Comparison of Method Chunks and Method Fragments for Situational Method Engineering

Brian Henderson-Sellers
University of Technology, Sydney, Australia
brian@it.uts.edu.au

Cesar Gonzalez-Perez
IEGPS, CSIC Spain
cesargon@verdewek.com

Jolita Ralyté
University of Geneva, CUI, Switzerland
Jolita.Ralyte@cui.unige.ch

Abstract

Two main candidates for the atomic element to be used in Situational Method Engineering (SME) have been proposed: the "method fragment" and the "method chunk". These are examined here in terms of their conceptual integrity and in terms of how they may be used in method construction. Also, parallels are drawn between the two approaches. Secondly, the idea of differentiating an interface from a body has been proposed for method chunks (but not for method fragments). This idea is examined and mappings are constructed between the interface and body concepts of method chunks and the concepts used to describe method fragments. The new ISO/IEC 24744 standard metamodel is used as a conceptual framework to perform these mappings.

1. Introduction

Both in theory and practice, it is argued [35] that high quality software development methods (a.k.a. methodologies: [21]) can best be created by means of *construction* – identifying small elements of a methodology, variously called fragments or chunks, and putting them together for a specific situation. Done in an *ad hoc* manner, this can lead to a large number of variations on the one methodology within a single organization together with its concomitant challenge of methodology management [25]. A more structured, formal and potentially repeatable approach to methodology construction is Situational Method Engineering (SME) [23], which is a subset of the IT sub-discipline of Method Engineering (ME) that itself includes not only SME but also comparison of methods and knowledge infrastructures [5]. SME provides a solid, theoretically sound basis for creating useful methodologies as well as giving the development team "ownership" of their methodological approach [14]. The chunks/fragments, of course, need to exist prior to construction – typically these are gleaned from best practice, theory and/or abstracted from other (static) methodologies. Once identified and documented, they are stored in a methodbase, which serves as a repository for these method chunks/fragments [6, 13, 30, 33, 34, 42].

While there are many research and practical issues regarding the engineering of software development methodologies, as documented in the software engineering literature on SME, one important basic concern is the need to agree on a definition of the atomic element from which methodologies can be constructed (see also panel discussion at ME07 [1]). There have been several proposals in the literature, variously known as a method chunk, a method fragment or a method component (or sometimes process component). In each case, the overall definition relies on an appropriate metamodel. In this paper, we examine the various proposals for an appropriate atomic element. Furthermore, since many authors, in describing their metamodel, make the assumption that such a method component is best described in terms of a body and an interface [7, 27], we further our analysis of fragments versus chunks to elucidate the value of using such a partitioning for the description of the atomic elements for situational method engineering. The analysis is based on a combination of qualitative descriptions of the components themselves (Sections 2-4.1) plus an analysis of the need for an interface/body model for method components (Section 4.2). Throughout we use the ISO/IEC International Standard 24744 [20] Software Engineering Metamodel for Development Methodologies as an underpinning "theory" that assists us in identifying similarities and differences in these two approaches to Situational Method Engineering (SME).

2. Method Fragments

As noted in [35, 36], the term *method fragment* was coined by Harmsen *et al.* [13] (and also reiterated in

[3]) by analogy with the notion of a software component – see also [38, 42]. Although ter Hofstede and Verhoef [43] define a method fragment as "a coherent part of a metamodel, which may cover any of the modelling dimensions at any level of granularity", thus envisaging a "fragment" as a portion of the metamodel itself, it is more widely supposed that a fragment is an element *generated* from the metamodel – usually by instantiation. In the latter, more generally accepted case, many authors discriminate between two kinds of method fragments: process fragments and product fragments [3, 6, 12, 13, 22, 32, 38]. The fragments generated, for instance, in the OPEN Process Framework [10] are each instantiated from a single class in the metamodel and are weighted towards specifying process elements (as noted by [34]) as are those of [26] in the JECKO framework; whereas fragments that could be extracted by sub-setting from OMT, OOSE or UML are more likely to be product-focussed fragments [34] as are the proposals of [18]. Thus the generated atomic SME element is generally regarded as either a process-focussed fragment (e.g. a kind of task or technique) or a product-focussed fragment (a kind of diagram, document or other work product). Using the metamodel of the OPF or of ISO/IEC 24744 (Fig. 1), for example, these method fragments are defined separately (process-only fragments or product-only fragments[1] – see Table 1) and then linked together. This is in contrast to the notion of a method "chunk" (Section 3), defined as a pre-determined linkage of only one process-oriented component with one product-oriented one i.e. a one-to-one relationship rather than many-to-many as in the OPF or the new ISO/IEC 24744 [20] standard metamodel.

The fact that method fragments are not encapsulated together into chunks does not mean that there are no relationships between them. All the fragment-based approaches utilise some kind of association between process- and product-oriented fragments to capture the appropriate dependencies. This is best illustrated by ISO/IEC 24744, which models this relationship as a complete class, named *ActionKind*, representing a single usage event that a given process fragment exerts upon a given product fragment. This class contains an attribute, *Type*, that specifies what kind of action the process part is exerting on the product part. For example, imagine a methodology that contains a requirements validation task. This task takes a draft requirements document as input and modifies it accordingly through the validation process, creating, as well, a requirements defect list. Modelling this task plus the two involved products (one of which is both an input *and* an output) can be easily modelled by using two actions: one action would map the requirements validation task to the requirements document, specifying a type "modify", and a second action would map the same requirements validation task to the requirements defect list, specifying the type as "create".

[1] Or indeed producer-only fragments – not discussed further here.

Table 1. Two example fragments, the details of each following the standard template as specified by the ISO/IEC 24744 metamodel.

a) Example of process-focussed fragment (an instance of the meta-element TaskKind)
Attributes Name: Analyze requirements Purpose: Study, understand and formalise requirements previously elicited. Description: A full textual description would be here. (Omitted for lack of space). Minimum capability level: 1 **Relationships** Causes (Action kinds): ▪ Modifies Requirements Specification Document, mandatory. Results in (Outcomes): ▪ Requirements have been analysed and understood. Includes (Task kinds): ▪ (none) Is involved in (Task-technique mapping kinds): ▪ A textual description of the linkage to various appropriate techniques would be inserted here (Omitted for lack of space). Is involved in (Work performance kinds): ▪ Business Analyst, mandatory. ▪ Customer, recommended.
b) Example of product-focussed fragment (an instance of the meta-element WorkProductKind
Attributes Name: Requirements Specification Document Description: A full textual description would be here. (Omitted for lack of space). **Relationships** Is acted upon by (Action kinds): ▪ Created by Elicit requirements, mandatory. ▪ Modified by Analyze requirements, mandatory. ▪ Modified by Document requirements, mandatory.

- Read by Develop class models, mandatory.

The relationships between process- and product-oriented fragments are thus clearly specified. (It must be noted that the actions are lightweight entities in the methodology that act as mappings between heavyweight process- and product-oriented fragments. Actions are not containers, as are chunks.)

3. Method Chunks

In contrast to the process-only or product-only fragment discussed in Section 2, other authors prefer the concept of a *method chunk* [30, 33-36, 38, 39] in order to emphasize the more constructive[2], collection notion. Here, a chunk is *the combination of a process part (also called a guideline) plus a product part* although, interestingly, the third major element, a metaclass to represent "Producers" (largely people), as identified in the OPF, SPEM and ISO/IEC 24744 metamodels (Fig. 1), is missing from the chunk-based approach (Fig. 2). In [36], a method chunk is thus a tightly coupled (process+product) representation[3]. In this approach, a method chunk is "an autonomous and coherent part of a method … supporting the realization of some specific ISD [information systems development] activity". The relationship "refers to" between a process part and a product part is an abstraction of all kinds of transformations (construction, modification, etc.) that the process part realises upon the product part. In Fig. 3, the process-focussed part of the chunk is illustrated on the left hand side as a "map" [40] while the right hand side is a class model depicting a particular product part, here expressed as a metamodel of the product to be constructed by applying this method chunk. The chunk captures the guidelines allowing production of the work product in the process portion of the chunk together with definitions of the concepts used in the product part [34, 36]. Although in any particular situational method there is one process part (say ProcA) connected to one product part (say ProdB), it is perfectly possible that in another situation (at a different time or in a different project) that ProdB may be linked to a different process part, say ProcC[4].

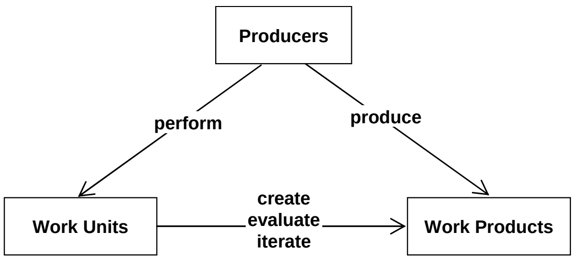


Fig. 1. The triangle of Producers, Work Units and Work Products that underpins the SPEM, OPF and 24744 standards for software engineering process modelling. Method fragments conform to one specific subclass of these metaclasses.

The process+product chunk approach states that a chunk has not only a body, consisting of the process plus product part as discussed above, but also an interface (Fig. 2). This interface describes the methodological situation where the chunk can be applied plus the intention (objective) it allows to be achieved, effectively defining the pre- and post-conditions for the chunk. This approach would therefore appear to embody, at a high abstraction level, a strong traditional process/workflow mindset, seemingly developing from its origins in Guidelines and Maps [40] in which a process-focussed fragment could be envisaged at a high level as being a "black box" acting as a transformation engine to change the input into an output (as in ISO/IEC 12207 [19]) for example). Once a chunk has been selected according to the methodological needs by looking at its interface, the evaluation process, based on similarity measures described by [35] retrieves the method chunk body – process plus product – and incorporates it to the methodology being constructed.

In addition to the interface and the body, each method chunk has a *Descriptor* [27, 36, 37] (Figs. 2 and 3). The descriptor extends the contextual view captured in the chunk interface to define the context in which the chunk can be reused. Besides the information relevant to the chunk identification, such as *ChunkName*, *ChunkID*, *Type* and *Objective,* the descriptor defines the *Origin* of the method chunk, the *Reuse Situation* and the *Reuse Intention.* The reuse situation captures a set of criteria taken from the reuse frame proposed by [27] and characterizes the project situation where the method chunk is useful. Mirbel and Ralyté [27] suggest that the reuse intention, which describes the objective of the chunk and has the same structure as the intention of the chunk, can be formally stated as verb + target + parameters [31]. For example,

[2] This use of positive, constructive language is echoed by Don Firesmith who recommends the term *method component* rather than *method fragment* (see http://www.opfro.org/).

[3] Note that, confusingly, Kraiem *et al.* [22], while using this same approach, state that a method fragment is called a chunk.

[4] This led to the use of a one-to-many cardinality between Process part and Product part (as in Fig. 4) to indicate temporal changes in bonding – although that is a non-standard use of UML cardinalities.

the informal method chunk intention “Construct a use case model following the OOSE approach” can be

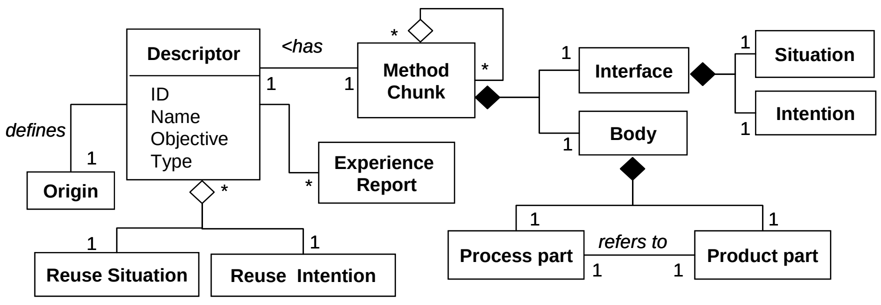
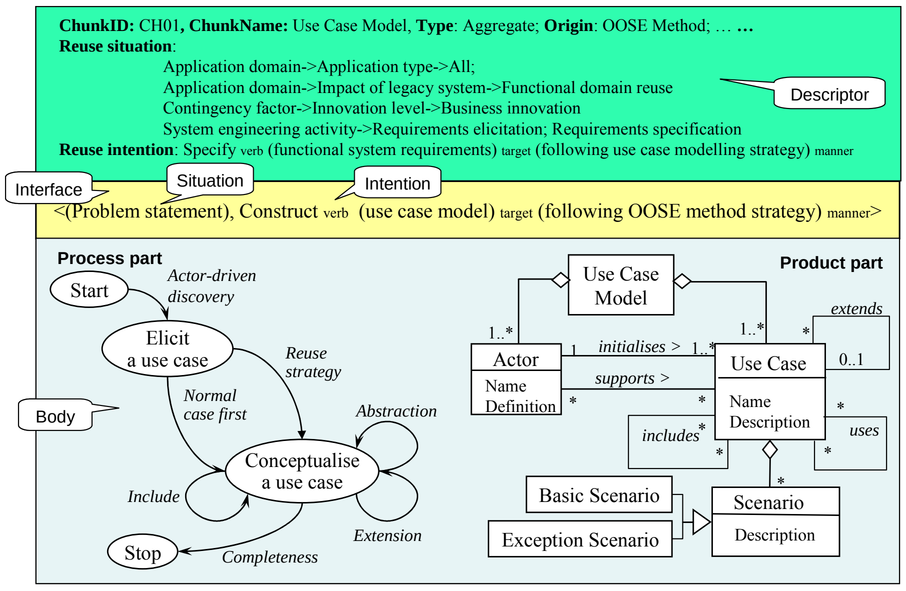


Fig. 2. Revised metamodel for method chunk (derived from [37]).



ChunkID: CH01**, ChunkName:** Use Case Model, **Type**: Aggregate; **Origin**: OOSE Method; … **…**
Reuse situation:
Application domain->Application type->All;
Application domain->Impact of legacy system->Functional domain reuse
Contingency factor->Innovation level->Business innovation
System engineering activity->Requirements elicitation; Requirements specification
Reuse intention: Specify verb (functional system requirements) target (following use case modelling strategy) manner
Descriptor
Interface
Situation
Intention
<(Problem statement), Construct verb (use case model) target (following OOSE method strategy) manner>
Process part
Start
Actor-driven discovery
Elicit a use case
Reuse strategy
Normal case first
Body
Abstraction
Conceptualise a use case
Include
Extension
Stop
Completeness
Product part
Use Case Model
1..*
1..*
extends
*
Actor
Name
Definition
1
initialises >
1..*
supports >
*
*
Use Case
Name
Description
0..1
*
uses
*
includes
*
*
*
Basic Scenario
Exception Scenario
Scenario
Description

Fig. 3. An example of a method chunk, consisting of a single process part and a single product part. A chunk has a body plus an interface as well as an affiliated descriptor. Details of the origin, reuse situation, reuse intention and experience report have been only partially presented in order to retain clarity.

reformatted as Construct$_{\text{verb}}$ (a use case model)$_{\text{target}}$ (following the OOSE approach)$_{\text{parameter=manner}}$. The reuse intention of this method chunk would be formalised as Specify$_{\text{verb}}$ (functional system requirements)$_{\text{target}}$ (following use case modelling strategy)$_{\text{parameter=manner}}$.

Descriptors also have connections to other elements, such as the *Origin* of the chunk (i.e. from which method was it derived/abstracted) and *Experience Reports* capturing experience gained from previous usage of the chunk. Mirbel and Ralyté [27] also propose to specify for each chunk its components and aggregates and the incompatible and alternative chunks and to provide examples.

To summarise, the descriptor contains information to help the method engineer to select the right method chunk in the situation at hand i.e. it characterizes typical situations in which the chunk may be useful – contextual or situational knowledge that is not knowledge relating to how software is developed but only to helping a method engineer to find the chunk in a methodbase. After a chunk has been selected, the Descriptor information is redundant such that only the body and interface parts of the method chunk are used thereafter i.e. during method construction and evaluation.

Finally, since chunks can be at any granularity (see also [38]), it is argued [34] that a full methodology itself can also be regarded as a chunk. This is similar to the model adopted more recently in SPEM Version 1 [28] in which a *Process* is modelled as a special kind of *ProcessComponent*. Since, by definition, a chunk is *one* process-focussed fragment plus *one* product-focussed fragment, this could work for fragments. However, we do not think that a full software engineering process (SEP) can be envisaged as being a fragment itself. In other words, there is no meaningful way to model a full SEP as a combination of one process-focussed fragment plus one product-focussed fragment, except at the most abstract level i.e. not in the endeavour domain where a methodology is enacted on a specific (situational) project.

3.1. Alternative Views on Chunk

A different definition of "method chunk" is given in [44], who instead use the concept of a *method component* defined as a "self-contained part of a system engineering method expressing the process of transforming one or several artefacts into a defined target artefact and the rationale for such a transformation". This has some similarity with the notion of a process in the ISO 12207 standard [19]. A method component also consists of two parts but these are, in contrast to the method chunk, its content and its rationale. This emphasis on "rationale" is a key part of this approach and has many similarities with the notion of Guideline in ISO/IEC 24744 and Descriptor in the method chunk approach (see also [1]).

In some contrast, Rupprecht *et al.* [41] talk about "process building blocks" that know how they should be connected to other process building blocks (although their context is manufacturing processes not software development processes).

4. Comparisons

4.1. The Pros and Cons of Fragments and Chunks

Since the chunk-based approach combines process and product parts into a single chunk whereas these are kept separate in the OPF and ISO/IEC 24744, it is reasonable to suggest the use of a supertype of *Method Component* with subtypes of *Method Chunk* and *Method Fragment* (Fig. 4). This suggestion was also made in [27], although these authors go further and include two other subtypes of *Method Component*: *Process Pattern* and *Product Pattern* (not shown nor discussed further here). In fact, the process part of the chunk can be easily associated to the notion of one process fragment producing one product fragment that is defined as product part of the chunk.

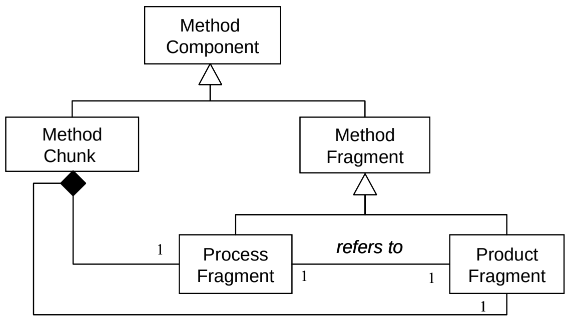


Fig. 4. Metamodel for Method Components, Chunks and Fragments.

We should also reiterate that, while Fig. 4 shows both the process and product aspects of fragments and chunks, there is no mention of a metaelement to represent *Producers* – a term that includes the people involved in software development, the roles they play and the tools they use. This concept, embodied in OPEN [9], SPEM [28] and ISO/IEC 24744 [20], is critical for creating a quality situational method and we recommend its inclusion (Fig. 1) in future versions of chunk models. However, the type of the Producer is not completely forgotten in the chunk-driven approach. The reuse frame proposed in Mirbel and Ralyté [27] includes the category of criterion "Human", which allows the specification of the type of the "producer" (analyst, designer, developer, etc.) and the required knowledge level (beginner, medium, expert). Therefore, this criterion can be included of the chunk descriptor as an element of the reuse situation.

There has been much debate about the efficacy of a method chunk as compared to a method fragment for SME. In essence, as noted above (Fig. 3), a method chunk is a conceptual combination of two method fragments: one process-focussed fragment and one product-focussed fragment. The advantage of such a combination is argued to be the speed of usage insofar as there are often a smaller number of chunks required for any specific situation and hence a small number that need to be located from the methodbase. Offsetting this to some degree is the fact that many of these chunks may contain the same product part. In other words, there is a potential disadvantage as a result of the fact that such a process-product linkage is neither one-to-one nor unique in real-life scenarios. Indeed, if all such linkages *were* one-to-one, then the flexibility of method construction offered by SME would be totally redundant since everything would be "hard-wired". In reality, for instance, some techniques and work products can be used with more than one task such that several method chunks may contain the same product part but a different process part [34]; some tasks have multiple output products (one to many); some tasks modify existing products or have multiple inputs – and there are other examples in industry situations where a one-to-one linkage is not viable.

When such many-to-one situations occur, with the existing chunk model of Fig. 4, a separate one-to-one chunk for *each* specific configuration needs to be created such that for instance, there is one chunk for one process fragment plus one product fragment and a second chunk for the same product fragment but different process fragment (i.e. different guideline to obtain the same output product). However, the chunk approach does not allow one to associate the same process fragment with different product fragments because the process part of the method chunk is closely related to its product part. For instance, it is impossible to define one chunk with one process fragment plus one product fragment; a second chunk for the same process fragment but with two different output product fragments, a third one for three outputs and so on. In each of these cases, the process fragment would also be different. Such duplication, across several chunks, could thus lead to both degradation of quality of the usage of the methodbase overall and to a maintenance problem analogous to the reuse issue that object technology originally sought to remove although this would be ameliorated to some degree at the implementation level since database technology can be used to ensure that only one copy of a fragment exists physically in the repository or methodbase (i.e. storage needs to be "by reference" and not "by value").

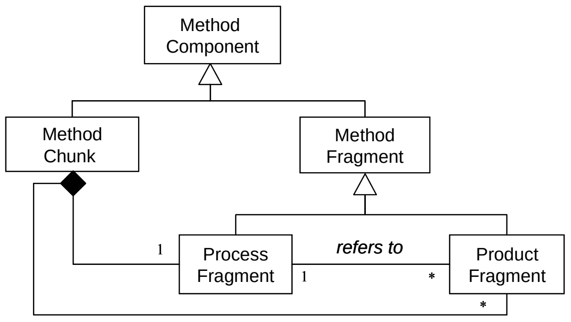


Fig. 5. Metamodel for Method Components, Chunks and Fragments with cardinalities revised to support a 1:m relationship between process and product parts of the chunk.

We therefore recommend a revision of the cardinalities in Fig. 4 as shown in Fig. 5.

A second difference in fragment- and chunk-based approaches is the expression of the relationships between the product and process fragments/parts. In the fragment-based approaches the relationships between process- and product-oriented fragments are clearly specified by defining the type of action the process fragment is exerting on the product fragment. These relationships are mainly used to find the right pair of fragments (product fragment and process fragment).

In the chunk-based approach the relationship between the process and product parts of a chunk does not have the same role as it is not necessary to search for product and process parts separately. However, it is expressed by the chunk's *Intention*. For example, the intention of a chunk: "Create a Use Case model" states that the process part provides guidelines "to create" the product "a use case model". The intention is one of the parameters used to select the appropriate method chunks in a given situation.

Despite these differences, fragment-based and chunk-based approaches share a number of commonalities. To start with, both acknowledge the need to capture information about the situation where usage of any particular method component may make sense. In fact, this is a crucial aspect of *situational* method engineering, and hence its name. Chunk approaches implement this via the chunk interface plus descriptor, which centralise situational information in a single place. In ISO/IEC 24744, as an example of a fragment-based approach, information has been modularised using different criteria, and situational information is distributed across different classes. First of all, the *Guideline* class is designed to capture information about where and how a method fragment (or collection thereof) can be used. Secondly, the *MinCapabilityLevel* attribute of the *WorkUnitKind* class captures the minimum capability or maturity level at which a particular process-oriented fragment is meant to be used, thus contributing to the establishment of a methodological situation.

Information about the intention of using a particular component is also captured by both approaches but, again, in different manners. The chunk approach uses an explicit intention description within the chunk interface. ISO/IEC 24744, on the other hand, captures intention in a more heterogeneous (and, possibly, richer) way. Two types of intention are distinguished: the intention of selecting a particular method fragment, and the intention of performing a particular process-oriented fragment (a work unit) or creating a particular product-oriented fragment (a work product). The first kind of intention (why a fragment has been selected) is expressed by the dependencies that exist between process-oriented and product-oriented fragments and are implemented by the *ActionKind* class, as described in Section 2; the products being created or modified by the enactment of the process fragment *are* the intention of selecting it. The intention of a product fragment, similarly, is given by the process fragments that modify, destroy or read the product fragment. With regard to the second kind of intention (why a certain process-oriented fragment must be enacted), the *Purpose* attribute of the *WorkUnitKind* class captures this information.

Another similarity between fragment-based and chunk-based approaches is related to capturing information that may complement the specification of a method component, such as bibliographic references. The chunk approach manages this through chunk descriptors, while ISO/IEC 24744 implements it through classes such as *Reference* and *Source*.

These comparisons are summarized in Table 2.

Table 2 Summary comparison

	Fragments	**Chunks**
Support for process	Yes	Yes
Support for work products	Yes	Yes
Support for producers	Yes	No
Attributes of element	Dependent upon type (see Table 1)	Always the sum of process part plus product part (see Fig. 2). Use of Descriptor important
Connection between process and product parts	Ad hoc based on situation	Hard-wired
Situational information	Guideline, Reference and Source	Interface and Descriptor
Capability assessment	MinCapability Level	No
Multiple inputs and outputs to a process element	Yes	No

4.2. The Pros and Cons of Interfaces for Method Components

As discussed above, method fragments relate to a single concept in the metamodel (process- or product- or producer-focussed: Fig. 1) whereas method chunks relate to a pair (process plus product), with the notion of method component introduced as the generic supertype to both chunks and fragments. In this section, we analyse the usefulness of dividing a method component into a body plus an interface as used by many method engineering theorists (e.g. Fig. 2).

The rationale for using an interface versus body description of a method component must not be assumed to come from the same source as the arguments that led to its use in OO programming. The choice of terms (“interface” and “body”) may lead the reader to believe that this discrimination is based on the original ideas on information hiding in [29]. Object-oriented programming introduced the idea that it was important (critical in fact) to separate the implementation (i.e. the body) of a coded class from its interface. There were probably three main reasons for this: (a) to isolate the variability of the implementation from the interface, so that the implementation could change without affecting the interface; (b) to actually hide non-disclosable data from users; and (c) to contain any run-time errors and avoid their propagation to other components [24]. However, in the method chunk approach, no information needs to be hidden; both body and interface are equally visible to the user. These terms (“body” and “interface”) are used with a different meaning, aiming to represent, respectively, the chunk itself (that gets incorporated into the methodology when selected) and the situation/intention contextual information about the chunk (Fig. 3). This is useful from a conceptual modelling point of view, allowing a black box interpretation of the chunk as a transformation engine at one point of time during chunk selection (i.e. use of the “interface”) and then later, after the chunk has been selected, the main “body” of the chunk description becomes relevant to the method engineer. The initial phase is in accord with the process decomposition philosophy underpinning the method chunk approach that, although chunks are presented as (equally) a product part plus a process part, in reality the process part dominates, particularly in the creation and selection of that chunk. Indeed, much of the method chunk literature stresses that the approach is process-driven.

4.3. Choosing Between the Two Approaches

The chunk approach offers simplicity of archival and selection that therefore matches well simple situational method engineering challenges. If the requirements for the method construction only need one-to-one linked process+product fragments and the personnel and tools involved are minimalist (and matching them into the chunks can be done by hand), then the chunk approach could work well – although to the best of our knowledge there are no industry case studies using the chunk approach.

Fragments, on the other hand, require a slightly deeper understanding of the architecture of the repository and the way that fragments can be linked e.g. using ActionKind of the ISO/IEC 24744 International Standard. However, the linkages achieved are more flexible and support a wider range of conceptual amalgamation, permitting to create, read and write access depending upon the specific situation. This approach also permits ad hoc many-to-many relationships, whilst retaining individuality of fragments stored in the methodbase (repository). A fragment-based approach to SME has been successfully used in a number of industry projects e.g. [2, 8, 16, 17].

Thus, in both engineering methodologies and using them on software engineering development projects, the process revolves around identification of the fragments/chunks and their linking together as appropriate. At the same time, it must be ensured that the resulting methodology has both quality as a static methodology model, is internally consistent and, most importantly, when applied to a real endeavour (e.g. a software application development) adds value to the software engineering organization. Current successful applications of SME have in fact undertaken these construction and quality assessment steps manually. However, repository tools to provide both higher quality construction and semi-automated assistance in method construction as well as overall management of the chunks/fragments contained in the repository are sorely needed. One such example, still in prototype form, is MethodComposer (see discussion in [11]). This tool supports the population of the repository with fragments and their retrieval to construct a method, together with initial support for project enactment. Third party commercial companies are also likely to make announcements of commercial tools supporting such a 24744-based approach[5].

4.4. Further Work

We have argued the similarities and differences of the method chunk and method fragment approaches to SME on conceptual/theoretical grounds. Whilst clearly it is critical for the metamodel to be of good quality (see for example discussions on good and bad aspects of metamodelling in [15]), further evaluation in a more practical and pragmatic context is also useful. This is the topic of further work.

As noted at the end of the previous subsection, we are aware of at least one commercial tool being developed and the availability of this and similar tools in the near future will also allow us to evaluate the utility of tool-based SME in contrast to the more manual approach to method construction currently used in industry e.g. [2, 17].

Indeed (as commented also by an anonymous reviewer), it has long been recognized that probably the most important currently unsolved issue in SME is formulating "rules" for both method construction and for quality evaluation of the constructed method e.g. [4], Having an underpinning International Standard [20] provides a firm basis for such future work.

5. Summary and Conclusions

In the context of Situational Method Engineering (SME), we have evaluated two issues regarding the definition and descriptions of the atomic elements that are stored in a method fragment repository or methodbase. Firstly, we have contrasted the models for a method fragment, which depicts either a solely process-focussed concept, a product-focussed concept, or indeed (although not discussed in detail here) a producer-focussed concept, with that for a method chunk, which is a combination of a single process-focussed fragment with a single product-focussed fragment. From a conceptual modelling point of view, insisting on a one-to-one relationship between process fragment and product fragment often creates an artificial model that does not relate simply to real-life requirements; consequently we have recommended a modification to allow for multiple product parts (Fig. 5). From a software engineering point of view, there is a possibility that this chunk approach loses the flexibility that is at the core of SME and may introduce potential maintenance problems.

[5] The details are commercially confidential at the time of writing, made available to the authors under a non-disclosure agreement.

In terms of capturing situational information, we found that the chunk and fragment approaches do this equally well but with different mechanisms. Situational information is captured in the chunk approach in the chunk interface whereas in the fragment approach, as embodied for instance in ISO/IEC 24744, uses dependency relationships and an *ActionKind* class in its metamodel. Bibliographic information is also captured differently in the two approaches: chunk descriptors or implemented (in the 24744 approach) by *Reference* and *Source* classes in the metamodel.

Our analysis of the use of body and interface as terms in describing chunks identifies a different meaning from what a programmer might infer: information hiding. Rather, the use of these terms in the chunk approach identifies, at the conceptual level, knowledge of the chunk and knowledge of the situation/intention of the chunk.

Throughout this analysis, we have used the new ISO Software Engineering Metamodel for Development Methodologies [20] as a means of providing a theoretical underpinning for our identification of similarities between the chunk and fragment approaches and for the mappings between them.

6. Acknowledgments

A brief overview of this topic was presented as a poster at the CAiSE Forum 2007 in Trondheim. This is Contribution Number 07/19 of the Centre for Object Technology Applications and Research, Sydney, Australia. This work was also partially supported by the Swiss National Science Foundation. N° 200021-103826.

7. References

[1] Ågerfalk, P.J., Brinkkemper, S., Gonzalez-Perez, C., Henderson-Sellers, B., Karlsson, F., Kelly, S. and Ralyté, J., Modularization constructs in method engineering: towards common ground? In *Situational Method Engineering: Fundamentals and Experiences* (eds. J. Ralyté, S. Brinkkemper and B. Henderson-Sellers), Springer, New York, NY, USA, 2007, pp. 359-368.

[2] Bajec, M., Vavpotič, D. and Krisper, M., Practice-driven approach for creating project-specific software development methods. *Inf. Software Technol.* **49,** 2007, pp**.** 345-365.

[3] Brinkkemper, S., Method engineering: engineering of information systems development methods and tools. *Inf. Software Technol.* **38(4),** 1996, pp. 275-280.

[4] Brinkkemper, S., discussions with the first author at University of Utrecht, July 2005.

[5] Brinkkemper, S., Helms, R. and van de Weerd, I., Method engineering, http://www.cs.uu.nl/education/vak.php?vak=INFOME&jaar=2006, 2006, Accessed 26.3.07.

[6] Brinkkemper, S., Saeki, M. and Harmsen, F., Assembly techniques for method engineering. *Advanced Information Systems Engineering. 10th International Conference, CAiSE'98, Pisa, Italy, June 8-12, 1998, Proceedings,* LNCS1413 (eds. B. Pernici and C. Thanos), Springer Verlag, 1998, pp. 381-400

[7] Cossentino, M., Gaglio, S., Henderson-Sellers, B. and Seidita, V., A metamodelling-based approach for method fragment comparison. *CAiSE'06. 18th Conference on Advanced Information Systems Engineering – Trusted Information Systems. Luxembourg 5-9 June, 2006. Proceedings of the Workshops and Doctoral Consortium* (eds. T. Latour and M. Petit), Namur University Press, Belgium, 2006, pp. 419-432,

[8] Coulin, C., Zowghi, D. and Sahraoui, A-E-K., A situational method engineering approach to requirements elicitation workshops in the software development process. A situational approach to requirements elicitation workshops. *Software Process: Improvement and Practice* **11(5)**, 2006, pp. 451-464

[9] Firesmith, D.G. and Henderson-Sellers, B., Improvements to the OPEN process metamodel. *JOOP/ROAD* **12(7),** 1999, pp. 30-35.

[10] Firesmith, D.G. and Henderson-Sellers, B., *The OPEN Process Framework. An Introduction*. London, Addison-Wesley, 2002.

[11] Gonzalez-Perez, C., Tools for an extended object modelling environment. *10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS),* 16–20 June 2005, IEEE Computer Society Press, 2005, pp. 20–23,

[12] Harmsen, A.F., *Situational Method Engineering*. Moret Ernst & Young, 1997

[13] Harmsen, A.F., Brinkkemper, S. and Oei, H., Situational method engineering for information systems projects. *Methods and Associated Tools for the Information Systems Life Cycle. Procs. IFIP WG8.1 Working Conference CRIS/94* (eds. T.W. Olle and A.A. Verrijn-Stuart), North Holland, Amsterdam, 1994, pp. 169-194

[14] Henderson-Sellers, B., Method engineering for OO system development. *Comm. ACM* **46(10),** 2003, pp. 773-789.

[15] Henderson-Sellers, B., 2007, On the challenges of correctly using metamodels in method engineering, keynote paper in *New Trends in Software Methodologies, Tools and Techniques. Proceedings of the sixth SoMeT_07* (eds. H. Fujita and D. Pisanelli), IOS Press, 2007, pp. 3-35.

[16] Henderson-Sellers, B. and Qumer, A., Using method engineering to make a traditional environment agile. *Cutter IT Journal* **20(5),** 2007, pp. 30-37.

[17] Henderson-Sellers, B. and Serour, M.K., Creating a dual agility method - the value of method engineering. *J. Database Management* **16(4),** 2005, pp. 1-24.

[18] Hruby, P., Designing customizable methodologies. *JOOP* **Dec 2000,** pp. 22-31.

[19] ISO/IEC,: *Software Life Cycle Processes. ISO/IEC 12207.* International Standards Organization / International Electrotechnical Commission, 1995

[20] ISO/IEC, *Software Engineering. Metamodel for Development Methodologies. ISO/IEC 24744* International Standards Organization / International Electrotechnical Commission, 2007

[21] Jayaratna, N., *Understanding and Evaluating Methodologies: NIMSAD, a Systematic Framework.* McGraw-Hill, 1994

[22] Kraiem, N., Bourguiba, I. and Selmi, S., Situational method for information system project, presented at SSGRR 2000, L'Aquila, Jul 31 - Aug 06 2000 (http://www.ssgrr.it/en/ssgrr2000/papers/283.pdf)

[23] Kumar, K. and Welke, R.J., Methodology engineering: a proposal for situation-specific methodology construction. In *Challenges and Strategies for Research in Systems Development*. 257-269. Cotterman, W.W. and Senn, J.A. (eds.). John Wiley & Sons: Chichester, UK, 1992

[24] Meyer, B., *Object-Oriented Software Construction,* second edition. Prentice-Hall, 1997

[25] Mirbel, I., Method chunk federation. *CAiSE'06. 18th Conference on Advanced Information Systems Engineering – Trusted Information Systems. Luxembourg 5-9 June, 2006. Proceedings of the Workshops and Doctoral Consortium* (eds. T. Latour and M. Petit), Namur University Press, Belgium, 2006, pp. 407-418

[26] Mirbel, I. and de Rivieres, V., Adapting analysis and design to software context: the JECKO approach. *Procs. 8th Int. Conf. on Object-Oriented Information Systems (OOIS'02), Montpellier, France, September 2-5, 2002* (eds. Z. Bellahsène, D. Patel, C. Rolland) LNCS 2425, Springer-Verlag, 2002, pp. 223-228,

[27] Mirbel, I. and Ralyté, J., Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requirements Engineering* **11**,2006, pp. 58-78.

[28] OMG, *Software Process Engineering Metamodel Specification, formal/2002-11-14*. Object Management Group, 2002

[29] Parnas, D.L., A technique for software module specification with examples. *Communications of the ACM* **15(5),** 1972, pp. 330-336.

[30] Plihon, V., Ralyté, J., Benjamen, A., Maiden, N.A.M., Sutcliffe, A., Dubois, E. and Heymans, P., A reuse-oriented approach for the construction of scenario based methods. *5th Int. Conf. on Software Process (ICSP'98), Chicago, Illinois, USA*, 1998

[31] Prat, N., Goal formalisation and classification for requirements engineering. *Procs. 3rd Int. Workshop on Requirements Engineering: Foundations of Software Quality REFSQ'97,* Barcelona, 1999, pp. 145-156.

[32] Punter, H.T. and Lemmen, K., The MEMA model: towards a new approach for method engineering. *Inf. Software Technol.* **38(4),** 1996, pp. 295-305.

[33] Ralyté, J., Reusing scenario based approaches in requirements engineering methods: CREWS Method Base. *Procs. 10th Int. Workshop on Database and Expert Systems Applications (DEXA'99), 1st Int. REP'99 Workshop,* Florence, Italy, 1999

[34] Ralyte, J., Towards situational methods for information systems development: engineering reusable method chunks. *Procs. 13th Int. Conf. on Information Systems Development. Advances in Theory, Practice and Education* (eds. O. Vasilecas, A. Caplinskas, W. Wojtkowski, W.G. Wojtkowski, J. Zupancic and S. Wrycza), 271-282, Vilnius Gediminas Technical University, Vilnius, Lithuania, 2004

[35] Ralyté, J. and Rolland, C., An assembly process model for method engineering. *Advanced Information Systems Engineering* (eds. K.R. Dittrich, A. Geppert and M.C. Norrie), LNCS2068, Springer, Berlin, 2001, pp. 267-283

[36] Ralyté, J. and Rolland, C., An approach for method engineering. *Procs. 20th Int. Conf on Conceptual Modelling (ER2001),* LNCS 2224, Springer-Verlag, Berlin, 2001, pp. 471-484

[37] Ralyté, J., Backlund, P., Kühn, H. and Jeusfeld, M.A., Method chunks for interoperability. *Procs. ER2006* (eds. D.W. Embley, A. Olivé and S. Ram), LNCS 4215, Springer-Verlag, 2006, pp. 339-353

[38] Rolland, C. and Prakash, N., A proposal for context-specific method engineering. *Method Engineering. Principles of Method Construction and Tool Support. Procs. IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering, 26-28 August 1996, Atlanta, USA* (eds. S. Brinkkemper, K. Lyytinen and R.J. Welke), Chapman & Hall, London, 1996, pp. 191-208

[39] Rolland, C., Plihon, V. and Ralyté, J., Specifying the reuse context of scenario method chunks. *Advanced Information Systems Engineering 10th International Conference, CAiSE'98, Pisa, Italy, June 8-12, 1998, Proceedings* (eds. B. Pernici and C. Thanos), LNCS1413, 1 Springer-Verlag, 1998, pp. 91-218

[40] Rolland, C., Prakash, N. and Benjamen, A., A multi-model view of process modelling. Re*quirements Eng. J.* **4(4),** 1999, pp. 169-187.

[41] Rupprecht, C., Funffinger, M., Knublauch, H. and Rose, T., Capture and dissemination of experience about the construction of engineering processes. *Procs. 12th Conference on Advanced Information Systems Engineering (CAISE),* LNCS 1789, Springer-Verlag, Berlin, 2000, 294-308

[42] Saeki, M., Iguchi, K., Wen-yin, K. and Shinohara, M., A meta-model for representing software specification & design methods. *Procs. IFIP WG8.1 Conf on Information Systems Development Process, Come,* 1993, pp. 149-166.

[43] ter Hofstede, A.H.M. and Verhoef, T.F., On the feasibility of situational method engineering. *Information Systems* **22**(6/7), 1997, pp. 401-422.

[44] Wistrand, K. and Karlsson, F., Method components – rationale revealed, Advanced Information Systems Engineering 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings (eds. A. Persson and J. Stirna), LNCS 3084, Springer-Verlag, 2004, pp. 189-201.