

# Adapting Use Case Model for COSMIC-FFP Based Measurement

Piotr Habela<sup>1</sup>, Edgar Głowacki<sup>1</sup>, Tomasz Serafiński<sup>2</sup>, Kazimierz Subieta<sup>1 2 3</sup>

<sup>1</sup> Polish-Japanese Institute of Information Technology, Warsaw, Poland  
{piotr.habela, edgar.glowacki}@pjwstk.edu.pl

<sup>2</sup> Łódź University of Technology,  
Łódź, Poland,  
tomaszek@kis.p.lodz.pl

<sup>3</sup> Institute of Computer Science, Polish Academy of Sciences;  
Polish-Japanese Institute of Information Technology, Warsaw, Poland  
subieta@ipipan.waw.pl

**Abstract.** The COSMIC-FFP estimation is defined using abstract, domain-independent terms that need to be mapped onto the notions of the utilized software development methodology. On the other hand, for productivity reasons it is important to avoid model reconstruction dedicated for just measuring purposes. Use Case model seems to be an optimum candidate for serving both general purpose requirement management and measurement input model needs. The measurement preparation influences to some extent the perspective of the use case model, but can also enforce its quality. In this paper we describe our experiences in adjusting both use case model style and detailed measurement principles to achieve synergy between requirement specification and size estimation, and to keep the counting intuitive and adequate. We focus on resolving the functional redundancy issue by appropriate Use Case organization. In addition, the suggested style of use case specification is intended to make allow counting the functional size units in a straightforward way for particular scenario step specifications. Some remarks on counting use case variants and different actor sets are also provided.

## 1 Introduction

Functional size measurement can be perceived as the most promising method of software development effort estimation. The efficiency of such approach comes from the following features potentially available with them:

- Applicable early in the software lifecycle: software requirements specification may provide the necessary input data.
- Independent of software development technology (in contrast to e.g. lines of code-based methods).
- Based on objective criteria. Appropriate precision of functional elements counting rules allows to expect measure repetitiveness.

- Universality. The properties being the subject of counting are generic enough to cover very different software domains (e.g. not only information systems – the original area of functional measurement application).
- Neutral concerning software development methods.

To realize the abovementioned features, especially the last one, the function point analysis methods usually introduce their specific functionality modeling notions. This may lead to the need of mapping the development method-specific model of system's functionality onto the one prescribed by the measurement method. We argue however, that this would be very problematic for overall software development productivity.

It would be very desirable so that we could apply a measure directly to general-purpose requirement documentation. Since the genericity of measurement rules is valuable, it would be necessary to provide a precise interpretation of particular common requirement model forms to perform counting directly on them. However, such an approach is not feasible without assuring the presence of necessary features of requirements models used. In fact, their adjustment to the measurement method needs may constitute the main effort of introducing functional software measurement. Fortunately, as have observed the adjustments dictated by measurement method contribute to overall requirements quality.

Apart from the measurement productivity problem, there are a number of issues inherent to functional size methods. They include, among others, dealing with non-functional requirements, internal processing complexity and functional model redundancy or underestimation. Their satisfactory solution depends on both the suitability of the measurement rules of a given method as well as on the quality of requirements modeling.

In this paper we describe our observations on initial experiences in adopting COSMIC-FFP method to measure software documented using use case model. As a requirements model we chose to adapt a use case approach that we perceived to be the most promising and universal considering the various kinds of software do be modeled. The paper is organized as follows. In section 2 we summarize the functional methods' development and describe our motivation behind choosing COSMIC-FFP. Section 3 provides an overview of the use case approach to requirement specification, its standardization scope, flavors and applications. In section 4 we describe the synergic potential between those two methods and present our approach to adopting them. Section 5 concludes.

## **2 Functional Methods: Needs, Issues and Development LNCS**

Functional measurement methods offer a big advantage due to the level of abstraction they use. However, as already mentioned, this approach is also prone to several issues that need to be solved to achieve measurement objectiveness and adequacy. Some of

them strongly depend on the modeling and measuring principles applied. The evolution of functional point analysis methods shows interesting trends in this extent.

## 2.1 Common issues

Looking for unambiguous determinants of functional size, measurement methods suggest counting the data flows towards and from external users, as well as persistent data-related manipulation. Abstracting from particular methods, this approach raises the following problems method rules should be specific about:

- **Granularity of data.** In order to count a proper number of data flows, granularity of data units needs to be precisely defined.
- **Data flow uniqueness criteria.** If functionality contains a number of similar data flows, it should be possible to determine which of them should be treated as unique and in what case they should be identified as the same functionality and thus counted once.
- **Functional units' uniqueness criteria.** Apart from proper identification and counting of data flows within a given functional unit, it is necessary to verify if model introduces no redundancy on a inter-unit basis.

Popular function point analysis methods – such as IFPUG-FPA [Lo04, I3-03] or Mk. II FPA [Mk98, I4-02] – attempt to address those issues to remove potential ambiguities. The task is easier when only a particular domain of software systems is considered. This is the case for those methods, as they are primarily oriented towards information systems area. Note that although user interface design belongs rather to the later phases of software development, the measurement guidelines referring to its constructs as found in Mk. II FPA are useful to properly count the functionality. Concerning the possibility of redundancy, the counting principles tend to extract the “common denominators”, thus promoting the software reuse.

Other key issues are non-functional requirements and other environmental factors that could increase the development effort. Taking into account that the effort estimation remains the main purpose of functional measurement, considering those factors is fully understandable. Typically measurement methods take the following approach:

1. Counting purely functional size.
2. Assessment of method-specified factors – a grade from a prescribed range is assigned to each of them.
3. Multiplying the functional size by the grades assigned using method-prescribed weights.

However, the approach is problematic for at least two main reasons:

- In comparison to functional size counting, it is not possible to provide equally precise criteria for weighting the non-functional factors.
- Proper weight (that is, impact) of particular factors may be technology-dependent and may change over time. Thus one of the main benefits of functional measurement – technology independence – is undermined.

Those problems are reflected in the evolution of the approach to the issue of non-functional factors represented by the most popular functional size methods. IFPUG-FPA assumes a regular usage of technical complexity factors to be applied to the counted purely functional size. Newer Mk. II FPA method includes a different set of non-functional factors, however does not recommend using them (the usage of unadjusted size is preferred). Finally, the newest COSMIC-FFP method completely gives up the non-functional aspects, focusing on formulation of precise rules for objective, purely functional sizing.

The last of main shortcomings of functional size-based effort estimation is the algorithmic complexity of internal processing. Namely, the functionality similar in the sense of external data flows may differ significantly in terms of internal processing and thus development complexity and effort. The assumption that the processing complexity is proportional to external data manipulations is acceptable only among the similar kinds of software. Thus, to make the measurements universally comparable, some additional means of representing the algorithmic complexity need to be employed. However, assignment of additional values (multiplicative or – more suitable – additive to the functional size counted) raises the same questions as above discussed, concerning the objectiveness of such assessment.

## 2.2 Use Case Points

In contrast to other measurement methods, Use Case Points [Ka93] directly refer to the notions we assume to use for requirements modeling (that is, use case model and class model). This makes the method potentially the most straightforward to apply in our case. However, as will be shown, some drawbacks may suggest considering an adaptation of another functional measurement method.

The method assumes classification of use cases into three groups, based on their roughly determined size or complexity. The criteria are either the number of interaction steps in use case scenario or the number of (domain model) classes involved in its processing. Based on that classification, each use case is assigned a number of 5, 10 or 15 Use Case Points. Interestingly, the method suggest the use cases connected through «uses» or «extends»<sup>1</sup> (that is, not connected directly with actor) should not be counted [AD01]. Also actor count contribute to the functional size, though their impact is smaller. Actors are assigned 1, 2 or 3 points, depending they access system through local API (1), through textual interface or network (2), or through a graphical user interface (3). Note that in effect, already at this stage a rather non-functional aspect (actor's complexity) was introduced.

Use-Case Points method puts also much attention to the non-functional factors. It adopts (with minor changes) the set of technical complexity factors (TCF) of the IFPUG-FPA method. In addition it also introduces a set of so-called environmental

---

<sup>1</sup> Newer versions of UML offer a modified organization of relationships among use cases. The uses and extends relationships are replaced with include, extend and generalization.

factors (EF), potentially influencing a given organization's productivity. Note that the TCFs may be considered as related to the developed software itself (i.e. influencing its "market price"). The EFs in turn, are rather internal to the software developing organization. This remark may be important for a subtle distinction between assessing software price and effort or productivity estimation. The factors, using method-prescribed weights are applied to the counted number of use case points, transforming the so-called Unadjusted Use Case Points into the result expressed in Use Case Points.

In our case there could be two benefits of applying Use Case Points approach. Firstly, it is directly applicable to the assumed form of requirements document. Secondly, the way functional size is counted does not enforce full refinement of use case scenarios (it is only necessary to provide enough detail to classify each use case as simple, medium or complex).

However, we have identified three issues that speak against the application of Use Case Points in the target environment:

- **Lack of the official standard status.** The standardization of a measurement method e.g. as an ISO specification (like in the case of IFPUG-FPA [I3-03], Mk. II FPA [I4-02] and COSMIC-FFP [I2-03]), provides it with an authority, which is desirable e.g. if the measure is to provide criteria for contract statements.
- **Potentially inadequate technical complexity factors.** Those factors were identified for the oldest of main functional measurement methods. Their influence today may differ significantly and further evolve. The same concerns actor complexity (e.g. taking into account the progress in the area of graphical user interface development tools or productivity of Web user interfaces).
- **Use case style impact on measured size.** There is a potential threat of under- or overestimation, due to appropriately locating a significant part of functionality in extending and included use cases, or because of not factoring out a common functionality from several use cases.

### 2.3 COSMIC-FFP

COSMIC-FFP (The Common Software Measurement International Consortium – Full Function Points) [AD03, I2-03] is the most recent of mainstream methods. It is based on experiences of earlier common methods, especially Mk. II and FFP. It assumes a broader area of applications, attempting to cover with adequate rules both real-time and information systems domains. That assumption results in a rather generic terminology used. Thus, it is necessary to map the method rules to concrete notions used in the domain of applications (in our case – information systems).

The counting principles are rather simple. The method specifies data unit called data group. In the area of information systems this is equivalent to normalized entity. Alternatively, a finer data granularity may be used: in that case the intuitive notion of data attribute is suggested. The functionality is modeled in the form of functional

processes which in turn are divided into sub-processes. The latter are divided into data movement and data manipulation. The main subject of counting are data movement sub-processes, classified into four kinds:

- **Entry** – a movement of data from a user into the functional process that needs it.
- **Exit** – a movement of data from a functional process to the user that needs it<sup>2</sup>.
- **Read** – a movement of data from persistent storage (internal to the modeled system unit) to the functional process that requires it.
- **Write** – a movement of data from a functional process to persistent storage (internal to the modeled system unit).

The above mentioned sub-processes are counted accordingly to the number of data groups being moved within a given functional process (each unique data group movement is counted as 1 Cfsu – COSMIC functional size unit). For algorithmically complex functionality, the method allows for additional counting for data manipulation sub-processes. However, no precise instructions are given how to objectively elicit the size of the manipulation sub-processes so that it does not upset the whole estimation.

An important feature of COSMIC-FFP is the explicit support for partitioning and heterogeneity of modeled systems. The method introduces the notion of layer to handle different levels of abstraction within a model (e.g. application functionality vs. database repository development). What is even more important, the method also explicitly deals with partitioning of functionality at the same level of abstraction. It addresses the complexity of multi-tier architecture by representing the communication of each distinguished component with its environment separately.

The method's scope of interest includes functional requirements only. Thus the method can be classified as purely functional. It was developed to be compliant with the respective ISO standard [I1-98].

As it can be seen, the method principles are quite simple and intuitive, however, they assume a detailed functionality specification. To deal with early functional size estimation, it suggests to count the functional processes identified, classify them into few categories according to their complexity and to assign them experimentally determined average functional size. This variant resembles the Use Case Points approach, except the mean values assigned to functional processes are not prescribed.

### 3 Use Cases

Use case modeling gained a huge popularity within the software industry as the software functional requirements, as well as business modeling mean. The fact that the

---

<sup>2</sup> The definition of user is rather broad and includes any external actors directly communicated with the modeled system.

approach does not enforce particular development methodology made it easier to adopt. There are several significant advantages over traditional, narrative requirements specifications:

- **Intuitiveness** – the model shows software user’s point of view.
- **Testability** – in contrast to narrative requirements specification, the functionality is described in a uniform way, from which the test scenarios can be easily derived.
- **Goal-orientedness** – interactions of each use case realize some actor’s goal. This makes it easier to identify stakeholders, verify requirement completeness with them and to assign priorities.
- **Reuse mechanisms** – the notions of use case model encourage identification and extraction of multiple use functionality elements.

The versatility of the notion resulted in the development of various flavors of use case specifications. This situation was reflected in the fact that e.g. the UML standard leaves many aspects of the use case model open. Just the basic visual notations and the definitions of main terms are the subject of the standard’s specification.

The main criterion of distinction is the purpose of a use case model. This affects modeled system’s boundaries, specification’s style and level of detail. The model may be oriented towards business logic representation or towards strict software functionality specification. Also within the latter area itself, the model may be more abstract or more design oriented (appropriately essential or system use cases, using the terminology from [AM04]).

The common association for the use case model is the diagram. However, in fact any of the mentioned applications requires structured specifications of each use case as the core of the model. The style of those specifications is a subject of large variability. The goal-oriented approach described in [Co00] suggests a popular and useful form of those specifications. Main course of interaction is provided as numbered list of steps, each of them qualified by its performer. An intuitive and compact specification of alternative flows is realized through separate numbered lists referencing the main scenario steps. This style of definitions encourages encapsulating simple variants within a single use case definition and thus prevents to large extent the abuse of the «extend» relationship.

Relationships between use cases are often considered a secondary feature, not recommended for inexperienced use case modelers due to the risk of obscuring the definition. However, due to their important functionality reuse potential, the use of those relationships needs to be carefully considered from the functional measurement point of view. It is worth mentioning, that originally the «uses» and «extends» relationships were variants of OO inheritance relationship among use cases. However, they were often used in another way. The former relationship was applied to include a reusable interaction fragments (in a procedure-call style). The latter was often applied to define more complex exceptional flows of events. Therefore, the later versions of UML [OM03] reorganized the model, introducing «include», «extend» and generalization relationships. This legitimizes the mentioned use of «include» relationship. However,

concerning the «extend» relationship some authors [AM00] argue that it should be used only for adding the steps into the existing base scenario – not for adjusting it.

## **4 Applying COSMIC to Use Cases**

In this section we present our observations and choices concerning the integration of the approaches described above. We start from formulating our needs concerning the measurement method, and then adjust the use case specifications to serve it. Finally, we tune the counting principles to be intuitively applied to such requirements model.

### **4.1 Choosing Optimum COSMIC Perspective**

Although the area of interest of our project was limited to the information systems area, we need to deal with very different kinds of software in terms of its purpose and architecture. Firstly, significant differences of complexity were observed depending on the required system's architecture. The COSMIC tier notion helps to overcome this issue. Modeling functionality that spans through more than one tier, results in specifying tiers' boundaries and counting additional entry and exit flows when data needs to cross them in a given functional process. It is just necessary to provide precise criteria on when separating a tier is allowed and when it would be an unjustified raise of measured size. The most general assumption is to partition the analyzed system according to different implementation technologies of particular components and to development organization (e.g. different teams/companies).

The second issue concerned various complexity of data groups (i.e. entities) processed in the analyzed systems. This led us to the decision to base counting on data attributes rather than data group units. The potential problems are: (1) bigger precision of requirement specification needed and (2) lack of straightforward size comparability with measurements based on the standard Cfsu. Our decision was dictated mainly by the specificity of the environment we developed our approach for. The main aim of our measurement approach was to address the requirements of a department responsible for development of software which in most cases integrate various solutions supporting essential business activities of a big corporation. The characteristic of such environment is the fact that attributes composing a given entity are often distributed amongst many software systems. In addition, for the future users of the planned solution it is easier to point out the attributes than name data groups such attributes are part of. Therefore eliciting entities would entail an extra effort without any justification for the inception stage whose main aim is to give the answer whether to continue or cease further development. In this particular environment the business users of the software solutions are able to formulate their expectations quite precisely, and so enumerate the attributes required to satisfy their needs. Taking into account the above circumstances we came to conclusion that founding the measurement on data groups would result in constructing a data model which unnecessarily will be reflected in the future solution. In addition the elicited model would have bigger

granularity than the one based on attribute flows which in turn would produce bigger estimation error – if the estimation model is not transferred into the implementation model. Grouping attributes into entities would lead to unnecessary setbacks in communication with the future users. Specifying the requirements in terms of attributes rather than data groups motivates the users to consider their real needs more deeply, which speeds up the requirements capture. Another reason we decided to found the measurement on attributes is the fact we are just beginning to gather the statistical data required for translation from the functional size into the assessment labour intensity. More exact measurement data will simply give us more information about the inherent dependencies between development effort and the functional size.

## 4.2 Required Features of Use Case Model

To provide the data movement information needed for measurement, in use case model we need to take a rather system- or design-oriented perspective. That is, technical realization of functionality needs to be clearly specified, perhaps at the cost of loosing some of model flexibility and robustness. Note however, we do not need to delve into e.g. UI design or programmer's interface signatures details. We need to identify all the data that is logically needed to perform a given functionality, but being aware of architectural constraints and external actor requirements.

In order not to loose a general picture, we keep in our use case template some business oriented specification elements. They include use case goal formulation, references to business rules catalog, preconditions and post-conditions. They allow e.g. to verify the completeness of modeled functionality from the business logic point of view. For measurement, the central parts of use case specification are main scenario and alternative scenarios. We assume keeping them detailed enough, to easily assign the number of functional size units to particular scenario step. Thus, a step must specify its performer and a set of data attributes being moved. The scenario can not be limited to external communications, since also persistent data reads and writes performed locally must be explicitly identified. If some step may be performed by different actors, we need to be aware of communication channels they use. This is because the respective data flows should be counted separately only when the communication method differs technically.

The key problem for measurement objectiveness is assuring true uniqueness of functional processes. Since we attempt to make the functional counting straightforward<sup>3</sup>, the main responsibility lays on the model organization side. While in typical, general-purpose use case modeling the use of relationships among use cases is optional, here, special emphasis for avoiding redundancies is necessary. Thus, iterative approach to use case modeling should be applied, to identify the repeating elements (whenever it is really feasible to implement them identically) and factor them out using the «include», «extend» or generalization relationship.

---

<sup>3</sup> That is, we assume that counting should be possible within particular use cases independently.

### 4.3 Interpreting COSMIC Rules on Use Cases

The way use case model is prepared and used for measurement needs to respect the standardized rules of the COSMIC method. Thus, we assume that a use case consists of one or more functional processes as defined by the method. However, some variation is introduced by the use of relationships among use cases, as it usually results in a number of use cases that are not self-dependent when considered separately from cases they are attached to. An intuitive rule of the method is that a functional process must at least consist of two data movements to provide functionality to its user. Namely, some triggering input (entry) and at least output (exit) or registration (write) of information should be present. This needs to be revised for any kind of abstract use cases (e.g. responsible for only a fragment of user goal). Therefore, the abovementioned rule should be applied to verify complete use case instances (joining the functionality of related use cases) rather than to separate use cases. The only place where we diverted from the cosmic rules is the treatment of triggering event. Since there is a difference of complexity between a flow that just triggers a functionality and an initial flow that provides some input data attribute, we assume counting it only in the latter case. Thus, when counting we may depend on just the steps of use case scenario.

To keep the method conceptually clean, we avoid any extensions for handling non-functional factors. However, as their impact on development effort is in many cases unquestionable, we suggest analyzing those requirements using aspect-oriented approach [Ki97]. This is where the COSMIC-FFP layer notion becomes especially useful. Namely, some important non-functional requirements can be refined as a functional, crosscutting concern, located on a lower level of abstraction. If this is the case, such aspect may be uniformly modeled using use cases and its data exchange with the main functionality should be counted according to the COSMIC-FFP principles.

## 5 Conclusions

In this paper we have provided our motivation behind adopting a COSMIC-FFP based purely functional measurements. The approach is to be used for management purposes in an organization developing various kinds of software in the information systems area. When analyzing input data needs of the COSMIC-FFP method we found them to be very demanding concerning requirements document completeness, precision and redundancy identification. However, the additional effort of their construction and refinement seems to be justified also by not measurement-related benefits. Namely, the quality enforced by measurement needs makes the later steps of development easier and promotes code reuse. Thus a synergy between measurement method and general development process needs can be observed. The most original of formulated postulates is avoiding of problematic non-functional factors by modeling those requirements functionally in the spirit of aspect-oriented programming.

Our future research will include experimental application of measurement method to full size software development project. This will require the refinement of method's rules. The most prominent issues here include:

- Detailed criteria of partitioning the system into separately counted units.
- Architecture-specific templates for modeling multi-tiered software.
- Patterns for modeling of various non-functional requirements as aspects.

The adequacy and universality of proposed use case specification template will be also verified.

## References

- [AD01] B. Anda, H. Dreiem, D. I. K. Sjøberg, M. Jørgensen: *Estimating Software Development Effort Based on Use Cases-Experiences from Industry*, 2001, p. 487-502.
- [AD03] A. Abran, JM. Desharnais, S. Oigny, D. St-Pierre, C. Symons: *COSMIC-FFP Measurement Manual (The COSMIC Implementation Guide For ISO/IEC 19761: 2003) Version 2.2*, Common Software Measurement International Consortium, 2003. <http://www.lrgl.uqam.ca/cosmic-ffp>
- [AM00] F. Armour, G. Miller: *Advanced Use Case Modeling: Software Systems*, Addison-Wesley, 2000.
- [AM04] S. W. Ambler: *The Object Primer 3rd Edition*, Cambridge University Press, 2004.
- [Co00] Alistair Cockburn: *Writing Effective Use Cases*, Addison-Wesley 2000
- [I1-98] ISO/IEC 14143-1:1998 "Information Technology – Software measurement – Functional size measurement – Part 1: Definition of concepts", International Organization for Standardization, 1998. <http://www.iso.org>
- [I2-03] ISO/IEC 19761:2003 "Software engineering – COSMIC-FFP – A functional size measurement method", International Organization for Standardization, 2003. <http://www.iso.org>
- [I3-03] ISO/IEC 20926:2003 "Software engineering – IFPUG 4.1 Unadjusted functional size measurement method – Counting practices manual", International Organization for Standardization, 2003. <http://www.iso.org>
- [I4-02] ISO/IEC 20968:2002 "Software engineering – Mk. II Function Point Analysis – Counting Practices Manual", International Organization for Standardization, 2002. <http://www.iso.org>
- [Ka93] G. Karner: *Metrics for Objectory. Diploma thesis*, University of Linköping, Sweden. No. LiTHIDA-Ex-9344:21,1993.
- [Ki97] G. Kiczales, J. Lamping, A. Mendhekar, Ch. Maeda, C. Videira Lopes, J.-M. Loingtier, J. Irwin: *Aspect-Oriented Programming*, ECOOP 1997, p. 220-242.
- [Lo04] D. Longstreet: *Function Points Analysis Training Course*, Longstreet Consulting Inc., 2004. [www.softwaremetrics.com](http://www.softwaremetrics.com)
- [Mk98] *MK II Function Point Analysis Counting Practices Manual*, United Kingdom Software Metrics Association (UKSMA), 1998.
- [OM03] Object Management Group: *Unified Modeling Language (UML) Specifi-*

*ation. Version 1.5, 2003. <http://www.omg.org>*