a sequence of interactions between actors and the SuD, usually numbering the *steps* performed during the interactions in order to achieve some actor's goals.

A thorough discussion of all the proposed templates for use case specification is out of the scope of this chapter, but a summary of some of them is essential in order to understand some of the metrics proposals. Notice that, as actual requirements templates, use cases templates usually include *requirements attributes*[3] like a unique identifier, version, status, stakeholders, writers, dependencies on other requirements, associated non–functional requirements, etc. As commented in following sections, use–case metrics based on some of these attributes like the number of stakeholders with a stake on the use case, the number of dependencies of the use case or the number and type of associated non–functional requirements, should be taken into consideration when using use cases for effort estimation.

[3] The interested reader can see Davis (1993) or Sommerville and Sawyer (1997) for more details on requirements attributes and requirements management.

2.3.1 Cockburn's template

One of the first, most widely used templates for use case specification was initially proposed in (Cockburn, 1997) and later reviewed by its author in (2001). Its most relevant elements are the following:

- **Name, scope, level and visibility**: the name of the use case is the primary actor's goal in a short, active verb phrase. The primary actor is the actor requesting services from the system, usually triggering the use case. In Cockburn's template, the use case name is decorated with the corresponding scope, level and visibility icons (see figure 1).

- **Preconditions**: the preconditions of a use case are assertions about the state of the SuD and its environment — the *state of the world* in Cockburn's words — that will be checked before letting the use case start and that will not be checked again during the use case execution.

- **Minimal and success guarantees**: Cockburn considers two different groups of postconditions depending on whether the use case ends successfully or not, i.e. whether the primary actor's goal is achieved or abandoned. Minimal guarantees must hold regardless of the success or failure of primary actor's goal. Success guarantees must hold only when the use case concludes successfully.

- **Trigger:** a trigger is an event that fires the execution of a use case. Depending on the writing style, it can be considered as the first step of the use case or specified outside the main success scenario.

- **Main success scenario:** the main success scenario is a numbered sequence of steps performed during the execution of a use case that leads to a situation in which the primary actor's goal is achieved. Apart from including another use case, Cockburn considers three possible kinds of action to be performed in a step: an interaction between two actors (considering the SuD as a special kind of actor), a validation step, and a internal change of the SuD (even if the visibility of the use case is black box). The number of actions to be included in a single step depends on the writing style[4], although one or two actions are the usual number. Cockburn also recommends not using conditional steps in the main success scenario but considering them as extensions.

- **Extensions:** extensions are branches of the main success scenario depending on a particular condition — the extension condition — in a given step. Some of these branches can lead to success while other can lead to failure of the use case. Cockburn recommends using extensions for handling both situations, while other authors like Leite et al. (2000) or Durán et al. (2002) use conditional steps for successful, usual branches and exceptions for branches triggered by exceptional conditions usually leading to use case failure.

[4] See Cockburn (2001), pages 93–95, for details about including a reasonable set of actions in a single step.

- **Technology and data variations**: Cockburn considers different ways of performing a step as variations, like using different payment methods, different data during an identification of a user, etc. They are not considered to be alternative branches, i.e. they have neither condition nor steps.

2.3.2 RUP template

The *Rational Unified Process* (RUP) (Kruchten, 2000) is a software engineering methodology developed by the Rational company (now a company of IBM) after the *Unified Process* (UP) (Jacobson et al., 1999). One of its defined artefacts is the *RUP use case template*, including the following elements:

- **Name:** like in Cockburn's template, the name of the use case in the RUP template is a short description of primary actor's goal, although is augmented with a brief description in which a short summary of the use case is provided.

- **Pre– and postconditions:** in the RUP template, a precondition is defined as the state of the system that must be present before the use case starts. Postconditions are defined as a list of the possible states the system can be in after a use case has finished. Kruchten (2000) does not specify if postconditions must always hold or if they must only hold on successful ending of the use case, which seems to be the usual semantics. Notice that unlike in Cockburn's template, pre– and postconditions in the RUP template do not take the SuD environment into consideration.

- **Basic flow**: the basic flow is a numbered sequence of steps describing what actors do and what the system does in response. In the RUP template, conditional branches are allowed provided they are composed of only a few steps. In case of complex alternative branches, using an *alternative flow* is preferred. As in Cockburn's template, the action of a step can be a *inclusion* of another use case (or an *extension* if the step is conditional).

- **Alternative flows**: alternative flows describe alternative behavior usually due to exceptions that occur in specific steps in the main flow. When an alternative flow ends, the main flow is resumed unless otherwise stated. If needed, alternative flows can be divided into alternative subflows at arbitrary depth, although that use is discouraged.

- **Extension points**: in the UML 1.5 specification (OMG, 2003), an *extension point* is defined as a reference to one or a collection of locations in a use case where the use case may be *extended*. An *extend relationship* defines that a use case may be — i.e. depending on a *extension condition*— augmented with some additional behavior defined in another use case.

  From our point of view, one of the problems of this vague description of, a probably unnecessary, concept is that extensions points are not related to any step in neither the basic flow nor the alternative flow. In other words, they seem to be unattached labels for the starting points of alternative flows expressed as separate use cases.

2.3.3 Leite's template

Leite et al. (2000) proposes not only a scenario template but also a whole process for scenario construction. A important difference in this approach is that Leite's scenarios are tightly coupled with a *lexicon* containing concepts from the problem domain. In this way, Leite ensures that scenarios are written using the vocabulary of customers and users, thus enforcing their communicability. The most relevant elements of this template are the following:

- **Title and goal:** Leite's template includes both a name and a goal. Usually, the former is a short form of the latter.

- **Context and resources:** in Leite's template, preconditions are part of the context of the scenario, which describes a geographical location, a temporal location and preconditions. Leite also includes information about relevant resources, i.e. physical elements or information, that must be available during the scenario performance. Postconditions are not considered in this template.

- **Episodes**: the episodes of Leite's scenario template are basically the same as the numbered sequences steps of previously discussed templates. In this template, episodes can include other scenario or describe a simple interaction. Conditions can be included in the episode sequence, but they affect only one episode. For conditional branches composed of more than one episode, scenario inclusion must be used. Leite also considers *optional episodes*, i.e. steps that may or may not be performed depending on conditions that cannot be explicitly detailed. Notice that groups of *non–sequential* episodes can also be defined in a Leite's template, thus allowing a parallel or indistinct sequential order.

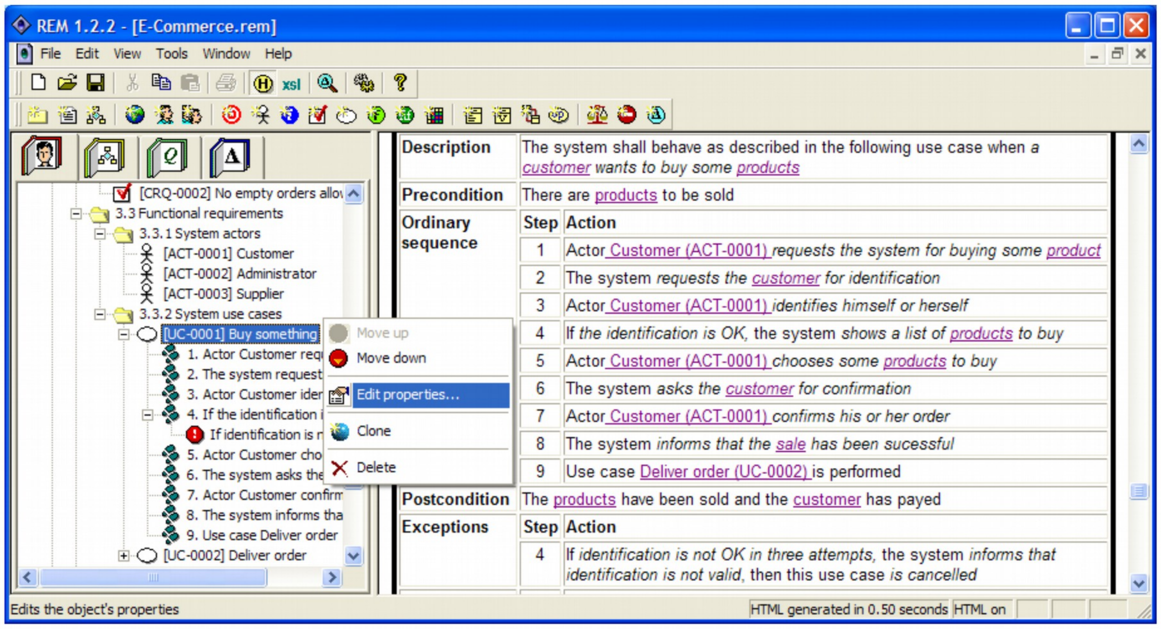
- **Exceptions**: this section of Leite et al.'s template contains the specifications of exceptional situations due to the lack or malfunction of some of the previously mentioned resources. The main differences with other templates are that exceptions are not associated to specific steps, and that only one action can be specified as the exception treatment, although that action can be a scenario inclusion.

2.3.4 Durán's template

Durán's template for use cases, formerly published in Durán et al. (1999), is one of the results of the PhD. thesis of one of the authors of this chapter (Durán, 2000). The most relevant characteristic of this template is that for a number of its elements some *linguistic patterns* are provided, thus easing use case writing[5]. Another interesting aspect of this template is that is fully supported by the free requirements management tool *REM* (Durán,

[5] A description of linguistic patterns is out of the scope of this chapter. The interested reader can see Durán et al. (1999) for details. A more extensive work on the use of linguistic patterns for use cases can be found in Ben Achour et al. (1999), one of the results of the CREWS project.

2003), as shown in figure 2. Its basic elements are the following:



[CRQ-0002] No empty orders allo
3.3 Functional requirements
3.3.1 System actors
[ACT-0001] Customer
[ACT-0002] Administrator
[ACT-0003] Supplier
3.3.2 System use cases
[UC-0001] Buy something
1. Actor Customer req
2. The system request
3. Actor Customer ider
4. If the identification i
If identification is n
5. Actor Customer cho
6. The system asks the
7. Actor Customer confirm
8. The system informs tha
9. Use case Deliver order
[UC-0002] Deliver order

Description	The system shall behave as described in the following use case when a *customer wants to buy some products*	
Precondition	There are products to be sold	
Ordinary sequence	**Step**	**Action**
	1	Actor Customer (ACT-0001) *requests the system for buying some product*
	2	The system *requests the customer for identification*
	3	Actor Customer (ACT-0001) *identifies himself or herself*
	4	*If the identification is OK, the system shows a list of products to buy*
	5	Actor Customer (ACT-0001) *chooses some products to buy*
	6	The system *asks the customer for confirmation*
	7	Actor Customer (ACT-0001) *confirms his or her order*
	8	The system *informs that the sale has been sucessful*
	9	Use case Deliver order (UC-0002) is performed
Postcondition	The products have been sold and the customer has payed	
Exceptions	**Step**	**Action**
	4	*If identification is not OK in three attempts, the system informs that identification is not valid, then this use case is cancelled*

Edits the object's properties
HTML generated in 0.50 seconds
HTML on

Figure 2. Support for use cases in REM

- **Name:** the name of the use case is, as in Cockburn's template, a short verb phrase stating the goal of the use case from the primary actor's point of view.

- **Description**: the description is based on a linguistic pattern including the triggering event of the use case. The simplified structure of this linguistic pattern is "*the system shall behave as described in the following use case when <triggering event>*". Notice that Durán considers the triggering event at the business level, whereas the first step of the *ordinary sequence* is usually a request of service from an actor to the system (see figure 3 for an example).

- **Preconditions**: preconditions include assertions that must be true in order to reach the goal of the use case. Like in Cockburn's and Leite's templates, preconditions are expressed not only on the system state but also on its environment.

- **Ordinary sequence**: the ordinary sequence describes the steps performed to achieve the use case goal when everything goes right, including single conditional steps like in Leite's template. The actions performed in any step can be actor actions (*actor–to–actor* or *actor–to–system* actions), system actions (usually only externally observable actions) or the performance of another use case (an UML *inclusion* or *extension*, depending on whether the step is conditional or not).

- **Postcondition:** postconditions include assertions about the system and its environment that must hold provided the use case ends successfully.

- **Exceptions:** exceptions have the same structure than in Leite's template plus additional information specifying if the use case is resumed or canceled after exception treatment.

UC–0015		**Register Book Loan**
Dependencies		• OBJ–0001 *To manage book loans (objective)* • OBJ–0005 *To know library users' preferences (objective)* • CRQ–0003 *Maximum number of simultaneous loans (business rule)* • CRQ–0014 *Return date for a loan (business rule)*
Description		The system shall behave as described in the following use case when *a library user requests a loan of one or more books*.
Precondition		*The library user has been identified by means of his or her identity card and has picked up the books to loan from the shelves*.
Ordinary sequence	**Step**	**Action**
	1	The librarian *requests the system for starting the book loan registering process*.
	2	The system *requests for the identification of the library user requesting a loan*.
	3	The librarian *provides identification data of the library user to the system*.
	4	The system *requests for the identification of the books to be loaned*.
	5	The librarian *provides identification data of the books to be loan to the system*.
	6	The system *displays the return date for each of the books to be loan and requests loan confirmation for each of them*.
	7	The librarian *tells the user library the return dates displayed by the system and ask him/her if he or she still wants to loan each book*.
	8	The library user *confirms the librarian which books he or she wants to loan after knowing return dates*.
	9	If *some of the confirmed books have an associated multimedia item*, then use case "*Add item multimedia to loan*" is performed.
	10	The librarian *re–confirms the book loans confirmed by the library user to the system*.
	11	The system *informs that the book loans have been successfully registered*.
Postcondition		*The library user can take the loaned books away and the system has registered the book loans*
Exceptions	**Step**	**Action**
	3	If *the library user has already reached the maximum number of simultaneous loans or has a penalty*, the system *informs of the situation, then this use case cancelled*.
Comments		*The maximum number of simultaneous book loans and the loan period depend on the library policy and can change in the future. See business rules CRQ–0003 y CRQ–0014*.

Figure 3. Use case example using a simplified version of Durán's template

2.4 Use case diagrams

Apart from their textual specification, use cases, their actors and their relationships can be depicted in the so–called *use case diagrams*. Use case diagrams were part of the initial proposal by Jacobson et al. (1992) and, with minor changes, they are also present in the current UML specification (OMG, 2003).

As commented by some authors like Cockburn (2001) or Kulak and Guiney (2000), use case diagrams must be understood only as a *table of contents* of use cases, not as an alternative of their textual specification. In use case diagrams, only the name of the use cases, the participating actors and some use case relationships are shown. The *essence* of use cases, i.e. their sequence of actor–system interactions, cannot be in anyway derived from use case diagrams. An example of a use case diagram can be seen in figure 4, where the system boundary is represented as a box containing some use cases. Actors are represented as stick men and their participations in use cases are depicted as association lines.

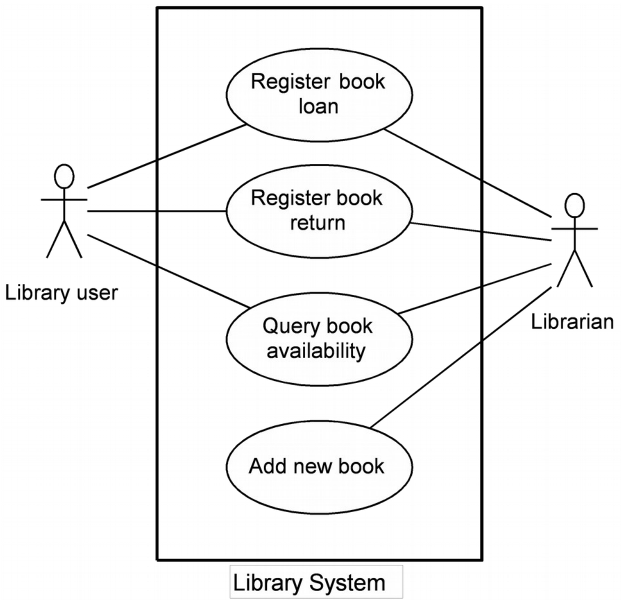


Figure 4. Use case diagram example

2.5 Use cases in the UML specification

As commented in the introduction of this chapter, the UML specification has caused confusion about some concepts related to use cases. Apart from focusing only in the diagrammatic notation, the introduction of three different kinds of *use case relationships*, i.e. *inclusion*, *extension* and *generalization* (see figure 5 for their graphical notation), has led many developers to build extremely complex use case models impossible to understand for their customers and users because of an excessive use these relationships.

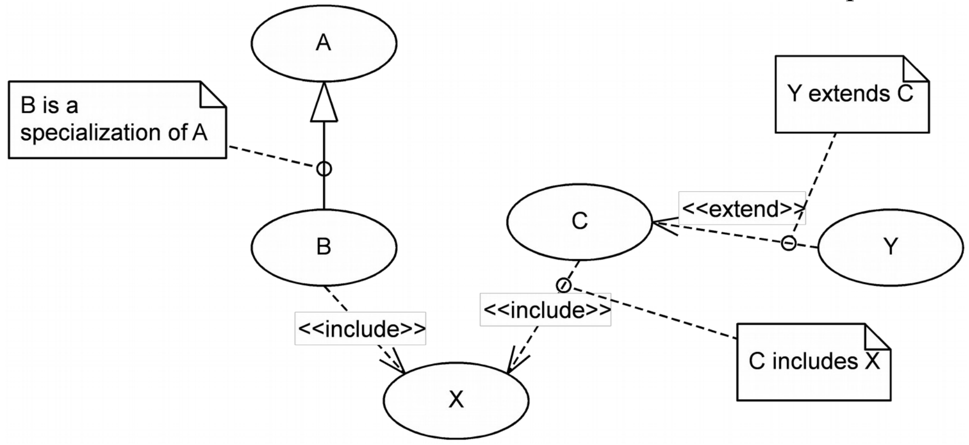


Figure 5. UML notation for use case relationships

We agree with Cockburn and other authors about the meaning of the so–called *include* and *extend* relationships. They must be considered as a means of avoiding redundancy in use cases specifications, but always taking into account communicability and understandability as the primary goals of use cases. In other words, a certain degree of redundancy is acceptable if it makes communication easier. Their counterparts in textual specification are those steps invoking or calling other use cases. Unconditional invocations are considered as *inclusions* and conditional invocations, i.e. conditional steps or exceptions, are considered as *extensions*.

Without a doubt, the most confusing use case relationship in UML is *use case generalization*. There are no clear semantics about the relationships of the sequence of steps of two use cases when one is a generalization of the other. In the UML specification, the semantics of use-case generalization are described in this way:

> *Generalization between use cases means that the child is a more specific form of the parent. The child inherits all features and associations of the parent, and may add new features and associations (page 2-132).*
> *[...]*
> *A generalization relationship between use cases implies that the child use case contains all the attributes, sequences of behaviour, and extension points defined in the parent use case, and participates in all relationships of the parent use case. The child use case may also define new behaviour sequences, as well as add additional behaviour into and specialize existing behaviour of the inherited ones. One use case may have several parent use cases and one use case may be a parent to several other use cases (page 2-138).*

As the reader can see in the previous definition of use-case generalization, its semantics are extremely ambiguous. On one hand, *child use cases* must contain the whole sequence of the *parent use case*. On the other hand, *child use cases* may define *new sequences, add new behaviour and specialize exisiting behaviour.* It is not clear at all how a child use case can redefine the sequence of steps of its parent use case — not to mention if the child use case has more than one parent!

Ambler (2001) interprets use case generalization in a different way:

> *Inheritance between use cases should be applied whenever a single condition would result in the definition of several alternate courses.*
> *[...]*
> *The inheriting use case is much simpler than the use case from which it inherits. It should have a name, description, and identifier, and it should also indicate from which use case it inherits in the "Inherits From" section. This includes any section that is replaced, particularly the pre-conditions and post-conditions as well as any courses of action. If something is not replaced, then leave that section blank, assuming it is inherited from the parent use case (you might want to put text, such as "see parent use case," in the section).*

Ambler's definition of use-case generalization is more concrete than the official one. It seems that use-case generalization should be use when a single condition is responsible of several alternate courses. In other words, if you find several — more than three, perhaps? — non-contiguous, conditional steps, all of them sharing the same condition, you should consider extracting all affected steps and create a child use case.

An obvious alternative to Ambler's proposal is using conditional steps — action steps or inclusion steps, i.e. extensions — instead of introducing a new kind of use-case relationship.

Cokburn (2001) recognises the problems of use-case generalization and proposes a completely different use:

> *In general, the problem with the generalizes relation is that the professional community has not yet reached an understanding of what it means to subtype and specialize behaviour, that is, what properties and options are implied. Since use cases are descriptions of behaviour, there can be no standard understanding of what it means to specialize them.*
>
> *If you use the generalizes relation, my suggestion is to make the generalized use case empty [...]. Then the specializing use case will supply all the behaviour [...].*

What Cockburn does not say is what is the reason for keeping the empty generalized use case, which seems to be useless.

As the reader can see, there is no consensus about what use-case generalization means. This situation is reflected in the fact that, as far as we know, no use-case metrics proposal takes use-case generalization into account.

2.6 Conclusions on the concept of use case

After analysing the concept of use case, their roles in software development, and some templates for their specification[6], we have reached the following conclusions:

- Although other uses are possible, use cases are mainly used as textual specifications of functional requirements. Hence, they can be considered as partial specifications of the system to be built for estimation purposes.

- Use–case templates usually include the use–case goal, preconditions, trigger, main successful scenario, postconditions, successful alternative branches and failure alternative branches. Some of them distinguish between usual and abnormal, i.e. exceptional, conditions of alternative branches. Some of them include information about the environment whereas other focus only on the software system.

- The usual number of actions specified in each step is usually one or two. Step actions are usually of one of three possible classes: actor action, system action and use case inclusion/extension.

[6] The interested reader can see other use case templates proposals (Schneider and Winters,1998), (Kulak and Guiney, 2000), and (Coleman, 1998).

- Use–case relationships must be used carefully, keeping use case specifications clear and easy to understand. Inclusion and extension should be used only as means of avoiding redundancy. Generalization should be used only when non ambiguous semantics were commonly agreed.

Taking these considerations into account, the different proposals for use–case metrics are analysed in the following sections.

3 Metrics for project estimation

As Smith (1999) commented, "*Intuitively, it seems as though it should be possible to form estimates of size and effort that development will require based on characteristics of the use case model. After all, the use case model captures the functional requirements…*"

In this section the main proposals to estimate size, effort and complexity of the system based on use cases will be presented.

3.1 Size and complexity estimation

3.1.1 Fectcke's et al. proposal

The *function points* (Albrecht, 1979) is one of the method that allows measuring the functional size of software systems. One of the main reasons for its use is because *function points* measures the functionality of software from the user viewpoint independently of technology used for implementation. With OO methods implantation it is advisable to adapt the *functions points* for the OO conceptual models.

In order to allow this goal, in Fetcke et al. (1997) a method to calculate *function points* based on conceptual models is shown. These conceptual models are use cases model, domain model and analysis objects model.

In order to calculate *function points*, previously it was necessary to know *unadjusted function points* that are calculated by adding the number of internal files and external files of the application, together with the inputs, outputs and inquiries from and to the user.

In particular, for the use cases model, Feckte et al. (1997) assume that some use cases will be mapped to direct interaction user-system and others will not because of the different possible detail levels of the use cases.

In order to correctly select use cases, the method proposed by these authors explains that first we have to apply the mapping actors rules that lie in choosing those actors (human or not) that are not part of the system under consideration, i.e. the system users and the others systems. Afterward, the use cases that are related with some of the selected actors will be chosen. Furthermore, we have to add the use cases that are related with a particular one by means of *extend* relationship.

Those use cases will be counted as systems interactions and will be added to the rest of elements selected in the analysis model or domain model in order to obtain *unadjusted function points*.

Fetcke et al. (1997) state how the method is applied to three development projects getting system size in *unadjusted functions points*. The main advantage is that the method is based on system requirements, collected at the very beginning of the system development process. The disadvantage is that it is only applicable when you use OOSE (Object Oriented Software Engineering) (Jacobson et al., 1992).

3.1.2 Marchesi et al.'s proposal

Also based on the use case model, metrics are defined to estimate system complexity. One of these proposal, explained in (Marchesi, 1998), which is based on possible mapping between use case and *function points* assures that the number of use cases (N_{CU}), the number of actors (N_a) and the number of *include* and *extend* relationships are good indicators of system complexity.

To estimate system complexity based on the metrics mencioned above Marchesi proposes the metric $UC4$ of which the equation is as following:

$$UC4 = K_1 UC1^2 + UC3 + K_2\left[smm(|C|) - smm(|E|)\right]$$

Where:

- The coefficients K_1 and K_2 are constants (less than one) and must be calculated empirically.
- *UC1* represents the number of use cases in the requirements specification. According to Henderson-Sellers et al. (2002), it is unknown why *UC1* appears squared. The author argues reasons of homogeneity because the other parts of the equation are proportional to $N_a \times N_{CU}$ and *UC1* is only N_{CU}.
- *UC3* is the total amount of communications among use cases and actors without redundancies introduced by *extend* and *include* relationships, taking into account the fact that the complexity of a use case increases more than linearly with of the number its communications.
- [*C*] is a matrix (with dimension $N_a \times N_{CU}$) and the element c_{ik} of [*C*] has value 1 if the actor i has a relationship with use case k , and value 0 otherwise.
- [*E*] is a matrix ($N_a \times N_{CU}$) representing the relationships between use cases after eliminating the redundancies due to *include* and *extend* relationships. Assuming a matrix [*M*], then $smm(|M|)$ is defined as the sum of all elements of matrix [*M*], taking into account that the difference $smm(|C|) - smm(|E|)$ is a measurement of the communications inherited by all use cases *extending* or *using* other use cases.

In our point of view, once the metric $UC4$ is defined it is advisable to try to empirically validate it in order to establish a correlation between this metric and system complexity, for example, doing a *correlational study* according to (Briand and Wüst, 2002). In the results of this study, it would be possible to establish the coefficients values (K_1 and K_2). On the other hand, it is difficult to fix the measure unit of $UC4$ and how the value bears upon project schedule.

3.1.3 Feldt's proposal

Feldt (2000) studies how the complexity and size of use cases influence system complexity and size.

In order to specify use cases, this author proposes to use UML sequence diagrams (OMG, 2003). For this reason, the metrics proposed in this section are suitable when it works with low level use cases. The only participants in the sequence diagrams used by Feldt are *actor* and *system*. They interchange messages as "calls to procedures" which can have parameters. Furthermore, repetition sequences and alternatives can appear as well as other elements usually seen in these types of diagrams (stimulus and interruptions).

In our opinion, it is preferable to specify functional requirements with some of the textual use case proposals (summarised in section 2.3. Specification of use cases) to make the communication between software developers and clients and users possible. Nevertheless, this proposal is interesting to see the aspects of use cases considered necessary to estimate size and system complexity.

In order to characterize the size and complexity of use cases, Feldt proposes the following metrics:

- Number of stimulus (external events that have influences in the use case performance).
- Number of alternative branches.
- Number of interruptions.
- Number of system responses (calls to procedures).
- Number of system actions (relationships with other use cases).
- Number of exceptions.
- Number of actors.

It is convenient to take into account that Feldt (2000) considers size and complexity as attributes that can be measured as a whole. Furthermore, it is interesting to study how he approximates the system complexity through the time invested in writing the sequence diagram.

3.2 Estimation of system effort

It is difficult to determine *a priori* the effort required to implement a use case. This is because use cases are used in different ways depending on engineers requirements or on the active rules in the organization. Consequently, use cases can have different abstraction levels.

In order to solve this problem there are some alternatives. Some of them suggest to classify use cases according to their detail level and then estimate the effort to implement them according to their assigned level. Other proposals count the number of analysis classes, which correspond to each use case, and based on this, they can estimate the effort of implementing it.

3.2.1 Karner's proposal

The proposal done by Karner (Rational Software) and collected in Schneider and Winters (1998) defines the concept *use case points*, analogue to the *function points* concept. *Use case points* is useful to estimate the effort (in man-hours) of development project.

The metric *use case points* (UCP) is defined as:

$$UCP = UUCP \bullet TCF \bullet EF$$

Where:

- $UUCP$ is the metric called *unadjusted use case points*. This metric is calculated as the weighted sum in number of actors and number of use cases in the requirements specification. Each actor and use case can have a different complexity. This provokes the weighing in the calculation of $UUCP$. The weights of the actors figure in table 1. The complexity of use cases depends on one of two factors: the number of steps, and the number of analysis classes corresponding to the use case. Tables 2 and 3 show the possible weights of the use case.

- TCF represents a technical complexity factor. This factor increases if there are complex non-functional requirements, for example, if the system is distributed, the code must be reusable or easy to change, etc. In table 4 you can see which factors have influence in TCF. Those factors have a weight between 0.5 and 2. According to the importance of this factor in the system, this weight will be multiplied by a number between 0 and 5 (0 means that the factor is not present in the system).

- EF represents the level of experience of the technical personnel that work in the project. The stability of the project also has influence on the value of EF.

Once the value of UCP is calculated, in order to estimate system effort Karner suggests applying the factor 20 man-hours per UCP. However, data obtained of its application in real projects advises that it is convenient to adjust this quantity.

One of these revisions is shown in Schneider and Winters (1998) which comments that it would be beneficial to adjust this quantity depending on the EF value. If the EF is highly affected by change in staff, this will provoke more effort in training team members or provoke the convenience of solving instability problems. In this case Schneider advises to use 28 man-hours per UCP.

In Banerjee (2001) it is also provided another possibility: to increase the number of man-hours to 36 per UCP. The reason for this approach is that negative numbers mean extra effort spent on training team members or problems due to instability. However, using this method of calculation means that even small adjustments of an environmental factor, for instance by half a point, can make a great difference in the estimate.

Furthermore the *use case points* method has been applied to different types of projects. Thus, in Arnold and Pedross (1998) how to use in large-scale software systems is explained. Experiences, based on empirical data of a productivity benchmark of 23 measured projects, have revealed the usefulness of the method in order to measure the size of a software system.

The method has also been applied in building Web Application Systems (Stoica, 2000) coming to the conclusion that this technique can be used by adding front-ends to the best existing cost models.

Furthermore, some CASE tools solve the calculation of UCP. One of them is Enterprise Architect (Systems, 2003) a modelling UML tool.

In our opinion, the most relevant thing of this approach is the consideration of the technical factors TCF and EF. More or less, this shows that it is insufficient in regards to the information collected in requirements specification to estimate effort because there are other factors whose influence is essential to take into account.

Actor Type	**Description**	**Factor**
Simple	Program interface	1
Average	Interactive, or protocol-driven interface	2
Complex	Graphical interface	3

Table 1. Actor weighting factors

Use case type	**Description**	**Factor**
Simple	3 or fewer transactions	5
Average	4 to 7 transactions	10
Complex	More than 7 transactions	15

Table 2. Transaction- based weights factors

Use case type	Description	Factor
Simple	Fewer than 5 analysis classes	5
Average	5 to 10 analysis classes	10
Complex	More than 10 analysis classes	15

Table 3.Analysis class-based weighting factors

Factor number	Factor description	Weight
T1	Distributed system	2
T2	Response or throughput perfomance objectives	1
T3	End-user efficiency(online)	1
T4	Complex internal processing	1
T5	Code must be reusable	1
T6	Easy to install	0.5
T7	Easy tu use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Includes special security features	1
T12	Provides direct access for third parties	1
T13	Special user training	Facilities are required

Table 4. Technical factors for system and weights

3.2.2 Smith's proposal

The approach described in this section presents a fundamental difference with the one described in the previous section; which is the detail level of the use cases object of the estimation. In the Karner's proposal, it is necessary that use cases have sufficient detail level, in order to know how many analysis classes correspond to each use case.

In brief, Smith (1999) defines four possible types of use cases according to detail level: Subsystem (L1), Group of subsystem (L2), System (L3) and System of subsystem (L4). A concrete use case can belong to a sole level, or to several of them in a concrete percentage.

The concept of subsystem coincides with the concept proposed by UML (OMG, 2003). The concept of subsystem group coincides with CSCI[7] (Computer Software Configuration Item) (DoD, 1993).

[7] A configuration item for computer software, where a *Configuration Item* is an aggregation of hardware or software that satisfies an end use function and is designed by the acquirer for separate configuration management.

The basic idea is that a use case of level L(i) needs less effort than a level L($i+1$) to be implemented in C++ language. This is because a use case of level L($i+1$) comprises several use cases of level L(i). On the other hand, the type of system also has influence in the effort of implementation of the use case. Thus, three types of systems are considered: simple business system, scientific system and complex command and control system.

Table 5 shows the level size in SLOCS (Source Lines of Code). In order to build table 5, the following sentences are assumed, whose justification are in Smith (1999):

- A subsystem (L1) implements 8 classes.
- To implement a class in C++ approximately 850 SLOCS are necessary.
- The level L($i+1$) is composed by 8 components of the level L(i).

Level	Size (SLOCS)
L1	7,000
L2	56,000
L3	448,000
L4	3,584,000

Table 5. Size of system (in SLOCS)

Once the level size is known it is necessary to take into account the following statements:

- to describe the functionality of 8 classes 300 scenarios are necessary.
- 10 use cases of level L1 can describe 300 scenarios.

Now, if models like COCOMO and SLIM are applied, the results obtained in table 6 show the effort in hours/use case (h/UC) necessary to implement a use case of level and type of a system in particular.

Level	Effort h/UC simple business system	Effort h/UC scientific system	Effort h/UC complex command and control system
L1	55 (range 40-75)	120 (range 90-160)	260 (range 190-350)
L2	820 (range 710-950)	1,700 (range 1,500-2,000)	3,300(range 2,900-3,900)
L3	12,000	21,000	38,000
L4	148,000	252,000	432,000

Table 6. Effort of use case depending on detail level and system type

In order to apply this estimation technique, we would have to take the set of use cases of the requirements specification and fit each of them in the suitable level L(i) or partially in several of them. Furthermore, it is convenient to take into account the number of pages that fill each use case. Subjectively, Smith exposed that a use case of a simple business system must occupy an average length of 5 pages, a use case of a scientific system, 9 pages, and a complex command or control system, 12 pages.

One example in Smith (1999) assumes a scientific system where the actual use cases count was 5, and one of them split at L4 and 4 at L3, further, the L4 use case is 12 pages and the L3 use cases average 10 pages, then the effort is: $1 \times 250 \times 12/9 + 4 \times 21000 \times 10/9 = \approx 2800$ staff months. The quotients $12/9$ and $10/9$ are used to account for the apparent complexity due to the 9 pages represented as the average length use case because the system is a scientific type.

3.2.3 Henderson-Sellers's proposal

Henderson-Sellers (2002) provides some metrics for size and complexity of use cases. According to the authors they could be useful for estimating external attributes (Fenton and Pfleeger, 1997), such as system effort and maintainability.
In order to measure the size of a use case, the following metrics are suggested:

- Number of atomic actions in the main flow.
- Number of atomic actions in each alternative flow.
- The longest path between the first atomic action of the use case to the final atomic action of the use case.
- Number of alternative flows (alternative flows are measured from the start of the use case to its termination).

On the other hand, the following *environment factors* contribute to the use cases complexity independently of size metrics shown above:

- Number of stakeholders.
- Number of actors.
- Total number of goals.

The author argues that these metrics measure complexity in the presence of two use case models with similar values in defined size metrics, but different values in environment metrics, probably the one with greater values requires more effort in doing any change. This is because there are more elements that must be reviewed to solve possible conflicts.

Other indirect metrics can be derived from the above metrics and include:

- Total number of atomic actions in the alternative flows.
- Total number of atomic actions in all flows.
- Number of atomic actions per actor.
- Number of atomic actions per goal.
- Number of goals per stakeholder.

After showing different proposed metrics to measure use cases size and complexity, most of the authors coincide that the main factor that has influence in the use cases complexity is the increase of the resources required to do a change in the use case. The greater the effort required to do a change the greater the complexity of the specification will be.

In these circumstances, also dependencies between requirements increase complexity. These dependencies appear in *traceability matrix* and in our point of view must be included as a factor to measure use cases complexity.

4 Metrics for Requirements Engineering

Leaving aside the project estimation and focussing on requirements engineering process, there are reasons to think that it would be beneficial to define metrics. Nevertheless, because the requirements engineering is a recent discipline, there are not too many proposals. The reasons mentioned are as follows:

- In Kamstiems and Rombach (1997) the importance of early detection of requirements problems is recognized to improve the quality in the software development process. This is because the cost of repairing defects increases as the project moved forward (Boehm, 1975).

- In order to increase the control and monitoring during the development of this task, it is necessary to know in detail the requirements engineering process. The fact of control in the process allows knowing early needs for change. This has advantages because one of the main problems in the development process is changing requirements and specifications, as TSG (1995) shows. These changes affect technology, schedule, budget and staff organization as commented in Costello and Liu (1995) quoting the paper of Glaseman and Davis (1980).

4.1. Quality in requirements specification

At the moment, there are not too many proposals, which specify how to predict quality requirements based on use cases. In spite of that, there are some proposals to evaluate quality of natural requirements. Generally, some of them can be applied to evaluate use cases quality. In this area, the following approaches exist:

- Manual verification of requirements: these approaches study aspects as stability, ambiguity or traceability of requirements. Some of them can be consulted in Davis et al. (1993), Costello and Liu (1995) or Hyatt and Rosemberg (1996).

- Automated verification of requirements: these proposals generally are based on NLP (Natural Language Processing). According to Fabbrini et al. (1998), the goal of NLP applied to requirements engineering is to know the vocabulary used, writing style, ambiguity (degree of syntactic and semantic uncertainty of the sentence), information conveyed by requirements, discovering underspecifications, missing information and unconnected statements.

One of these proposals applied to use cases was collected in Fantechi et al. (2002). The idea of this proposal is to automatically identify defects in requirements specification. In order to achieve this goal, a tool CASE automatically identifies words in the text of use cases that denote lack of expressiveness (due to ambiguity or incompressibility), lack of consistency or incompleteness.

On the other hand, some proposals to evaluate the design quality of use cases models have been done.For example, these metrics have been proposed:

- NumAss: The number of associations the use case participates in.
- ExtPts: The number of extension points of the use case.
- Including: The number of use cases which this one includes.
- Included: The number of use cases which include this one.
- Extended: The number of use cases which extend this one.
- Extending: The number of use cases which this one extends.

Some of these metrics were automated by the tool SDMetrics (SDMetrics, 2003).

4.1.1. Saeki's proposal

The modifiability is one of the desiderable properties of the requirements specification. IEEE (1993) defines a modifiable requirements specification as one whose structure and style is so that any change can be performed in an easy, complete and consistent way maintaining its structure and style.

In Saeki (2003) a set of metrics for use cases diagrams are defined. Based on these, the rate of modifiability can be calculated. The basic idea of the defined metrics is that if a use case needs a change, probably other use cases will also need a change: those that have a relationship with the originally changed use case. In short, *include* and *extend* relationships and the control[8] and data[9] dependency relationships are considered. The intuition suggests that, the more existing relationships in the model, the more difficult it will be to make any change.

Another factor that has influence in the modifiability of use cases is the type of use case. Simplifying the idea, if a use case has several goals (*types* to Saeki), it is more susceptible of changing than if it only has one goal.

In order to approximate the modifiability, the defined metrics are NOD (Number Of Dependencies) and $NUCT$ (Number of Use Case Types).

The next equation is the pattern that stands for $NO\_extends$, NO_uses, NO_CD and NO_DD metrics. These metrics express the modifiability index due to *extend*, *include*, control dependency and control data relationships, respectively.

[8] Control dependency expresses the order of execution of use cases.
[9] Data dependency expresses that one use case gives data to another.

$$NOD = \frac{AllDependencies - \#Dependency}{AllDependencies}$$

Where $\#S$ stands for the number of the elements of the set S, $UseCase$ stands for the set of all use cases in the diagram, $AllDependencies = (\#UseCase \times (\#UseCase - 1))/2$ is the set of all the possible dependencies that exist. In our opinion, term $AllDependencies$ is a teorethical term since semantically it would not make sense that all use cases are connected between them in the model.

On the other hand, the equation shown below expresses the modifiability index due to the fact that a use case covers more than one goal (*types*).

$$NUCT = \frac{1}{\underset{u \in UseCase}{AVE} \{\#\{ut \in UseCaseType \mid aggregates(u,ut)\}\}}$$

Where $u \in UseCase$ represents one of the use cases in the diagram, $aggregates(u,ut)$ means that a use case u has a type ut, $AVE_{p(x)}\{s(x)\}$ means the average value of a set of numbers $s(x)$ constructed from x such that $p(x)$. $NUCT$ is the reciprocal number of an average of attached use case *types* for each use case.

Based on these metrics the rate of modifiability ($MODIFIABILITY$) of the use cases diagram is computed as:

$$MODIFIABILITY = w_1 \times NO\_extends + w_2 \times NO\_uses + w_3 \times NO\_CD + w_4 \times NO\_DD + w_5 \times NUCT$$

where each w_i represents the weighting factor of the corresponding metric and the sum $w_1 + w_2 + w_3 + w_4 + w_5$ may be equal to 1 and $0 \leq w_i \leq 1 (i = 1,...,5)$. One possible solution is proposed by Saeki: $w_i = 0{,}2 \; \forall i$

This manner of defining the metrics is not obvious but it is justifiable in our point out view. The goal achieved by the author was to find an indicator rate ($0 \leq MODIFIABILITY \leq 1$) that would reveal the modifiability degree of a use cases model.

This proposal is interesting because of its capability to measure one of the desiderable properties in requirements specifications, the modifiability. The traces existing between use cases should also be included in the calculation of modifiability because the traced requirements can change with the original requirements. In general, the coupling between use cases is caused by *include* and *extend* relationships and by the use cases connected in a *traceability matrix* which should be considered in the modifiability calculation. The control and data dependencies reveal that use cases technique has been used in an inferior specification level, close to the sequence diagram.

4.1.2. Bernárdez and Durán's proposal

Bernárdez et al. (2004) have empirically revised a set of heuristics to identify use cases that potentially can have defects.

These heuristics, presented in Durán et al. (2002), are based on a set of use case metrics defined for the use cases model of REM (Durán, 2003). In this use cases model, a use case is seen as a sequence of steps that can be action-step, system-step or realize another use case as commented in section 2.3.4: Durán's template. An example of a use case of this model can be seen in figure 3.

The heuristics, based on the metrics shown in table 7, rely on a basic concept: there is a normal range of values for each metric $m \ \lfloor m_1, m_2 \rfloor$, out of which the probability of a use case c presenting defects ($P_{def} \lfloor c \rfloor$) increases.

$$m(c) \notin \lfloor m_1, m_2 \rfloor \Rightarrow P_{def} \lfloor c \rfloor >> 1 - P_{def} \lfloor c \rfloor$$

Metric	**Description**
NOS	Number of steps of the use case (NOS=NOAS+NOSS+NOUS)
NOAS	Number of actor action steps of the use case
NOSS	Number of system action steps of the use case
NOUS	Number of use case action steps of the use case (inclusion or extension)
NOCS	Number of conditional steps of the use case
NOE	Number of exceptions of the use case
NIE	Number of times the use case is included or extends other use cases
NOAS/NOS	Rate of actor action steps of the use case
NOSS/NOS	Rate of system action steps of the use case
NOUS/NOS	Rate of use case action steps of the use case
CC	Cyclomatic complexity of use case (NOCS+NOE+1)

Table 7. Use case metrics which Durán's heuristics are based

In table 8, we will apply the outlined metrics to the use case example shown in figure 3.

Metrics	**Value**	**Explanation**
NOS	11	There are 11 steps in use case "Ordinary sequence"
NOAS	6	There are 6 actor (*librarian* and *library user*) steps
NOSS	4	There are 4 system steps
NOUS	1	There is one *include* in step 9
NOCS	1	There is only a conditional step (step 9)
NOE	1	There is only an exception associated to step 3
NIE	0	This use case is not performed during another use case execution (The "Description" of the use case not include other use cases)
NOAS/NOS	0.54	6 divided by 11
NOSS/NOS	0.37	4 divided by 11
NOUS/NOS	0.09	1 divided by 11
CC	3	NOCS+NOE+1 (=1+1+1)

Table 8. Use case metric values for the use case of figure 3

The normal range of values (see table 9) was set using data from 414 non-verified (i.e. containing defects) use cases from students of Computer Science at the University of Seville.

Metrics	**Normal range**
NOS	[3,9]
NOAS/NOS	[30%,70%]
NOSS/NOS	[40%,80%]
NOUS/NOS	[0%,25%]
CC	[1,4]

Table 9. Use case metrics normal range

In order to consolidate the intuition of these heuristics, Bernárdez et al. (2004) have verified 8 requirements specifications from their students, containing 127 use cases. Some of the results achieved reveal that use cases outside of the normal range are fault—prone requirements. This fact is confirmed in figure 6, which shows the percentage of use cases that have defects in and out of the normal range.

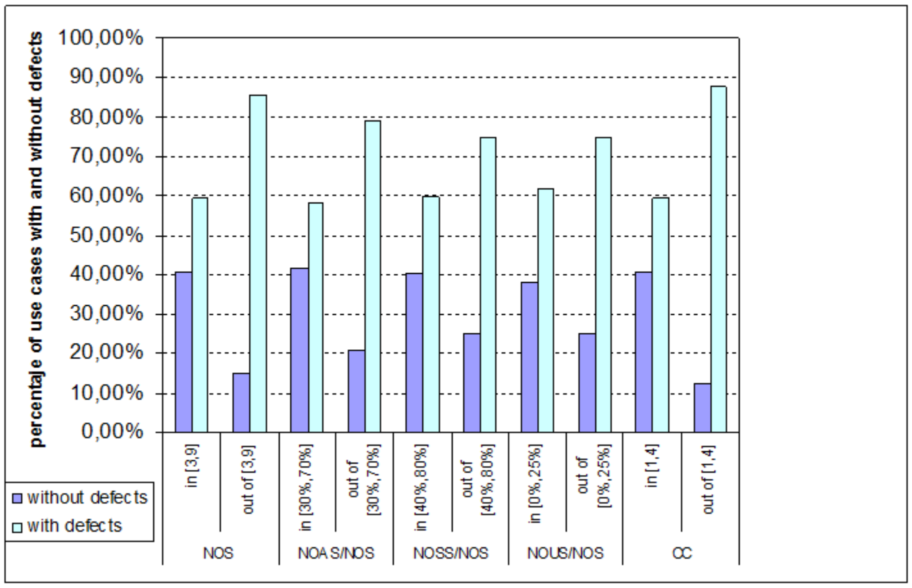


Figure 6. Empirical results on heuristics reviews

The main source of defects in use cases, which are revealed by the empirical data, are incompressibility, incompleteness (both defined by Davis et al.(1993) and the incorrect use of the use cases technique (according to Lilly (1999)). If data collected in figure 11 is confirmed by a controlled experiment, the most interesting result in the empirical study is that these metrics really allow the prediction of potential defects in requirements.

4.2 Progress of requirements engineering process

Some authors like (Costello and Liu, 1995) advise the calculation of some metrics as indicators of the requirements engineering process. As commented above, this would be beneficial because it would permit project managers and requirements engineers to monitor and better control the requirements engineering process.

4.2.1 Kim and Boldyreff's proposal

Kim and Boldyreff (2002) propose the following metrics:

- NAU (Number of Actors associated with a Use case): The goal of this metric is to measure importance of the requirement. If there are many actors interested in performing the use case, it must be very important to the system.

- NMU (Number of Messages associated with a Use case): This metric is calculated from a UML interaction diagram (OMG, 2003). NMU is useful to trace requirements to design elements.

- NSCU (Number of System Classes associated with a Use case): This metric measures the number of classes of which objects take part in a use case scenario. This metric goal is to know the impact of the change in a use case.

4.2.2 Alexander's proposal

Alexander (2001) proposes some metrics to learn the status and progress of requirements engineering and some metrics to reveal possible problems in requirements. Those metrics (that are shown below) can be calculated using the Scenario Plus Use Case Toolkit (Plus, 2003).

The first group of metrics points out status and progress of the requirements engineering process:

- The number of use cases.
- The number of actors.
- The number of alternative steps of the use case.
- The number of exceptions.
- The number of constraints.

The second group reveals problems in requirements:

- The number of use cases without exceptions.
- The number of use cases without steps.
- The number of use cases isolated.
- The number of relationships between use cases.

5. Conclusions

In this chapter, the basic main ideas to learn the use cases technique have been presented, also the main proposals to estimate software project attributes (in brief, size, complexity and effort) based on use case metrics has been collected. Furthermore, the main proposals to improve the requirements engineering process based on use cases metrics have been presented.

Tables 7 and 8 sumarise the most relevant proposals analysed in this chapter. The first column contains the author of the proposal. The second column contains the use cases metrics defined in the proposal The third column lists the external attributes to be estimated. The fourth column indicates whether the proposal provides a prediction equation or not. The fifth column indicates whether computation of the proposed metrics and estimated attributes is supported by any CASE tool or not.

Looking at table 7, we can see that the more estimated attribute is development effort. In order to estimate this attribute, authors usually measure attributes like size and complexity of use cases. Nevertheless, most of the proposals do not provide a prediction equation and they are not supported by CASE tools.

Author	**What is measured?**	**What is estimated?**	**Prediction equation?**	**CASE tool support?**
Marchesi (1998)	Number of use cases, number of actors, number of *include* and *extend* relationships	System complexity	No	No
Schneider et al. (1998)	*Use case points*	Development effort	Yes	Yes (Sparx System)
Smith (1999)	Number of use cases in each detail level, number of pages of the use case	Development effort	Yes	No
Feldt (2000)	Use cases complexity and size	System complexity and size	No	No
Software Solutions on Time (2001)	Number and type of use cases	Development time	Yes	Yes (Metric Data)
Henderson-Sellers et al. (2002)	Use cases complexity and size	Development effort or maintainability	No	No
In et al. (2003)	Number of actors and number of use cases	Development effort	Yes	Yes (OSMAT)

Table 7. Summary of use case metrics proposals for project management

Table 8 reveals that the relationships in use case models and use cases themselves can be used to predict attributes like modifiability and the existence of potential problems and defects in requirements. There is not any prediction equation, but there are some CASE tools available to calculate the proposed metrics.

At the moment, there are no proposals to deal with the theoretical validation of these metrics. Concerning empirical validation, there are no thorough studies that guarantee the causal relationship between the use cases metrics and the external attributes , such as development time, development effort, system complexity, etc., i.e. *internal validity* according to Wohlin et al. (2000). However some of the estimation methods explained in this chapter have been applied to real projects.

In order to empirically validate the metrics to estimate effort, it is suitable to perform experiments in a real environment because the experiment will cover several phases of the life cycle. But in real environments it is difficult to do controlled experiments and moreover too many resources are needed.

We have reached to the conclusions that there is a general intuition among several authors who think that some metrics regarding use cases are useful to project estimation or to improve the requirements engineering process, increasing thus quality in requirements specifications. On the other hand, the different metrics explained here are based on different use cases models and this makes the possibility of adapting the proposal to other situations difficult.

Furthermore, there are other proposals which have not been deeply investigated in this chapter, that define use cases metrics to estimate project cost or development time instead of effort. For example, the proposal presented in In et al. (2003), which presents a CASE tools called OSMAT (Ontology Software Metrics Analysis Tool). This tool is useful to estimate project cost based on UML models. In particular, the proposed metrics to use cases are: Number Of Actors (NOA) and Number of Use Cases (NOUC). On the other hand, there are some development CASE tools companies that propose techniques of project estimation based on use cases. For example, Software Solutions on Time (2001) estimates the time that will be invested in each use case along each phase of the development process.

Author	**What is measured?**	**What is estimated?**	**Prediction equation?**	**CASE tool support?**
Alexander (2001)	Number of use cases, number of actors, number of use cases without exceptions, etc.	Status of requirements and potential problems	No	Yes (DOORS)
Kim et al. (2002)	Number of actors, number of messages in the interaction diagram associated with the use case, number of system classes associated with the use case	Importance of the requirement, impact caused by change a requirement	No	No
Saeki (2003)	Number of relationships and dependencies between use cases	Modifiability	No	No
Bernárdez et al. (2004)	Number of steps of use case steps, rate of each type of step and ciclomatic complexity	Fault-proneness	No	Yes (REM)

Table 8. Summary of use case metrics proposals for improving the requirements engineering process

Acknowledgements

This research is part of the AgilWeb project (TIC2003-02737-C02-01) financed by "Subdirección General de Proyectos de Investigación, - Ministerio de Ciencia y Tecnología" (Spain), the TAMANSI project (PCB-02-001) financed by "Consejería de Ciencia y Tecnología de la Junta de Comunidades de Castilla-La Mancha" and the MUML project (011.100623) financed by the University of Castilla-La Mancha.

References

Albrecht, A. J. (1979). Measuring Application Development Productivity. *Proceedings of the IBM Application Development Symposiu*m. Monterey, CA, pp. 83–92.

Alexander, I. (2001). Visualising Requirements in UML. *Telelogic Newsbyte*. Available in http://easyweb.easynet.co.uk/~iany/consultancy/reqts_in_uml/reqts_in_uml.htm.

Ambler, S.W. (2001) *The Object Primer*. Cambridge University Press. 2nd edition.

Arnold, M., and Pedross, P. (1998). Software Size Measurement and Productivity Rating in Large–Scale Software Development Department. *Proceedings of the 1998 International Conference on Software Engineerin*g. Los Alamitos, CA, USA, pp. 490–493.

Barnejee, G. (2001). Use Case Points. *White paper*, Isavix.

Ben Achour, C., Rolland, C., Maiden, N. A. M., and Souveyet, C. (1999). Guiding use case authoring: Results of an empirical study. *Proceedings of the Fourth IEEE International Symposium on Requirements Engineering*. Limerick, pp. 36–43.

Bernárdez, B., Durán, A., and Genero, M. (2004). An empirical review of use cases metrics for requirements verification. *Proceedings of the SOFTWARE MEASUREMENT EUROPEAN FORUM (SMEF'04*). Rome. Accepted for publication.

Boehm, B. W. (1975). Some Experience with Automated Aids to the Design of Large–Scale Reliable Software. *IEEE Transactions on Software Engineerin*g,Vol. 1 No. 1, March, pp. 125–133.

Briand, L. C., and Wüst, J. (2002). Empirical Studies of Quality Models in Object–Oriented Systems. *Advances in Computers, Academics Press,* Vol. 59, pp. 97–166.

Cockburn, A. (1997). Structuring use cases with goals. *Journal of Object–Oriented Programming*, Sep-Oct 1997.

Cockburn, A. (2001). *Writing effective use cases*. Addison–Wesley.

Coleman, D. (1998). A use case template: Draft for discussion. Available in http://www.bredemeyer.com/pdf_files/use_case.pdf

Costello, R. J. and Liu, D. (1995). Metrics for Requirements Engineering. *Journal Systems Softwar*e, Vol. *29*, pp. 39–63.

Davis, A., Overmyer, S., Jordan, K., Caruso, J., Dandashi, F., A.Dinh, Kincaid, G., Ledeboer, G., Reynols, P., Sitaran, P., Ta, A., and Theofanos, M. (1993). Identifying and measuring quality in software requirements specification. *Proceedings of the 1st International Software Metrics Symposium*. Los Alamitos, California: IEEE Computer Society Press, pp. 164–175.

Davis, A. M. (1993). *Software requirements: Objects, functions and states*. Prentice–Hall. 2nd edition.

DoD (1993). *DoD-STD-2167, Defense System Software Developmen*t. Departament of Defense of the United States of America.

Durán, A. (2000). *A methodological framework for requirements engineering of information systems (in spanish)*. Doctoral dissertation, University of Seville.

Durán, A. (2003). REM web site. http://klendathu.lsi.us.es/REM.

Durán, A., Bernárdez, B., Ruiz, A., and Toro, M. (1999). A requirements elicitation approach based in templates and patterns. *Proceedings of the 2nd Workshop on Requirements (WER'99)*. Buenos Aires, pp. 17–29.

Durán, A., Ruiz-Cortés, A., Corchuelo, R., and Toro, M. (2002). Supporting requirements verification using XSLT. *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02)*. Essen, pp. 165–172.

Fabbrini, F., Fusani, M., Gervasi, V., Gnesi, S., and Ruggieri, S. (1998). Achieving Quality in Natural Language Requirements. *Proceedings of the 11th International Software Quality Wee*k. San Francisco.

Fantechi, A., Gnesi, S., Lami, G., and Macari, A. (2002). Application of Linguistic Techniques for Use CAse Analysis. *Proceedings of the IEEE Joint International Requirements Engineering Conference (RE'02*). Essen, Germany, pp. 157–164.

Feldt, P. (2000). Requirements metrics based on use cases. Master's thesis, Department of Communication Systems, Lund Institute of Technology, Lund University, Box 118, S-221 00 Lund, Sweden.

Fenton, N., and Pfleeger, S. (1997). *Software Metrics: A Rigorous and Practical Approac*h. PWS Publisher.

Fetcke, T. A., Abran, A., and Nguyen, T. (1997). Mapping the OO–Jacobson Approach into Function Point Analysis. *Proceedings of the 23th Technology of Object–Oriented Languages and Systems (TOOLS–23*). Santa Barbara, California, pp. 1–11.

Glaseman, S., and Davis, M. (1980). *Software Requirements for Embedded Computers: A Preliminary Repor*t. Document R-2567-AF. U. S. Air Force.

Henderson-Sellers, B., Zowghi, D., Klemola, T. and Parasuram, S. (2002). Sizing use cases: How to create a standard metrical approach. *Proceedings of the 8th Object–Oriented Information Systems 2002*. Montpellier, France. Springer–Verlag, pp. 409–421.

Hyatt, L., and Rosenberg, L. (1996). A Software Quality Model and Metrics for Identifiying Proyect Risk ans Assessing Software Quality. *Proceedings of the 8th Software Technology Conferenc*e. Available in http://satc.gsfc.nasa.gov/support/STC_APR96/quality/sct_qual.html.

IEEE (1993). *IEEE Recommended Practice for Software Requirements Specification*s (IEEE/ANSI Standard 830–1993). Institute of Electrical and Electronics Engineers.

In, P., Kim, S., and Barry, M. (2003). Uml–based object–oriented metrics for architecture complexity analysis. *Proceedings of Ground System Architectures Worksho*p. El Segundo, CA. Available in http://sunset.usc.edu/gsaw/gsaw2003/s8e/in.pdf.

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The unified software development process*. Addison–Wesley.

Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992). *Object–oriented software engineering: A use case driven approach*. Addison–Wesley.

Kamsties, E., and Rombach, H. D. (1997). A Framework for Evaluating System and Software Requirements Specification Approaches. *Proceedings of Requirements Targeting Software and Systems Engineering.* Bernried, Germany, pp. 203–222.

Kim, H., and Boldyreff, C. (2002). Developing Software Metrics Applicable to UML Models. *Proceedings of the 6 th International Workshop on Quantitative Approaches in Object–Oriented Software Engineerin*g. Málaga, Spain, pp. 67–76.

Kruchten, P. (2000). *The rational unified process: An introduction*. Addison–Wesley. 2nd edition.

Kulak, D., and Guiney, E. (2000). *Use cases: Requirements in context*. Addison–Wesley.

Leffingwell, D., and Widrig, D. (2000). *Managing software requirements: A unified approach*. Addison–Wesley.

Leite, J. C. S. P., Hadad, H., Doorn, J., and Kaplan, G. (2000). A scenario construction process. *Requirements Engineering Journal*, Vol. *5* No. 1, pp. 38-61.

Lilly, S. (1999). *Use Case–Based Requirements: Review Checklist* (Technical Report). SRA International, Inc.

Marchesi, M. (1998). OOA Metrics for the Unified Modeling Language. *Proceedings of the 2nd EUROMICRO Conference on Software Manteinance and Reengineerin*g, pp. 67–73.

Meyer, B. (1997). *Object–oriented software construction*. Prentice–Hall. 2nd edition.

OMG (2003). OMG Unified Modeling Language Specification, v1.5.

Regnell, B., Anderson, M., and Bergstrand, J. (1996). A hierarchical use case model with graphical representation. *Proceedings of the IEEE International Symposium and Workshop on Engineering of Computer–Based Systems*. Friedrichshafen, pp. 65–84.

Plus, S. (2003). Use Case Toolkit for DOORS. Available in http://www.scenarioplus.org.uk.

Saeki, M. (2003). Embedding Metrics into Information System Development Methods: An Application of Method Engineering Technique. *Lecture Notes in Computer Scienc*e, Vol. *268*1, pp. 374–389.

Schneider, G., and Winters, J. P. (1998). *Applying use cases: a practical guide*. Addison–Wesley.

SDMetrics (2003). SDMetrics: The Software Design Metrics tool for the UML. Available in http://www.sdmetrics.com/.

Smith, J. (1999). *The Estimation of Effort based on Use Case*s (Rational Software white paper). Rational Software. Available in http://www.rational.com/media/whitepapers/finalTP171.PDF.

Software Solutions on Time (2001). A fresh and innovative approach to systems development and software project management. Available in http://www.tassc-solutions.com/omx/pages/metric_data.htm#usecase-metrics.

Sommerville, I., and Sawyer, P. (1997). *Requirements engineering: A good practice guide*. Wiley.

Stoica, A. (2000). Aspect of Building Web Application Systems Using the MBASE Approach. *Proceedings of the 15th International Forum on SCM/Focused Worksho*p. USC–CSE.

Systems, S. (2003). Enterprise Architect: UML modeling and design tool. Available in http://www.sparxsystems.com.au.

TSG (1995). *The CHAOS Repor*t. The Standish Group. Available in http://www.standishgroup.com/chaos.html.

Weidenhaput, K., Pohl, K., Jarke, M., and Haumer, P. (1998). Scenarios in system development: Current practice. *IEEE Software*, Vol. *15* No. 2 , pp. 34–45.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2000). *Experimentation in Software Engineering: An Introductio*n. Kluwer Academic Publishers.