Towards a Life Cycle for Method Engineering

Naveen Prakash, S.B. Goyal

[praknav @hotmail.com, goyalsb@yahoo.com]

GCET, 1 Knowledge Park Phase II, Greater NOIDA 201306

Abstract We propose a three stage method development life cycle. The requirements engineering phase consists of elicitation and representation of method intentions, the design phase produces the architecture of the method and the construction phase consists of organizing method features in a coherent whole. We concentrate in this paper on the Design and Construction phases of the life cycle. We explain our notion of method architecture and organization and illustrate them. Finally we show the relevance of method architecture and organization in SME. The design and construction engineering phases of our life cycle are illustrated for a small SME example.

Keywords: Method, Method Engineering, Meta-model, SME

1 Introduction

Traditionally, method engineering has relied on the basic technique of instantiation of meta models. This approach calls for a detailed identification of the *method features* of the method. In order to ensure that methods fit well in the project situation where they shall be used, Situational Method Engineering, SME [Har94], was developed. SME assumes the existence of a method repository from where the method(s) of interest are retrieved, and modified or assembled into a new method that is subsequently stored in the repository for reuse. Some early proposals for SME are [Bri98, Gru96, Gup01]. Recently goal oriented approaches have been proposed. Ralyte Ral03] suggests that method engineering should be done in two steps. First, a method engineering goal is established. Thereafter, the assembly based method engineering task is carried out by eliciting method engineering intentions like *add event concept*. Prakash et al [Pra07] propose a three stage SME process, intention matching, architecture matching, and organization matching.

Though method engineering has been investigated extensively, we are unaware of any proposal for a method development life cycle on which the engineering of methods can be based. Indeed, we believe that the development of such a life cycle is a pre-requisite for the systematic development of method engineering, including SME and propose to develop a method development life cycle (MDLC). In the proposed life cycle, SME as known today, is shown to be at the down stream end. The proposed life cycle calls for greater emphasis to be placed on the upstream activities of method design and method requirements engineering.

The layout of the paper is as follows. In the next section we present our method development life cycle. We show the inputs and outputs of the different stages as well as the essential process required to do SME. In section 3, we develop the notion of method architecture and give an example of a method architecture. In section 4, we define method organization and illustrate it. In sections 5 and 6 we deal with the processes of our life cycle relevant to SME. We present our processes for architecture and organization respectively and illustrate these through a small example.

2 Method Development Life Cycle

Our MDLC has been motivated by [Pra07]. As shown in Table 1, the Requirements Engineering stage consists of Intention Matching. First, the intention of the method To Be is elicited. The intention matching process uses synonym matching to identify intentionally similar methods that reside in the method repository. These methods become candidates for the second stage of the MDLC.

Stage	Process	Input	Output
Requirements Engineering	Intention Matching	Intention of the method To Be obtained from Interviews etc.	Intentionally similar methods to the method To Be
Design Engineering	Architecture Matching	Architectures of intentionally similar methods	Architecturally similar methods
Construction Engineering	Organization Matching	Organizations of architecturally similar method	Method To-Be

Table 1: The Method Development Life Cycle

In the Design Engineering stage, the method engineer retrieves the architecture of each candidate from the method repository. That subset of these components and inter-relationships is selected which best meets the broad functional needs of the method To-Be. Such selections are made from all the candidate methods and are synthesized together into the architecture of the desired method.

In the Construction stage the architecture is populated with instances of the method features needed in the method. These features are in accordance with the meta model used, fragment, contextual, decisional et..

3 Method Architecture

The Cambridge dictionary defines it as meaning "the art and science of designing and making buildings". Similarly, the Webster's dictionary defines it as the "discipline dealing with the principles of design and construction and ornamentation of fine buildings".

In computer science this term has been used in various contexts like Computer Architecture and Software Architecture. Computer architecture [Buc62] is the art of determining the needs of the user of a structure and then designing to meet those as effectively as possible. Mano describes CA [Man03] as the structure and behaviour of the computer as seen by the user. For Stallings [Sta96] Computer Architecture refers to those attributes of a system visible to a programmer. Clark says [Cla05] that computer architecture is about decomposition, interfaces and dependencies, and exploiting reusable parts as well as fundamental design principles and approaches.

Similarly, a number of views exist on software architecture. [Sub99] sees Software architecture (SA) as a combination of structural views of a system. Lane views it [Lan90] as the study of the large-scale structure and performance of software systems. [Kru94] says that SA deals with abstraction, with decomposition and composition, with style and esthetics. [Sha96] say that SA involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on those patterns. Yet another view is that architecture [Bas98] focuses on the externally visible properties of software components.

Following the above, **we define method architecture (MA) as an abstraction of the method that identifies its components and inter-relationships to highlight the externally visible functionality of the method.** Notice that we are talking about **the architecture of the method and not of tools** like CASE and CAME. Method architecture organizes the different features of methods into functionally coherent featues that can be recognized by application or method engineers as providing some useful capability.

Type	Urgency	Necessity	Abbreviation
1	Immediate	Must	IM
2	Immediate	Can	IC
3	Deferred	Must	DM
4	Deferred	Can	DC

Table 2: Types of Links

We adopt the basic notation of nested rectangles and links between these to represent a MA. Rectangles represent abstractions of method features. For links, we adopt the dependency types introduced in [Pra06]. Two attributes, urgency and necessity, are associated with each dependency type (see Table 2). Urgency refers to the time at which the dependent method, M_2, is to be enacted. If M_2 is to be enacted immediately after M_1 is enacted then this attribute takes on the value *Immediate*. If M_2 can be enacted any time, immediately or at any moment, after M_1 has been enacted, then urgency takes on the value *Deferred*. Necessity refers to whether or not the dependent method block M_2 is necessarily to be enacted after M_1 has been enacted. If it is necessary to enact M_2, then this attribute takes the value *Must* otherwise it has the value *Can*.

We define a link between rectangles of a method rectangle for each dependency type shown in Table 2. Thus, there are four kinds of links, IM, IC, DM, and DC.

It has been shown in [Pra02] that methods can be built for diverse domains, other than those of Information systems Development as well. Here we consider such a domain to illustrate our SME approach. As an example consider the architecture for Admit Student shown in Fig. 1. It consists of 5 rectangles, Process Regular Student, Process Distance-learning Student, Declare Result, Collect Fee and Register respectively. As shown, these are connected by links of the type DM, DC, and IM. Process Regular Student consists of nested method components, Publish Advertisement, Receive Application, and Conduct Examination respectively. Again these are connected by their own type of links as shown. Similarly, Process Distance-Learning Student has its own nested rectangles with appropriate links between nested components. Declare Result, Collect Fee, and Register are atomic components and display no nesting. These are connected by the types of links as shown in the Figure.

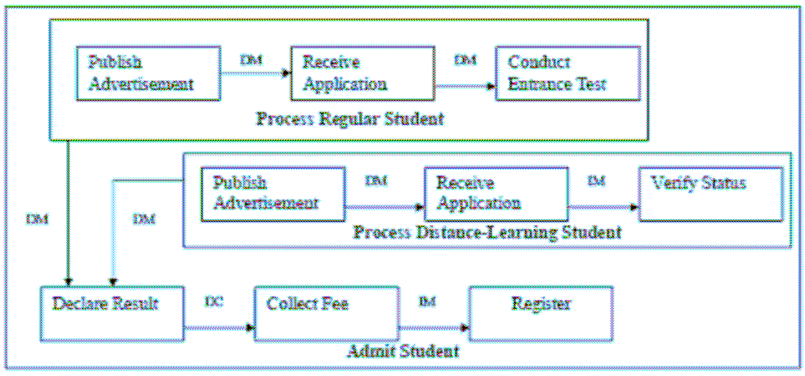

Fig. 1: Method Architecture of Admit Student

The architecture of Admit Student shows that the functionality the method provides is captured in the top level components, Process Regular Student, Declare Result etc. In this sense, it shows the externally visible functionality to the application engineer and/or to the method engineer. These rectangles are abstractions of the method features that shall be provided by a real method to meet these architectural specifications. Nested rectangles elaborate the architectural components of outer level rectangles. Functionality represented by nested rectangles is visible as part of the overall functionality provided by outer level rectangles.

4 Method Organization

The term organization has been used extensively in the context of computer organization. As far as the authors are aware, there is no notion yet of software organization. Stallings [Sta96] sees computer organization as the operational units and their interconnections that realize the architectural specifications. Mano [Man03] views computer organization as the way the hardware components operate and the way they are connected together to form the computer system. [Hay98] looks upon computer organization as the logical aspects of the implementation.

The notation for computer organization is very similar to that for computer architecture: nested rectangles and control or data links between these. However, rectangles of computer organization are at a lower level of abstraction than those of computer architecture and refer to the physical units that realize the architectural specifications.

We can define method organization by analogy with computer organization. **Method organization is the way in which method features are connected together to realize the MA**. Method organization is the manner in which MA is implemented. It is an elaboration of MA to determine how the method is constructed, what capabilities it provides, what constraints are applicable etc. Again, a method organization is represented using nested rectangles and links between these. The links are of the same four types as for method architectures.

In illustrating method organization, we first note that it is in accordance with a model of a method. This may be a meta-model [Har94, Gro97, Pra97, Pra99], or alternatively it can be based on a generic model [Pra06]. Method organization results from an instantiation of the meta-model or the generic model. Here, we develop method organization using the generic model of methods [Pra06].We shall provide only a broad view of the generic model to establish a basis for using it for method organization. For full details of the generic model please refer to [Pra06].

According to the generic view, a method is a triple <M, D, E> where M is the set of method blocks, D is the set of dependencies between method blocks, and E is the enactment mechanism. The notion of a dependency is used to build a dependency graph with nodes as method blocks and edges as dependency types. A dependency graph has START nodes that have no edges entering them and STOP nodes that have no edges leaving them. The enactment algorithm guides method enactment from START nodes through the intermediate nodes to STOP nodes.

Broadly speaking a method block has two parts, a product primitive and process primitive. For example, <entity, attribute, attach> is a method block. It specifies that an attribute can be attached to an entity. Method blocks show dependencies among each other. These are the same as DM, DC, IM, or IC respectively considered above. Using the notion of a dependency, a method can be organized as a dependency graph [Pra06, Pra07].

Now, **assuming that each method block corresponds to a method feature**, it can be seen that the dependency graph provides a full representation of the features of a method as well as their inter-relationships. Thus, given an architecture, **we define method organization based on the generic model as the method blocks comprising the method and the links between these**. We illustrate this for the Receive Application component of the MA of Fig. 1.

Method organization for the Receive Application method component is shown in Fig. 2. The method starts off by proposing the filling of the application form, <application, fill>. The form can be received either by courier, by hand, or by email. This is captured by the nodes, <Courier, Receive>, <Hand, Receive>, and <E-mail, Receive> respectively. The method says that couriering and emailing is to be done immediately after filling the form (IC) whereas delivery by hand can be done after some time gap (DC). Irrespective of the path taken, the next feature of the method is to check the application, <application, check>. This **must** be done but its enactment can be **deferred** giving rise to a DM link. Finally the method proposes to issue an acknowledgement, <acknowledgement, receive>. This **must** be done and it must be done **immediately** after the check is completed. This explains the IM link.

The Receive Application method terminates when <acknowledgement, issue> has been enacted. Now, the next method, Conduct Entrance Test, in the MA of Fig. 1 is to be enacted as dictated by the link between Receive Application and Conduct Entrance Test. It can be seen that the method features of which the MA is an abstraction can be organized with the feature as the node and the link as the edge between nodes.

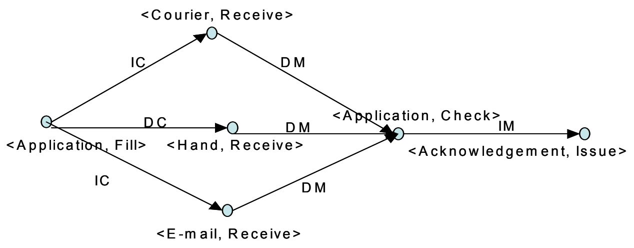

Fig. 2: Organization of Receive Application for Admit Student

5 The Design Phase of MDLC for SME

In this section, we show the basic process by which MA of a method can be used in the Design phase of the MDLC for performing situational method engineering.

Our architecture matching process can be characterized as a left to right, top down, depth first matching process. It starts from the top left most component of an architecture, and asks the method engineer to examine it to see if it meets project needs. If it does, then the nest level of nested components are examined for their relevance to the method To-Be. This is done recursively till all components have been matched. Thereafter, the process asks the method engineer to move to the component on the right of the top most component. The matching process is again performed. The process continues till all components at the top most level are matched. At this point, the method engineer can move to the left most component below the top most and exhaust the components at this level. In this manner when the bottom most component has been matched, the process of architecture matching ends.

To, illustrate, we shall use the architecture of Fig. 1 and follow the architecture matching process to build a method for registering a delegate in a conference. Following our process, we start matching Process Regular Student. We find that this has relevance because we have to Process 'Regular' Delegates. All delegates are regular in the sense that shall all physically attend the conference. We adopt this in the architecture of our method as shown in Fig. 3. Now, the nested components are matched. Following our process, we find Publish Advertisement in Fig. 1. We need this method as it is, so we nest Publish Advertisement in Process Regular Delegate of Fig. 3. Then we match Receive Application, find that it is needed, adopt the link property DM and arrive at the left side of Fig. 3. Conduct Entrance Test of Fig. 1 is not required in the Process Regular Delegate.

Now, we move to second row of the architecture of Admit Student. Process Distance Learning Student is not required in Register Conference Delegate because all delegates are to physically attend the conference. So we remove it. Next we move to the third row of Fig. 1. Declare Result is not required for Register Conference Delegate, so we remove it. Collect Fee is required and is included as such in Fig. 3. We find the link DM between Process Regular Delegate and Collect Fee. Then we move to the next method component in Fig. 1 Register. It is required for Register Conference Delegate with the link IM.

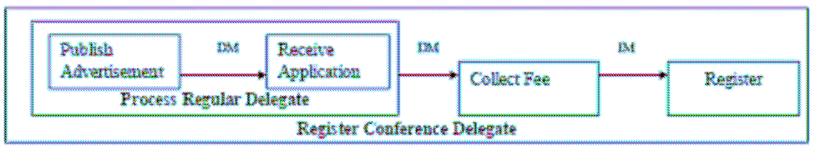

Fig. 3: Derived Architecture of Register Conference Delegate

It can be seen that the matching process at the architectural level leads to the examination of legacy method components and links between them for their appropriateness in the MA To-Be. Using knowledge of what is to be achieved, the new architecture can be developed.

6 The Construction Phase of MDLC for SME

Now, let us consider the construction of the method To-Be. For this purpose each component of the MA must be expressed as a dependency graph. As discussed earlier, this graph constitutes method organization.

The organization matching process lies at the heart of the Construction phase of the Life Cycle shown in Table 1. This matching process is left to right and top down. This means that the method engineer shall consider each node in the graph starting from the left most. If a node branches out into many nodes, then the top most link is first examined for matching. Then the next lower edge is examined and so on till all the outgoing branches have been considered. Thereafter, the method engineer moves to the next node on the right and the process continues.

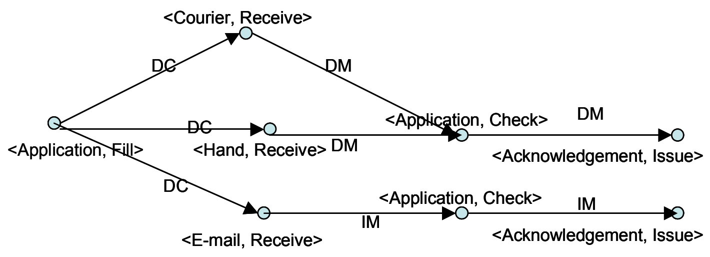

Fig. 4: Organization of Receive Application for Register Conference Delegate

We illustrate this process with the Receive Application organization of Fig. 2. Starting from the left most node, we find that <Application, Fill> is relevant to the new method. So, it is included in the new organization. We notice a number of branches coming out of this node in Fig. 2. Starting from the top most, we find that it is relevant but the link is changed to DC since there can be a delay between filling and couriering the application. The edge from <Courier, Receive> to <Application, Check> does not change. This is shown in Fig. 4. The middle edge to <Hand, Receive> of Fig. 2 and its outgoing edge are accepted as such. Similarly, the bottom edge is accepted as such. This is shown by the middle and bottom edges of Fig. 4. Just as in Fig. 2, it is necessary to issue an acknowledgement for the new method. However, in the new method, acknowledgement of applications received by e-mail is done immediately after checking is over. For the other ways in which applications are received, the acknowledgement is to be issued in deferred mode, i.e., perhaps, after a delay. This forces us to reproduce the <Application, Check> node and create different links to <Acknowledgement, Issue> as shown in Fig. 4. The link to the next method component is now to be examined and organization matching for the Collect Fee component is done. As for architecture matching, organization matching enables the systematic examination of each method feature and an evaluation of its fitness for use in the method To-Be. Every link between features is also examined to find its suitability.

7 Conclusion

The area of method engineering lays great emphasis on meta models. Meta models provides an abstraction of the set of concepts that go into defining a method. Thus, GOPRR can be used to instantiate a large range of methods. MA is different from a meta model. First it is developed for a given method, Register Delegate, Reserve Room, Issue Passport etc. and does not have the capability of meta-models to yield different methods. MA elaborates method components and links between them to reveal method functionality. Second, when instantiated, a MA yields a method organization, the manner in which method features 'hang' together. A given MA can be instantiated to yield more than one method organization. Thus, it can be seen that whereas a meta model can yield **different kinds** of methods, a MA yields different organizations for a **given** method.

A method organization is represented as a dependency graph. This graph reveals relationships among method features and is controlled by the urgency and necessity properties. Diagrammatically, the dependency graph looks similar to a map used in [Ral03]. However, the map is at a completely different level of abstraction. Its nodes are intentions and edges are strategies for fulfilling these intentions. This is in contrast to a method organization where nodes represent the capability of operating on given product elements. Again, in a map, there is a successor-predecessor relationship between intentions. Relationships are controlled by different map topologies, bundles, multi-paths, etc. That is, a map is an expression of a method at an intentional level but is not a dependency graph whereas a method organization is an expression at the operational level and is a dependency graph.

In this paper we proposed a life cycle for the task of method engineering and have concentrated on elaborating the construction and design phases of this life cycle. The output of the design phase is the architecture of a method. The meaning of MA and its representation was developed by analogy with the notion of architecture found in computer technology. Similarly, the notion of method organization was developed for the construction phase of the life cycle. In order to develop the method To-Be, we proposed matching processes: at the design phase, for architecture matching and at the construction phase, for organization matching.

It can be seen that method organization provides a way to realize a MA. Each feature of the method represents a low level functionality and an abstraction of a collection of such features results in a component of an architecture. Whereas MA is about the arrangement of the method, the organization is about the working of the method.

References

[Bas98] Bass L., P. Clements, R. Kazman (1998). *Software Architecture in Practice.* Reading, MA: Addison Wesley Longman, Inc.

[Bri98] Brinkkemper S., Saeki M., Harmsen F., Assembly Techniques for Method Engineering, Proc. CAiSE 98, Pernici B. hanos C. (eds.) LNCS 1413, Springer, 381-400

[Buc62] Planning a Computer System, W. Buchholz ed., McGraw-Hill, 1962, p. 5

[Cla05] Clark D. David, http://find.isi.edu/presentation_files/Dave_Clark-What_is_architecture_4.pdf 2005

[Gro97] Grosz G., et al, Modelling and Engineering the Requirements Engineering Process: An Overview of the NATURE Approach, Requirements Engineering Journal, 2, 3, 115-131

[Gru96] Grundy J.C. and Venable J.R., Towards an Integrated Environment for Method Engineering, in Method Engineering Principles of Method Construction and Tool Support, Brinkkemper, Lyytinen, and Welke (eds.) Chapman and Hall, 45-62

[Gup01] Gupta D. and Prakash N., Engineering Methods form Method Requirements Specification, Requirements Engineering Journal, 6, 3, 135-160

[Har94] Harmsen F., et al, Situational Method Engineering for Information System Project Approaches, in Methods and Associated Tools for the Information Systems Life Cycle, Verrijn-Stuart and Olle (eds.), Elsevier, 169-194

[Hay98] Hayes P. John, Computer Architecture and Organization, McGraw Hill International Editions, 3/e, 1998, P.34

[Kru94] Kruchten 1994, Software Engineering Institute, Carnegie Mellon, http://www.sei.cmu.edu/architecture/published_definitions.html

[Lane 90] Lane 1990, Software Engineering Institute, Carnegie Mellon, http://www.sei.cmu.edu/architecture/published_definitions.html

[Man03] Computer System Architecture, M.M. Mano, P-H, 2003, P. 3

[Pau02] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, Judith Stafford, Documenting Software Architectures: Views and Beyond, by Addison Wesley Professional. 2002

[Pra97] Prakash N., Towards a Formal Definition of Methods, Requirements Engineering Journal, Springer, 2, 1, 23-50

[Pra99] Prakash N., (1999) On Method Statics and Dynamics, Information Systems Journal, 24, 8, 613-637.

[Pra02] Prakash N. and Bhatia MPS**,** Generic Models for Engineering Methods for Diverse Domains, , Advanced Information Systems Engineering. Pidduck A.B., Mylopoulos J., Woo C.C., Ozsu M.T. (eds.) CaiSE 2002, LNCS 2348, 612-625, 2002

[Pra06] Prakash Naveen, On Generic Method Models, Requirements Engineering Journal, 11, 4, 221-237, 2006

[Pra07] Prakash Naveen, An Intention Driven Method Engineering Approach, First International Conference on RCIS, Morocco, April (to be presented)

[Ral03] Ralyté J; Deneckère R., Rolland C., Towards a Generic Model for Situational Method Engineering, Proc. CAiSE 2003, Eder J. & Missikoff M. (eds.) LNCS 2681, Springer, 95-110

[Sha96] Shaw M. and D. Garlan. Software Architecture Per-spectives on an Emerging Discipline. Prentice Hall, New Jersey, 1996

[Sta96] W. Stallings: Computer Organization and Architecture, 4. ed. 1996, p-3

[Sub99]Subbu Allamaraju, http://www.subbu.org/articles/architecture/Paradox.html