Chapter 1

Methods and Method Engineering

In this chapter, the different definitions of Methods and Method engineering will be presented. This is followed by a brief discussion on the process of method engineering. The chapter further explores the previous proposals on method engineering, the concept of system configuration and the reason for extending configuration in method engineering domain. Finally, the proposals on method configuration will be discussed; using this discussion the chapter turn up at the problem of the thesis and further discusses the solution approach.

1.1 Importance of Method

Use of Methods in Information System Development (ISD) is widespread as it provides standardized and control way of developing the good quality product. This is achieved through two significant features:-

- Way of working - Best path or route to construct a new product
- Guidance - The choice of executing a new step.

A method may accompany a Computer-Aided Software Engineering (CASE) tool that implements the discipline imposed by it. Thus, increases the productivity of development.

Some of the definitions of Method Engineering in literature are as following:-

(Brinkkemper, 1996): described a method as "*an approach to perform a systems development project, based on a specific way of thinking, consisting of directions and rules, structured in a systematic way in development activities with similar development products*".

(Prakash, 94): proposed a method as *"a collection of tools and techniques, product and process models, guidelines, checklists, heuristics, etc. that help an application engineer to build a suitable product"*.

(Iacovelli et. al., 2008): describes *"A method is based on models (systems of concepts) and consists of some task/activities/steps, which should be performed, in particular, order"*.

(Smolander et. al., 1990): A method is an *"a predefined and organized collection of techniques and a set of rules that state by whom, in what order, and in what way the techniques are used to achieve or maintain some objectives."*

There are two aspects of a method - Product aspect and Process aspect.

1. **Product Aspect-** The product aspect provides features for product development and ensure product standard. Product model defines a system of concepts and their inter-relationships including constraints. Examples of product model are – *ER diagram, OOA, OMT,* etc. The product aspect provides:

   A. <u>Functional features</u>- Functional features identify the set of building blocks and rules to combine them so that complex concepts can be built from simpler concepts. Final product structure can be created as an appropriate combination of simple and complex concepts.

   B. <u>Non-functional features</u>- Non-functional features are the quality constraints some of which are mandatory and others are desirable.

      (Prakash, 97) has classified mandatory constraints as:

      - *Consistency Constraint*: If something holds then opposition does not hold.
      - *Completeness Constraint:* All the components necessary for the concept to be well structured are defined and put together. For example, an *entity* in ER diagram is

complete, if it has, at least, *one attribute* and *one primary key* associated with it (Chen, 1976).

- *Conformity Constraint:* Use of the concept is in conformity with the product model. For example, this ensures that only one *primary key* is *attached* to the concept *entity*.
- *Fidelity Constraint:* System to modeled is represented in the product faithfully. For example, *entity* participates in at least one *relationship*.

The quality checks and quality criteria are desirable and are defined as-

- *Heuristics:* Heuristics is experience-based rules that ensure product structure to be comfortably understandable, ex. In (Coad and Yourdon, 91) not more than five processes in DFD, Maximize fan-in of the module in the design phase.
- *Design Factors:* These are the product qualities features that method assumes for its product. For example in, (Coad and Yourdon, 91) design factors are cohesion and coupling.

2. **Process Aspect** – The process aspect is the route that needs to be followed to ensure the efficiency of product development. For example, In (Coad and Yourdon, 91) *DFD must be completed before construction of design begin*. (Dowson, 98) has classified Process Models as:

A. Activity oriented models - These models ignore the relationship between the activity and product produced for example- Water Fall model.

B. Product/Activity oriented Models- These models view product development as successive transformations performed on the product by the activities. As a result relationship between product and activity is clearly articulated. For example, View Point Oriented Model (Sommerville, 95).

C. Decision-Oriented Models – These models comprises of development decisions that cause product transformation. They are not pre-ordered but taken with a particular situation at hand. Choices of the situation to be handled are dynamically decided by application engineer. There is a close relationship between situation and decision that can take on the situation. For example, Decisional Meta Model (Prakash, 99).

1.2 Method Engineering

It has now been proved that no universal method can apply to all projects since different projects have different characteristics (Brooks, 87; Avison, 96; Kumar, 92; Glass 00; Glass04). To complete the project with perfection in the given time line, one should use the most suitable method according to the particular project characteristics also known as situations. The field of Method Engineering (ME) has evolved in response to this requirement. The widely accepted definitions on method engineering are:-

(Brinkkemper, 1996): Defines ME as *"Engineering discipline to design, construct and adapt methods, technique, and tools for the development of information systems."*

(Henderson-sellers et. al., 2005): Sees it as a process to combine *"separate fragments of methods, which are not interdependent or even intertwined to create a method."*

(Engels et. al., 2010): Defines ME as *"Providing a framework for defining and tailoring Information System Development and software engineering methods".*

(Raylte et. al., 2008): "*Emerged as the research and application area for using methods for information and software systems development*".

(Tuunanen et. al., 2004): States ME as *"Methods and processes to specify, make explicit, codify, and communicate method knowledge as well as technical tools to enact such processes effectively".*

The base of Method Engineering is the **Underlying Meta Model** other important factors are **Method component, Method base, and Project characteristics** (Prakash, 97). Also, Method Engineering is supported by software tool called Computer Aided Method Engineering.

1.2.1 Meta Model

A Metamodel is a set of ‘generic concepts’ and relationship between them. Metamodel defines the common principles underlying the design of the method. It can be used to compare and evaluate methods. Metamodels can be divided into three broad categories:-

1. Data Metamodel: Data Meta Models are the product Metamodels and can model only the product aspect of a method. For example, OPRR Metamodel (Smolander, 1991).
2. Activity Meta Model: Activity metamodel augments the product Metamodel with the task-oriented approach of process Metamodel. In this, the product models are instances of Data Metamodels and process aspects are instances of Activity Metamodel. For example, Fragment Metamodel (Harmsen, 97).
3. Integrated Product-Process Meta Model: These models invested the importance of process models deeply and conclude to couple process and product aspects of the models. The coupling removes the product-process dichotomy. Examples are- Contextual Metamodel (Rolland et al., 95) and Decisional Metamodel (Prakash, 97: Prakash, 99).

1.2.2 Project Characteristics

The requirements of the method are determined by Project characteristics (Harmsen and Brinkkemper, 93; Harmsen et al., 94; Rolland and Prakash, 96b). (Slooten and Hodes, 96) has proposed to elicit specific need of the method as project *contingency factors* and constraints on these factors. They have identified sixteen contingency factors. Some of the important factors are Management commitment, Time pressure, Skill, Formality, Knowledge, and experience, etc. Project-specific methods are made by retrieving methods from method base as per the contingency factors or project characteristics.

1.2.3 Method component

Method components are ‘partial methods’ - that are reusable in generating new method. They are defined in compliance with the underlying metamodel, and may be described as *fragments* (Harmsen et al., 94), *contexts* (Grosz G. et al., 97: Rolland et al., 98: Raylte and Rolland, 01: Kornyshova et al., 2007), *decisions* (Prakash N., 99), *patterns* (Plihon and Rolland, 95) *etc.*

1.2.4 Method Base

Method base is a repository of method components of existing methods, these components are accessed based on the project characteristics. The method base is populated every time a new project-specific method is generated.

The process for retrieving method components from method base may briefly describe as- First, project situations are expressed regarding project schema or contingency factors. These are then used to select appropriate method component from method base, selected method component are further used to define a new method in accordance with the metamodel. The method base is populated with new method components.

1.2.5 CAME tools

Method Engineer is a role responsible for generating the project specific method, for this he needs to be empowered by software engineering tool referred as *Computer Aided Method Engineering* (CAME) tool. Researchers have developed different CAME tools, for example, Decamerone (Harmsen et al., 94; Harmsen et al., 95), MetaEdit (MetaCase, 95), Mentor

(Plihon, 96; Si-Said et al., 96), MERU (Gupta and Prakash, 01). The tools provide user-friendly interfaces for selection of method fragments, assembly and administration of new component in the method base based on the method engineering approach used.

1.3 Different forms of Method Engineering

The important forms of Method Engineering are:-

- Method Assembly
- Method Generation
- Method Modification

Method Assembly- Assembly-based approaches for method engineering rely on a method base (Ralyté and Rolland, 2001). From this method base, method components are retrieved as per the project characteristics. The retrieved components are then assembled to form project-specific method. The retrieval and assembling operations are performed in accordance with the Metamodel.

Method Generation – Method generation generates a new method from scratch. Project situations are used to instantiate the underlying Metamodel concepts and to generate the specification of method.

To avoid the tedious task of instantiation recent approaches store generic pattern or rules in the method base. They do not require complete knowledge of Metamodel for generating new method. Based on Project characteristics, generic patterns are selected that automate the generation of the new method.

Method Modification – Method modification modifies an existing method. During this approach, method component is retrieved from the method base and new method is formed by:-

- Changing the concepts of existing method.

- Adding a new method concept.
- Deleting a method concept.

Method modification process is also called termed as tailoring and extending a method.

1.4 Method Engineering Approaches

This section will present a literature review of Method Engineering proposals. The purpose is to gather the efforts of the several method engineers, summarized them and conferred them to show the overall growth of this vital discipline.

Early approaches for method engineering were centred on method assembly and method generation. Later method engineering was done using Architecture-centric approaches. These proposals are analogous to software engineering domain and are two stages – first, the architecture of situated method is formed, and then method is organized from this architecture.

Very recently, the Method Engineering has moved to Method Configuration to construct a project-specific method. They rely on base method/method components which can be transformed into a situation specific method through the process of tailoring, extension or assembly.

The section starts with the proposals on method assembly and method generation then it moves towards the proposals on Architectural centric method engineering approaches that provide a rich set of guidelines. The section further analyses the proposals that perform method engineering through Method configuration and concludes with the industrial case studies showing the relevance of the method configuration in the functional domain.

1.4.1 Method Assembly approaches

Method Assembly process ensures that methods formed are in coherence with the underlying metamodel. Two widely accepted approaches for method assembly are:-

1. Fragment-Based Approach.
2. GOPRR Based Approach.

1. **Fragment-based approach -** Fragment-based approach for method assembly is proposed by (Harmsen et al., 94; Harmsen et al., 95; Harmsen, 97). The meta-concepts of this approach are modelled using Fragment Meta Model. (See figure 1.1).

   Fragment Meta Model

   Fragment Metamodel describes the method as a collection of product and the process fragments and various relationships such as *precede, part of, require, supports*. Product fragments represent products and sub products like deliverable documents, models, and diagrams, etc. Process fragments can be stages, activities, and tasks to be carried out. These fragments are further classified as a conceptual fragment and technical fragment.

   Conceptual fragments represent Information System Domain methods or parts of these, whereas the technical fragments are tool details for operational part of the method.

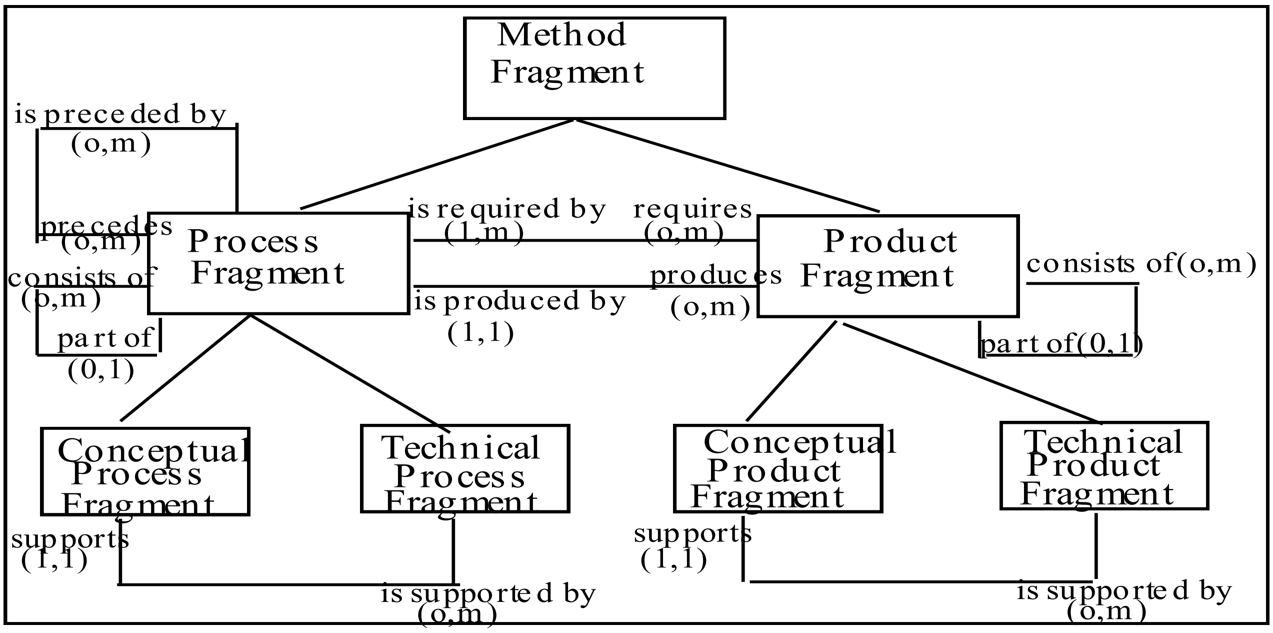


Figure 1.1: - Fragment Meta Model (Harmsen, 97)

Method Base: - The method base is structured into three parts - *Method Repository* that consists of method fragments of already existing methodology. *Selected Method Fragments Repository (SMFR)* that stores selected method fragments for assembly and *Situational Method Repository* that stores the assembled situational method.

Project Characteristics: The project characteristics or project requirements explicitly expressed as contingency factors. Harmsen identifies contingency factors as *organization culture, existing information infrastructure, application characteristics, external factors, technical factors and development expertise,* etc. An example of contingency factors is IS Adaptation, Incorporation of standard software, Database conversion, Average response time, Low complexity, the Average level of experience needed.

The Process - In this approach new methods are constructed as follows:

- Project requirements are elicited as contingency factors.

- Suitable method fragments are retrieved and selected from the method base depending on contingency factors.
- The retrieved method fragments are integrated to form the coherent method represented in terms of Fragment Metamodel. There are about one dozen rules that check the consistency of integration.

Tool Support - Based on the approach Harmsen also design a CAME tool named Decamerone. Its front end provides following interfaces:

- Determination and evaluation of Contingency factor.
- Selection and assembly of method fragment.
- Adaptation of specific method.
- Administration of new method fragments in method base.

2. GOPRR Based Approach

The other assembly based proposal is from (Kelly et al. 96). It uses GOPRR metamodel (see figure 1.2) that is an extension of OPRR Metamodel (Smolander, 1991). OPRR meta-model has four Meta concepts Object, Property, Role and Relationship where an Object is a thing, which exists on its own. The relationship is an association between two or more objects. A role specifies the link between Object and Relationship. The role is *involved* with Relationship and specifies the role played by an object in a Relationship. The property is a *describing* or qualifying characteristic associated with Object, Relationship or Role.

GOPRR Metamodel

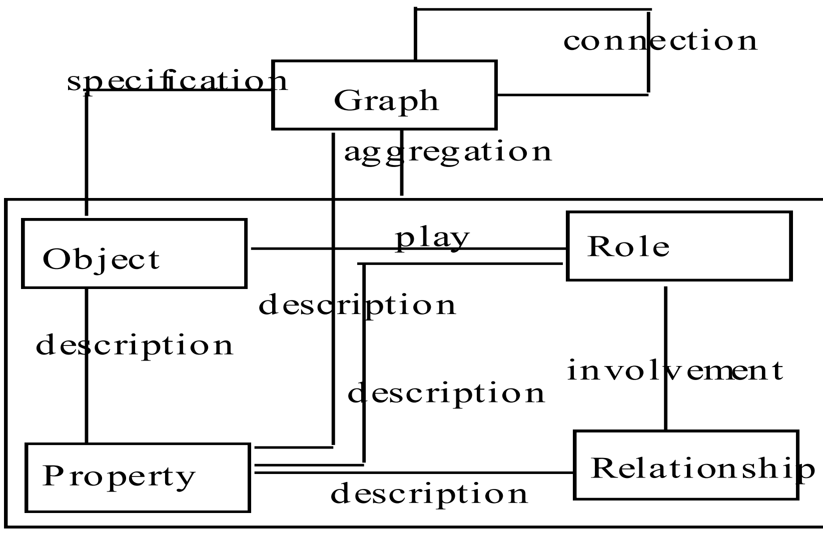


Figure 1.2: - GOPRR Meta Model (Kelly et al. 96).

Besides all the previous concepts of OPRR, GOPRR has a new concept of Graph. A Graph is an aggregate concept for collecting primitive types (Object, Role, and Relationship). A Graph(s) can connect to another Graph.

- A Relationship can specify between Object and Graph
- Properties can be described as a Graph.

Graphs are used to support the construction of a new method by collecting reusable elementary Graph types and expressing them as an aggregation to form a new Graph. In this approach, the new method can be defined either from scratch or by reusing the already formalized methods/method components.

The method base of GOPRR approach stores method specifications represented as GOPRR concepts in the *Objects Specification Base (OSB)* and symbols needed to represent Objects, Relationship and roles in *Symbol Specification Base* part. The information necessary to represent objects in tools stored in *Tool complementary information base*. *User information base* contains user related information. *Report specification Base* contains all reports and other output specifications.

Project Characteristics: GOPRR approach gathers project characteristics in the form of contingency factors defined by the Fragment-based approach.

The Process: The Process is as follows:-

- Method Components are residing in the OSB as GOPRR concepts.
- The retrieval toolbox is used to retrieve Objects and their instances from the Object Specification base by the elicited contingency factors.
- These concepts are then assembled into complete method specification using the Graph.
- The consistency checking system incorporates several rules that ensure the syntactical completeness and consistency of the assembled method.

Tool Support: MetaEdit+ provides an environment that supports multi-user, multiform, multi-tool, and multi-method and multi-level. The various features of the environment and their components are managed by the *Environment Management tools*. The other tools include *Model Editing tools*, *Model Retrieval tools*, *Model Linking and annotation tools, Method management tools*. Our interest is in *Method management tools*.

Method Management Tools **(MMT)**

Method management tools shown in fig.1.3 is a family of tools that supports the construction of methods, their management and reuse. It consists of components as described below.

A. Repository: This consists of three parts Object specification base, Symbol base and Report specification base. Object specification base consists of method fragments, which are method specifications in terms of GOPRR concepts. Symbol specification base consists of graphical symbols needed to represent method concepts and Report specification base consists of all other reports and output specifications.

B. Method Assembly System: This includes following tools:

- *Meta-model Editors*: These are the Object, Property, Role, Relationship, and Graph editors. These editors can be used to create instances of Object, Role Relationship, etc. or reuse existing instances in the method base.
- *Symbol Editor*: It is an editor used to specify symbols for instances of meta-concepts.
- *Process Subsystem*: This consists of process editor and other form-based tools for defining the information system development process (Koskinen, 1996).
- *Consistency checking system*: It checks the syntactical completeness and consistency of specified method and analyses it for contradictory specifications.
- *Metric and Static subsystem*: This provides analysis report for the newly defined method.

C. Environment *Generation* System: This system of tools is responsible for delivering the CASE tool by using the method definitions obtained from *Assembly.* It consists of following sub-systems:-

- *Help generator*: This is used to generate online help.
- *Method support environment generato*r. This produces the method object file.
- *Report code generator*: This is used to generate reports on models.

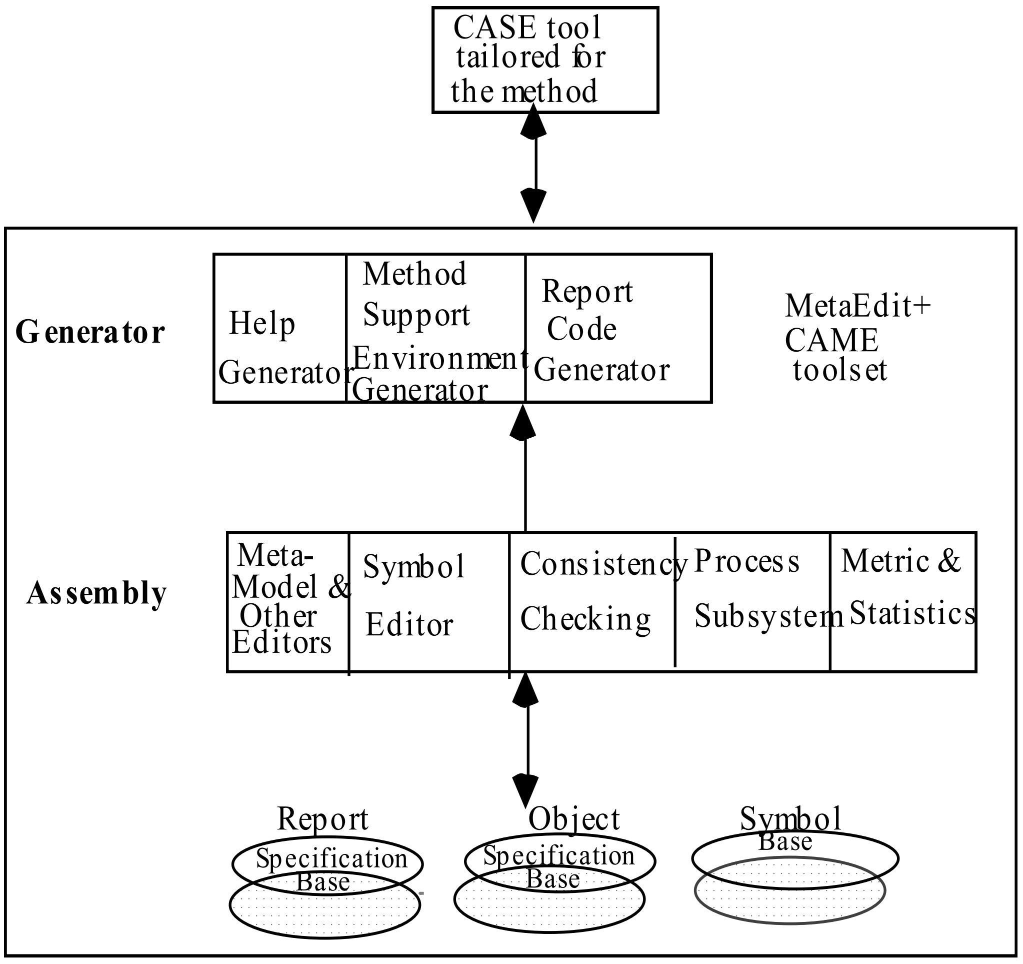


Figure 1.3:- MMT in MetaEdit+ (Kelly et al. 96).

Drawbacks of Assembly-Based Proposals:-

- Method Assembly is a detailed and tedious task as different selected components are to be integrated to form coherent method.
- For assembly, it requires complete knowledge about the Metamodel. To maintain coherency, the method component along with the new concepts needs to make instances of the meta-model used.

1.4.2 Method Generation approaches

Major proposals in this category are:-

1. Contextual Approach for method generation.
2. Method Engineering Using Rules.

1. Contextual Approach for method generation.

Method generation has evolved from the problems caused by method assembly. It first starts with the contextual approach proposed by (Rolland C. et al., 95) and, later on, moves towards the more consistent and generic approach offered to generate methods using rules (Gupta, D. and Prakash, N., 01). The contextual approach supports to generate method from generic pattern stored in method base. The Meta concepts of contextual approach are supported by the underlying metamodel i.e. contextual meta-model (see figure 1.4). **Contextual Meta Model:** In this, a method is represented as a *collection of hierarchies of contexts.* A context is an ordered pair of <situation, decision>.

Where, **Situation** represents the product state, and D**ecision** represents an intention or a Goal, to fulfill for a given situation.

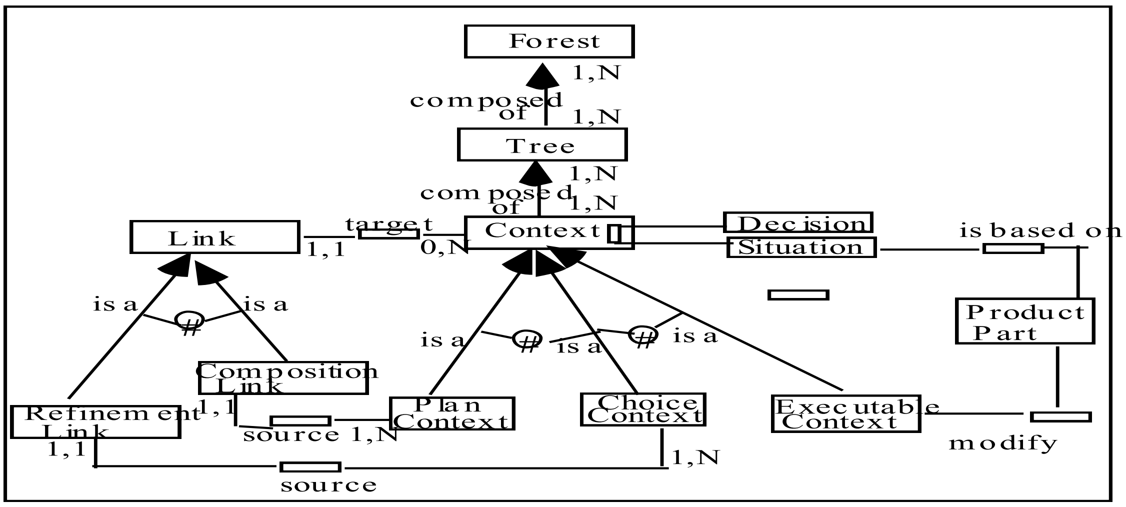


Figure 1.4: - Contextual Meta Model (Rolland et al., 95).

The contexts are classified as - C*hoice,* P*lan,* and E*xecutable* and are related through *refinement* and *decomposition* links to build *trees*. A node of a tree is a context, and the edges of the tree are refinement or decomposition links. The trees can organize into *forests*.

Method Base: Contextual approach stores generic patterns in method base (Rolland et al., 96a) - these generic patterns are represented as contexts to build the desired method from scratch. The situation part of generic patterns represents the product whereas decision part represents the process goal.

There are four classes of generic patterns *describe, construct, refine and check*. For example the *describe patterns* when instantiated for "class" of the class diagram, would require that "data-type" and "operation" composing "class," should be portrayed regarding product metamodel. Thus, instantiation of generic patterns requires the instantiation of the situation part of the context.

Method Base is organised at two levels:-

- *Method knowledge level* – It represents method chunks at different levels of granularity and different levels of abstraction.
- *Method meta-knowledge level* - It captures the knowledge associated with a method chunk, in the method base. This helps in determining the context of its use.

Project Characteristics: The generic patterns stored in the method base are retrieved from using *Descriptors (*Rolland and Prakash, 96b). A descriptor is a meta-context that describes a method chunk that is relevant to an individual situation to achieve an assertive intention. The descriptor has two main classes- Area of the project and Risk and Complexity of Problem Domain. Problem Domain is further classified into two parts, Target Domain, and Project Domain. Each of these parts is further characterised by factors Task, Structure, Actors, and Technology. Complexity is measured as simple, moderate or high. Similarly Risk is measured as low, moderate and high.

The Process: The steps of method creation are:

- Identify project characteristics in the context of descriptors.
- Retrieve from method base corresponding generic patterns and choose the most suitable ones.
- Instantiate the situation part of the generic pattern.

Tool Support: Mentor (Plihon, 96; Si-Said S. et al., 96) is the CAME tool support for the proposed approach. Its primary components are:-

- *Editors*: There are two types of writers: Product Editor and Process Editors. Product Editor provides graphical features to specify a product. Whereas Process editor consists of services to specify a method in the contextual form.
- *Method Generator*: It automates the construction of a method using the generic patterns. Once the product has been specified and the appropriate pattern selected, it automatically selects the product parts and produces the hierarchy of contexts rooted in the pattern.
- *Browser*: It is used to scan the product parts and method chunks (components) stored in the method base. It has two sub-components: Product browser and Process browser.

2. Method Engineering Using Rules

The other approach for method generation is given by (Gupta and Prakash, 01). They proposed to generate the new method using a set of generic rules. The proposal is based on

Decisional Metamodel (DM) (Prakash, 97) (See figure 1.5) and uses Method View Metamodel (MVM) (Gupta and Prakash, 01) (see figure 1.6).

Decisional Meta Model

Decisional Metamodel categories method as: transformational and constructional. A transformational method is used for transform a product into another product. In contrast, a constructional method is used whenever a new product, is to be constructed. A method can be atomic or compound. Atomic methods are those that are expressed in exactly one product model whereas compound method composes of other simpler method. A constructional method that builds products for the ER model is atomic since the product is expressed in exactly one model. Similarly, the transformational method for converting an ER product into a relational product is atomic since each of the products is represented in exactly one product model.

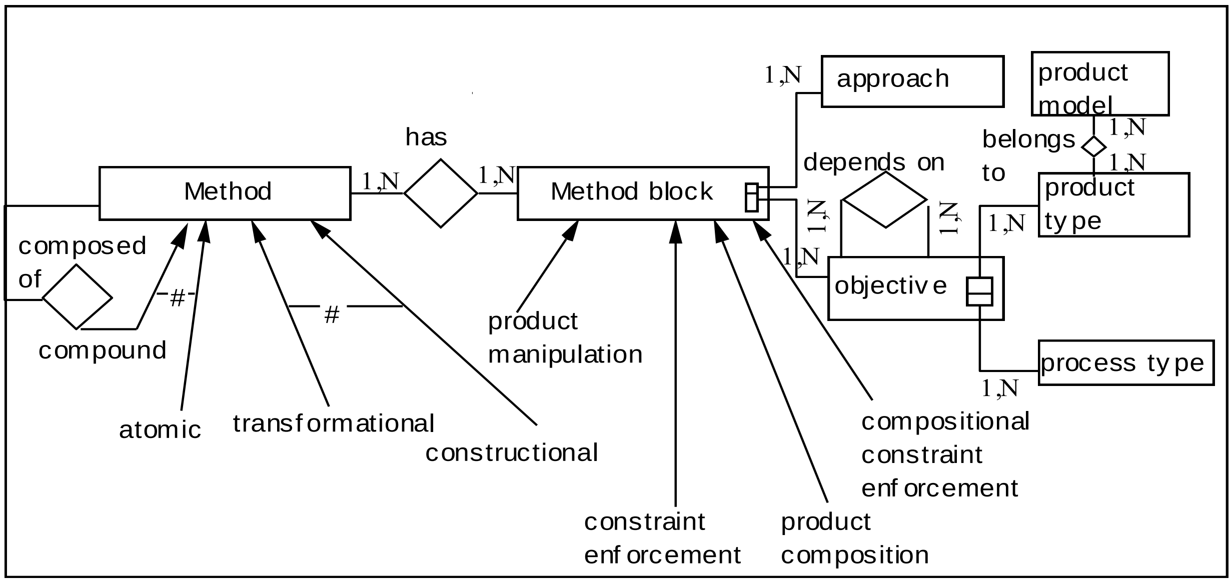


Figure 1.5:- Decisional Metamodel (Prakash, 97).

A method as a set of decisions and decision is a pair of <purpose-approach>. They concentrated on the purpose part of the method and ignored the P-approach. So, their view of the method is in the form of a triplet <P, Dep, Ed>, and where P is the set of purposes of a method, Dep of dependencies between purposes, and Ed is the enactment mechanism expressed regarding the instantiation.

Method <Purpose, Dependencies, Enactment Algorithm>.

Method View Meta Model: The MVM is an abstract Metamodel at a level higher than Decisional Metamodel and is not technical Metamodel like OPRR, GOPRR and Fragment Metamodel. It is used for Method Requirements Specifications (MRS), the MRS in terms of MVM are used to instantiate decisional metamodel using a set of rules.

The Meta view model contains two concepts: '*thing*' that specifies the concepts in a product model and '*is related to*' specifies the relationship between things.

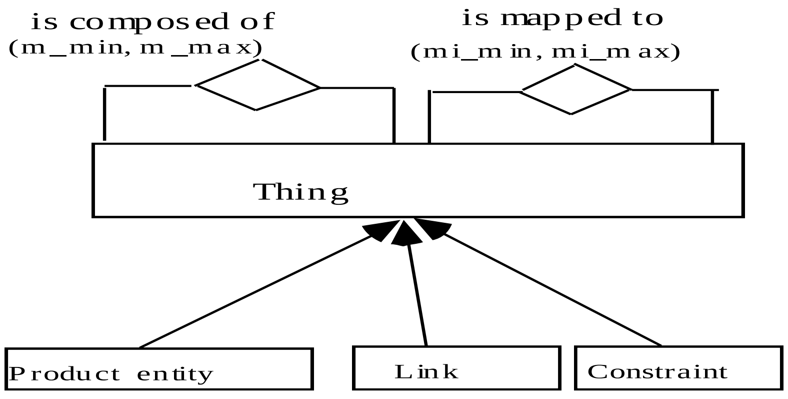


Figure 1.6:- Method View Model (Gupta and Prakash, 2001)

Further MVM Metamodel partitions the things into product entities, link, and constraints. A link is anything of the product that connects two product entities together. Examples of links are aggregation links and specialization links. Constraints are those things that can be used by application engineers to specify properties of links and product entities. Finally, anything that is not a link or a constraint is a product entity. The relationship *is related to* partition into two namely, *is composed of* and is mapped to respectively. The former says that things of a product model built out of simpler ones. For example, an entity type of the ER model is composed of attributes and primary keys. The *is mapped to* relates together things of two different models, for example, a method for transforming the **ER model into the relational model**, the *thing* entity of the former is mapped to the *thing* relation of the latter.

Method Base: - Method Base is partition in two parts the first part contains the set of generic rules for instantiating purposes and dependencies from MRS. The other part contains the Method component expressed in terms of Decisional Metamodel, which are reusable to generate new method by modifying existing method. Method modification means new concepts can be added/deleted/modified in the existing method. These method components are linked to the corresponding MRS.

The Process: There are three steps for method generation in the approach:-

- **Developing MRS**- Based on project characteristics in terms of descriptors (refer contextual approach) produce Method Requirement Specification (MRS) inconsistent with Method View Model (MVM).
- **Developing Method Design**: MRS is then translated into an instantiation of product part of Decisional Meta Model using an instantiation algorithm.
- **Constructing Method**: From above instantiation, purposes and dependencies of the methods are generated using a set of rules. These set of rules generate purposes of different types - Basic life cycle, Relational, Method Constraints and Integration.

Tool Support: The proposal is automated with the help of a **CAME tool named Method Engineering Using Rules (MERU)**. MERU offers following functionalities through different interfaces:

- For method engineers, MERU provides an interface to compose Method Requirement Specification Components inconsistent with Method View Model. Additionally, it provides features to modify existing MRS for method modification.
- For application engineers, it produces a list of purposes and dependencies between them. This defines the functionality available in the method. Also, different method components are generated that can be stored in the Method base.
- For the CASE generator, it produces a complete description of the method and method component.

Drawbacks of Method Generation Approaches:

Method Generation solves the instantiation problem of underlying Metamodel to a great extent. Still method engineers face some difficulties:-

- The contextual approach requires instantiation of situational part of the contextual metamodel. The decisional approach requires instantiation of method in terms of MVM.
- Method Generation and Method assembly proposals lack in providing proper guidance to the Method Engineer.

To facilitate method engineers, there are proposals to provide a rich set of rules and guidelines to form a coherent method. Section 1.4.3 describes the major proposal under this domain.

1.4.3 Architectural centric Method Engineering approaches.

Architectural based proposals are analogous to architectural based software engineering domain proposals. They present the task of method engineering to be performed in a more disciplined and cohesive way.

Major proposals that provide guidance to Method engineers are:-

1. Intension based Method Architecture approach (MIA).
2. Architecture-Centric Method Engineering Approach (ArCME).

1. Intension based method Architecture (MIA)

(Prakash and Goyal, 07: Prakash and Goyal, 08) have proposed a generic method engineering approach that can be used to engineer Information System Domain methods as well as Business Process Models. Analogous to software development approach their process consists of three phases.

The Process

- **Elicit intensions:**-This phase is analogous to interviews approach in software development. Requirements are gathered in the form of intentions (Prakash, et.al 07).
- **Retrieving architecture of intentionally similar method:** - From the architecture pool, method architecture of intentionally similar method is selected and retrieved. Precisely method architecture is the functional abstraction of the class of organization, and there can be many organizations for a produced architecture.
- **Method organizations**-: Finally method organization is obtained by organizing method features represented in method architecture. The method organization is defined as dependency graph with method blocks as nodes and dependencies between method blocks as edges.

Method architecture can be *Atomic* or *Complex. Atomic method architecture execute*s a function that cannot be split to its components. However, *complex* method *architecture* abstracts out an operation that is constituted of some other simpler method *architectures.*

Architectures are related to one shown by *is related to* relationship. This relationship institutes a Successor predecessor relationship between the complex architectures; link type is an attribute of this relationship and takes on a value from the set {IM, IC, DM, DC}. Where,

IM - Immediate-Must Mode, IC - Immediate-Can Mode, DM - Deferred-Must Mode and DC - Deferred-Can Mode.

Their method architecture meta-model has two main properties, *genericity and modularity*. Genericity of method architecture lies in the collection of methods having a common functionality, for example, architecture Admit Students used for admitting students is a complex method architecture built from four methods. Admit National Applicant, Admit International Applicant, Collect Fee and Register Student respectively. The method, Collect Fee, is dependent on enactment of Admit National Applicant and Admit International Applicant. Once either of these is enacted, Calculate Fee is enacted in a Deferred-Can (DC) mode. Whereas, register students must be done immediately after collect fees having an Immediate-Must dependency with collect fees details can be taken from (Prakash and Goyal, 08).

Modularity produces architecture components desirable for reuse; any method architecture can take part as an architecture component in multiple architectures and itself have zero or more architecture components.

The research focuses on the design engineering part, and deals with the retrieval of architecture from the repository, a set of operations are then defined that can be performed on the retrieved architecture. Few of them are- Rename architecture, Nest architecture into another, and Design a sequence of architectures.

The construction process for MIA based technique is assembly based, and its ongoing selection strategy makes it best for usage. An organization can select an architecture and can then reselect from a short-list of selected architectures (as an iterative process) until the most appropriate architecture get selected. The MIA approach is used to represent ISDM as well as BPMs and results that MIA is a generic approach to method engineering.

2. Architecture-Centric Method Engineering Approach (ArCME).

The second major proposal is by (Ahmadi et al.,08; Moaven et.al.08), they proposed an Architecture-centric Method Engineering approach (ArCME). The Architecture-Centric Method Engineering Approach for Assembly based Method Engineering aims at performing ME processes in a more disciplined and cohesive way. “ArCME can be defined as the action of performing ME processes and resting its components on an architecture framework”. Fig. 1.7 explains the process as follows

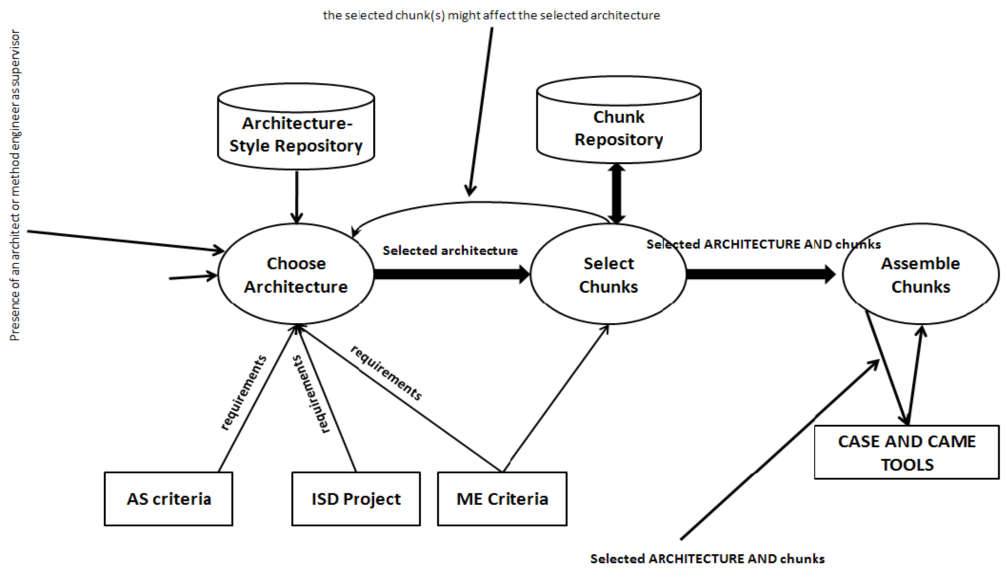

the selected chunk(s) might affect the selected architecture
Presence of an architect or method engineer as supervisor
Architecture-Style Repository
Chunk Repository
Choose Architecture
Selected architecture
Select Chunks
Selected ARCHITECTURE AND chunks
Assemble Chunks
requirements
requirements
requirements
CASE AND CAME TOOLS
AS criteria
ISD Project
ME Criteria
Selected ARCHITECTURE AND chunks

Figure 1.7:- Architecture-Centric Method Engineering Approach (Moaven et.al.08).

The Process

- **Elicit method requirements as situational needs**: - In ArCME, method requirements are gathered in the form of situational needs. The specific needs depend upon information system development projects, method engineering principles, etc.
- **Retrieve method architecture: -** Analogous to the architectural phase of software development, ArCME adds an initial phase in the method engineering task results in some benefits like cost saving, easy training, simplicity in usage, etc. ArCME is centred on an architecture-style repository from where architecture is to be selected. Based on the selected architecture whole SME process is done under the supervision of a method engineer or an architect.
- **Retrieve method chunks:** - The next step is the selection of decomposed components. Method chunk is a decomposed component and is defined by (Ralyté, 04) as "A chunk is a combination of a process fragment (also called as guidelines) plus a product fragment".
- **Assemble method:** - In ArCME, since the architecture is selected at first, all the subsequent steps are performed on the selected architecture. The presence of a structural framework (i.e. architecture) at each step ensures the assembly based SME to be completed in an easier and structural way. Furthermore, the selected architecture and chunks are then used as an input to the CAME tool.

ArCME provides a rich set of guidelines results in a more precise selection of components and then assembling them on the architecture results in significant decrease of

- Refining a selected component
- Adoption time for aggregation strategy
- Decomposing a selected component

- Integration strategy because of satisfying granularity and loose coupling in the selected components.

These approaches still lack in some areas like suitable style selection and further composition of these selected styles results in the evolution of more flexible approaches like OPEN Process Framework (OPF).

OPEN Process Framework approach (OPF)

As the time progressed, software industry moved towards new approaches such as Aspect-Oriented Software Development (AOSD) (Henderson-Sellers et al., 07).The challenging task before method engineers was to identify decomposed components in these new approaches to accomplish their task. The choice may be flexible frameworks like OPF since it has a rich repository of method fragments (or decomposed components) making it suitable for generating situation specific methods (Nguyen and Henderson-Sellers, 03). The OPEN Process Framework as shown in fig. 1.8 consists of:-

- A rich method base consisting of method fragments.These method fragments are defined by < endeavour, language, producer, stage, work product, work unit > and are instances of underlying Metamodel and thus support the basic principle of method engineering.
- Constructional guidelines for fragment retrieval.
- Retrieved method components are then assembled into possibility matrix.

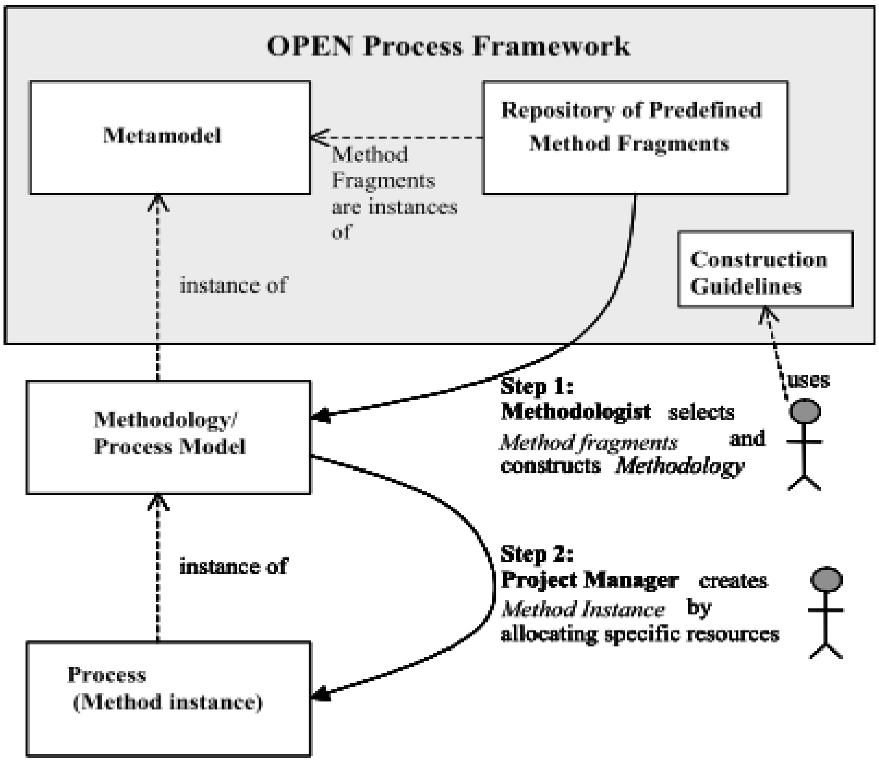

OPEN Process Framework
Metamodel
Repository of Predefined Method Fragments
Method Fragments are instances of
instance of
Construction Guidelines
uses
Methodology/ Process Model
Step 1: Methodologist selects Method fragments and constructs Methodology
instance of
Step 2: Project Manager creates Method Instance by allocating specific resources
Process (Method instance)

Figure 1.8: - OPEN process Framework (Henderson-Sellers et al., 07).

The possibility matrix has seven pairs and is use to map a method fragment with another method fragment.The seven pairs are -Process-Activity, Activity-Task, Task-Technique, Producer-Task, Task-Work product, Producer-Work product, Work product-Language. These pairs are stored in a deontic matrix. To calculate the extent of the relationship between two fragments, a deontic value is calculated for each of the seven pairs. Deontic values can have one of the five values ranging from mandatory through optional and is the responsibility of Method Engineer to allocate these values.

OPEN process framework, because of its flexibility gaining popularity and is now enhanced to support for Component-based development (Haire et al., 01), Organizational transition (Glass, 04), Agent-oriented Development (Debenham and Henderson sellers, 03) and Web-Development (Henderson-Sellers et al, 02).

1.4.4 Method Configuration

(IEEE Std 610.12, 1990) defines configuration as “The arrangement of a computer system or component, defined by the number, nature, and interconnections of its constituent parts”.
The task of configurability is first to create a new model called a configurable model followed by selecting those elements of the configurable model that are relevant to the user’s requirement. Configurable models use notions of commonality and variability.

(Coplien et al., 98) define commonality as an assumption held uniformly across a given set of objects whereas variability is an assumption that is true for only some elements of the set.

(Weiss and Lai, 99) defines the variability as an assumption about “How members of a family may differ from one another: A configurable model identifies commonality and variability that can be exploited in developing a new system from the configurable model.

(Davenport, 98) describes the process of configuration as a methodology performed to allow a business to balance their IT functionality with the requirements of their business.

(Soffer et al., 03) consider configuration as an alignment process of adapting the enterprise system to the needs of the business.

(Moon and Yeom, 05) proposes a method that systematically develops requirements using commonality and variability in product line approaches.

(Karlsson and Ågerfalk, 04) introduced method configuration in method engineering field. According to him, "*Method configuration can be understood as a particular form of Method Engineering that focuses on tailoring and extending of a Standard System Engineering method*". Here, *tailoring* refers as a process that supports minor modifications in a pre-existing method (Basili, 87: Jeffery, 88) and *extending* a method, is adding new concepts in a method to address the need of overall software development process (Fitzgerald et.al, 06).

This section reviews proposals on method configuration. The proposals in the field of Method configuration are:-

1. A Method for Method Configuration (MMC).
2. Method Component for Method Configuration.
3. Method Families for Method configuration.

1. Method for Method Configuration (MMC)

The MMC is proposed by (Karlsson and Ågerfalk, 04). The approach starts off with defining the project requirements in the form of development situations and development characteristics which is followed by determination of *Configuration Package and Configuration template.*

Configuration Packages (CP) - Configure a base method on individual project requirement.

Configuration Template (CT) - Individual project requirement is not sufficient to capture all development situations for a project; therefore Configuration Template (CT) is defined. CT supports multiple project requirements. The authors propose a meta-method named MMC – *Method for Method Configuration* for configuration process.

The process

- Method component used in MMC framework is the base method. Base method is configured by development situations and characteristics use to elicit project development requirements.

- If many development requirements exist, configured method is formed by configuration template. Configuration template is generated by combining configuration packages.
- The configured method is then adapted by the Project Situations to form Project-specific Method.

The MMC framework is shown in figure 1.9:-

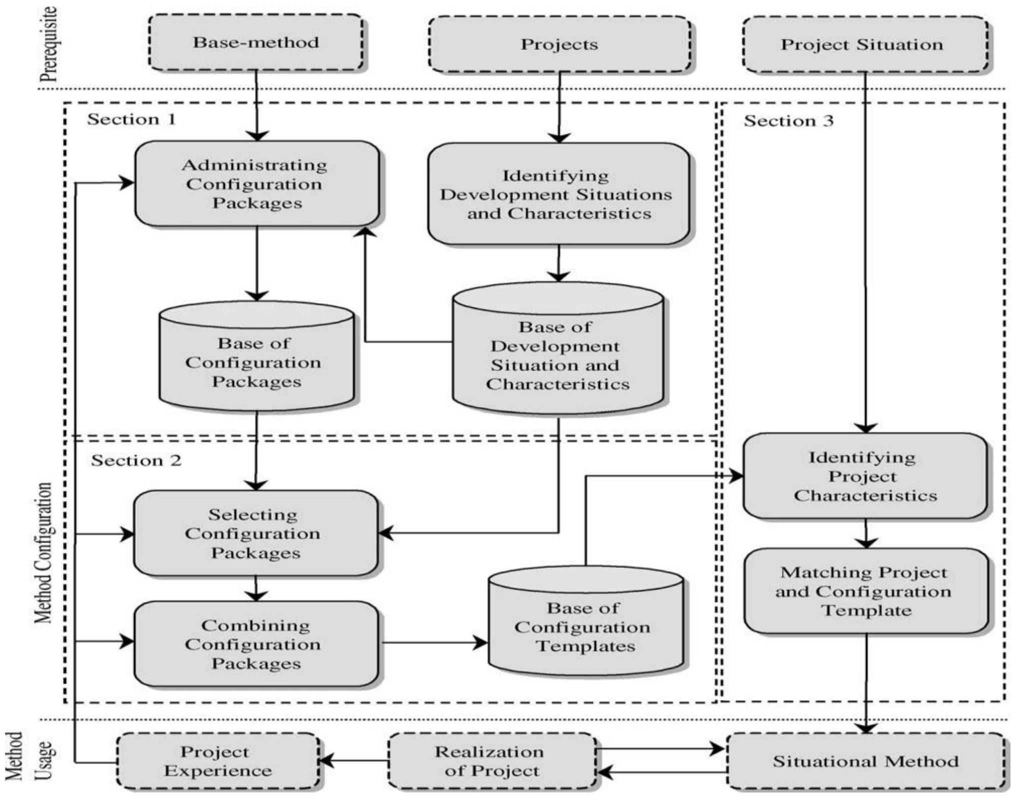

Prerequisite
Base-method
Projects
Project Situation
Section 1
Section 3
Administrating Configuration Packages
Identifying Development Situations and Characteristics
Base of Configuration Packages
Base of Development Situation and Characteristics
Section 2
Identifying Project Characteristics
Selecting Configuration Packages
Matching Project and Configuration Template
Combining Configuration Packages
Base of Configuration Templates
Method Configuration
Method Usage
Project Experience
Realization of Project
Situational Method

Figure 1.9:- MMC Framework (Karlsson and Ågerfalk, 04)

The criteria for selecting base method is external to this approach. Therefore, the issues like - *the selection of Base Method* and the *granularity of method component* remains unanswered.

2. Method Component for Method Configuration

(Wistrand and Karlsson, 04) proposes a conceptual construct to facilitate the method engineer's task of method configuration and termed it as "Method Component". They have formally established Method component as "A self –contained part of a system engineering method expressing the process of transforming one or several artefacts into a defined target object and the rationale for such a transformation".

In this approach, the project requirements are collected in the form of artefacts. These artefacts can take recommended inputs as <prerequisite> and delivers <outcome>. Based on the outcomes, goals are identified. The identified goals are further used to configure method components by using the following process:-

The Process

- **Define the Input/output of a method component** - Method component consists of artefacts, each artefact has a value either prerequisite or outcome. Prerequisite is the inputs for the method component whereas output or deliverables are specified by the outcome artefact.
- **Define the operations performed by the method component** - Content part of a method component is defined by Internal View of method component. *Internal View* as shown in fig. 1.10 focuses on the operational part of a method.

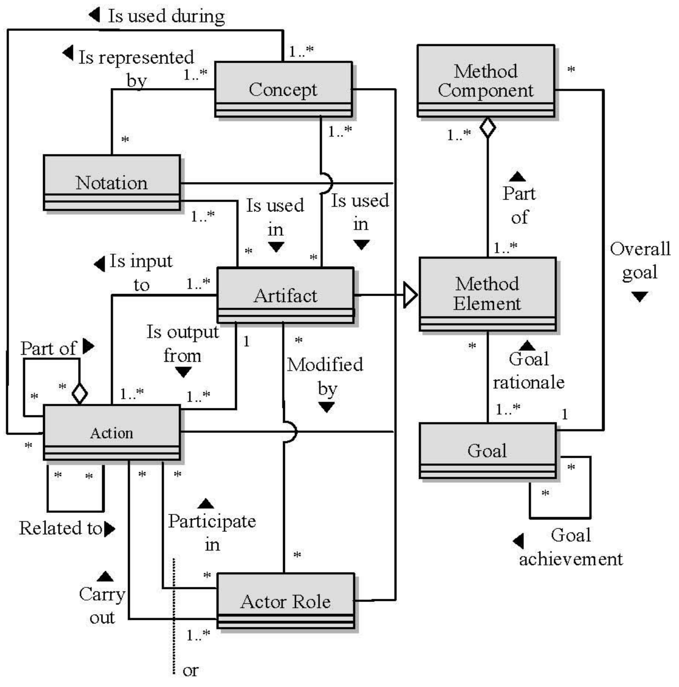


Figure 1.10:- Internal View of a Method Component (Wistrand and Karlsson, 04).

- **Assembly of method components** - To satisfy the overall goal of a method, method components are combined to form situation specific methods. The connection is made with the external view of the component that considers method component as a black box.

Method component approach is certainly a step ahead in configuration process but to make the process generic the *method component need to be instantiated with the underlying meta-model and further to a generic model*. Previously, generic Meta model are proposed by (Ralyté et al., 2003: Prakash, 2006).

3. Method Families for method configuration

(Rolland, 09) proposes method configuration in the form of method families; these method families are further surfaced to form a method line that ultimately results in a configured method. The proposal fails to provide a detailed, consistent and generic process for configured method construction.

1.4.5 Industrial Case studies on Configuration.

In the reviewed literature, various proposals were found where configurability proves as a possible tool for providing practical solutions to various industries like Intel Shannon, IBM, Nokia, to form situation specific methods.

1. Customizing Agile methods at Intel Shannon - (Fitzgerald et al., 06) explores tailoring of agile methods at Intel Shannon. The outcome of the investigation suggests that agile methods can improve delivery time and reduces defect densities. Developers at Intel Shannon found that agile processes may individually be incomplete to support the whole development process well; to get maximum assistance their processes can be tailored. In the research, they have shown that Extreme Programming (XP) is tailored and only 6 out of 12 key practices of it are used and combined with another agile method i.e. Scrum. XP is particularly useful for technical development stages (Beck, 99) and Scrum provides the necessary overall project management process (Schwaber and Beedle, 2002). By configuring and assembling these two most popular agile methods, developers make the development process more efficient and organized.

2. **Applying Scrum principles to software product management-** (Vlaanderen et al., 2011) has extended the agile method Scrum principals to software product management that enable the product managers to cope with complex requirements. In Scrum, the final product is developed by several teams in a series of flexible black boxes called 'sprints'. No new requirement can be introduced during these sprints. In the Scrum framework (Schwaber and Beedle, 2002) the two backlogs Product backlog and development sprint backlog plays a significant role. Product backlog contains a prioritized list of items relevant to a particular product once a requirement has been fully specified, with the approval of a developer they can be copied to development sprint backlog for further processing.

From the practical experience, authors identified that large and complex systems require a steady flow of elicited requirements is necessary for smooth functioning of the process. To meet this requirement, the Scrum is extended by introducing a Product Management Sprint Backlog (PMSB). The PMSB takes input from PB and outs back the requirement definition onto PB inside the PMSB there exist a requirement refinery that refines the complex requirements from coarse-grained to fine-grained to handle the requirements of different granularity.

Later on, constraints have also been checked to ensure feasibility and compatibility these definitions are further tested with architects and designers.

3. Configurability of work products – (Cameron, 2002) conducted research at IBM and found that the components of the configurable model are defined as modules and these modules are further grouped together to form various subsets called as work products. For each distinct work product there exists a Work Product Descriptor (WPD). WPD's describe "*what the work product is, why and when it is needed, and how it is produced*". These WPD's encapsulates the knowledge about the work products and are an efficient resource of information. Based on the information stored in WPD's, the work products are chosen for reuse.

4. Customized Agile methods at Nokia Corporation- (Kahkonen, 2004) reveals some Customised agile methods that are being practically implementing for software development at Nokia Corporations. The method applies the Community of Practice theory (COP) to analyse and solve the multi-team communication and coordination between different parts of the organisation to perform an accurate, straightforward task. The customized methods using in the group are:

- *Rapid7*: Works for requirement elicitation process, it suggests that for fast requirement elicitation stakeholders should get involved from the earlier stages to the later stages results in the reduction of Calendar time used for software development.
- *Integration Camp*: In traditional approaches, the integration of the components was done by integration teams that work independently. But to make the process agile Nokia Corporation introduces the idea of arranging the separate integration camps. In these camps, integration teams work in full coordination with the development team for the integration and testing of the component.

The case company suggested the use of Facilitated workshops in various domains of software development like in requirement domain, in architecture management, design phase or in project management. These workshops help the multiple teams spreading in the different parts of the organisation to perform the defined task effectively and efficiently. Many more case studies are found for example method tailoring at Motorola presented by (Fitzgerald et al., 2003), (Green P., 2012) describes Scrum adoption at Adobe and (Benefield, 2008) presents the effect of using agile at Yahoo.

1.5 Problem statement

From the literature survey in the foregoing sections, the thesis concludes following problems in the Method engineering domain.-

1. Early approaches to method assembly had a number of problems like (i) The appropriateness of the retrieved method component, the retrieved method components may or may not be found suitable to form the desired method. (ii) Ensuring coherency of the integration or the assembly process.

   Method generation approaches mitigated the problem of instantiation of the metamodel to some extent, but still partial instantiation was needed. In addition, to this, they lacked proper guidance to method engineer. Recently there are proposals to provide a rich set of guidelines and structured approaches to form a coherent method (Prakash and Goyal, 07: Ahmadi et al.,08). Thus, method construction task is performed in a more disciplined and cohesive way. Still the issue of appropriateness of the method component being selected remain unanswered.

2. The proposals on Method Configuration are in infancy stage they are centred on one single base method that is configured to form situated method. This reduces the scope of method generation for each new project and hence decreases the flexibility in the process.

   Analogous to the system configuration, the method configuration model needs to be sufficiently generic to specialise in a range of method models. This requires to address issues like : (i) A Configurable metamodel used to model the concepts of the configured method, (ii) what a ‘good component’ is and what the ‘right granularity’ is (iii) Selection of appropriate method component (iv) The complete process of method configuration to reach the desired coherent method.

3. The industrial case studies in the foregoing section show that no single agile method can be directly applicable to a particular project. It may need to be adapted, tailored or extended. This calls for a Method Configuration process where these light-weight methods can be configured by adapting an existing agile method or extends it by adding new practice or combine practises of two methods. Also latter method formed should confine to the principle of agility.

Thus, the **problem addressed** in this thesis is as follows:

Develop a method configuration process to build project-specific method consisting of different activities such as tailoring and extending.

Attarzadeh in his work found (Attarzadeh, 2008) that large numbers of software projects fail due to the high reliance on inappropriate Software development paradigm. The development methodologies whether agile or non-agile have their merits and demerits. So there is a need to draw some criteria that assist the software developers to select appropriate software development paradigm for the current project.

Hence, the research problem now ends up in the following sub-goals:

- Selection of Methodology Paradigm
- Configuring Traditional Methods
- Configuring Agile Methods
- Extending Methods

1.5.1 Selection of Methodology Paradigm

The first research goal of the thesis is to develop a Decision Support System that assists the software developers in selecting the appropriate Software Paradigm for the project-in-hand. This requires process for:-

- Identification of project characteristics for lifecycle selection.
- The finding impact of each of the above-identified project characteristics on the software development selection.

1.5.2 Configuring Traditional methods

The second research goal of the thesis is to develop a configuration process that can configure the project-specific method from the method components reside in the method base. The problem ends with some subproblems that the research work had taken care of

- Develop a Metamodel that can model the concept of configured method.
- Define method components that support essentiality attribute.
- Develop a Method configuration process to arrive at the coherent method by selecting situation specific method component.
- Design a Method base.

1.5.3 Configuring Agile methods

The third research goal of the thesis is to develop “The process for the selection of suitable agile method and further tailor it to form situation specific method”.

Given the above, the thesis investigates the tailoring of agile methods in actual practice. Specifically, the research objective is to:-

- Defining the agile configurable method model.

- Defining the organisational characteristics to find suitability of the methods
- Finding the most appropriate methods.
- Provide guidelines to tailor further the most suitable method found.

1.5.4 Extending Methods

Extending a method means balancing the consistency needs of business enterprises with the flexibility required by project teams. The applicability of the method thus formed will be significantly improved than the existing methods because the extended method thus formed contains the necessary constituent of a different method. Hence, the final research goal of the thesis is to "Develop a Method Extension process to extend the selected methodology that adapts the practices as per the requirement of other methods".

1.6 Thesis approach

This section describes the solution approach that the thesis adopted for achieving the above-stated research goals. The above sub-goals of the thesis can be viewed as composed of four modules. These modules are not independent but are related to each other. Next, the chapter gives a brief description of the approach followed to address the problems for the sub-modules.

1. Selection of Methodology Paradigm

Thesis identified 22 significant project characteristics like requirements, development team, users, project type, associated risk, etc. and evaluated their impact on the software development life cycle. These project characteristics are selected based on the past knowledge of Agile and Non-agile methods (Sommerville, 2010: Abrahamsson, et, al., 2002). The contingency factor approach by slooten (Slooten, et, al., 1996) was also considered in identifying these characteristics, since the contingency factor approach believes *that the* *specific features of the development context should be used to select an appropriate method from portfolio of methods.*

The project characteristics depends upon the situation in hand and will vary project to project. For example value of risk involved should be more for safety systems than for general purpose software. Next the weights are assigned to the identified project characteristics. Initially, the weight was distributed manually by the case project. But the process is very complicated and requires an enormous amount of calculations. Therefore, to make the prediction process intelligent, different machine learning algorithms were explored and neural network was selected for its simplicity. The Feed-Forward Back-Propagation neural network (Sivanandam and Deepa, 2007) with one hidden layer was tuned for the task, different values at the output layer helps predicting the software development paradigm.

2. Configuring Traditional Methods

The research presents a Method Configuration process on configuring traditional methods. The process is based on *Configurable Meta Model (CM)* that is obtained by modifying Decisional Metamodel. As explained in Sec 1.4.2, the Decisional Metamodel is an instantiation of the generic model. Correspondingly, the Configurable Metamodel is also an instantiation of the generic model that configurability is generic in nature and can be used to configure a number of methods.

The Configurable Metamodel is used to realize the concepts of Configurable Method Component (CMC). The Configurable method component has an *Essentiality* attribute that can take two values either *Common* or *Variable*. Common are those method concepts without which a method will lose its identity whereas variable is the configurable part of a method.

The Configurable Metamodel supported the notion of atomic and compound methods and proposed that the right granularity components are entire methods, whether atomic or compound. This is because such methods provide to us the most fundamental, coherent assembly of method components.

These configurable method components reside in the method base, and retrieval is based on situational characteristics determined by Global Properties of the method.

Finally, the configured method formed by considering the purposes of the artefacts chosen for the situated method and dependencies that need to be satisfied in configuring the method.

3. Configuring Agile Methods

The thesis introduces an *Agile Method Engineering (AME)* process, to form situation specific method. In Agile Method Engineering, the projects situations are gathered in the form of organisational requirements. These organisational requirements are then fed to Fuzzy Logic Controller to find the weight of the agile methods. Here the term ‘weight’ refers to the degree of applicability of the method for the specified set of requirements. The highly ‘weighted methods’ or ‘most suitable methods’ are retrieved from the method base, further, these retrieved methods are configured to form project-specific method.

Similar to Traditional method engineering, AME also supports an Essentiality attribute for the agile methods. Since, these methods adhere to a set of practices it's hard to produce a generic model for the purpose. The agile values defined in the agile manifesto are seen to define ‘essentialities’ in these methods. Further to configure a method for an agile project, each project is considered individually. The functional requirements are extracted from the projects; these requirements provide support to the method engineer for deciding the ‘variability in the methods’.

4. Method Extension Process

During the research, it was observed that there may be some requirements that may not be covered by the configured method alone. To satisfy the complete set of project requirements, the candidate method may need to extend to other method practices. Method extension addresses this need; method extension starts from the process framework of the method, followed by the selection of method components from other methods or finding the practices need to be added. Finally these are integrated to form the configured method.

1.7 Outline of the thesis

The structure of the thesis in terms of the contents of its various chapters is as follows.

Chapter 2 presents the decision support system for software paradigm selection for the project-in-hand. The decision support system is based on the weighted project characteristics and the project-specific input metrics for the identified project characteristics. The set of project characteristics is identified and discussed followed by the project-specific input metrics. Two case studies are shown for the illustration purpose.

This work has been published in Gupta, D. and **Dwivedi, R.** (2015). A frame work to support evaluation of project-in-hand and selection of software development method. *In Journal of Applied and Theoretical Information Technology*, 73(1), 137-148**.**

Chapter 3 presents the method configuration process for traditional methods. The configurable model is presented inconsistent with the Configurable Metamodel to support the method configuration process. The selection and retrieval of method component is discussed, finally presents the process to fine tune the configurable method component into the situated method.

This work has been published in Gupta, D. and **Dwivedi, R**. (2012). A step towards Method Configuration from Situational Method Engineering. *Software Engineering International Journal***,** 2(1), 51-59 and **Dwivedi, R.** and Gupta, D. (2015). A Complete method configuration process for configuring project-specific methods. In *Journal of Software*, 9(3), 29-40.

Chapter 4: This chapter deals with the process for the selection of suitable agile method and further tailor it to form situation specific method. The complete process for configuring agile methods along with the case studies is presented in this chapter.

This work has been published in **Dwivedi, R**. and Gupta, D. (2015). Applying machine learning for configuring agile methods. *In International Journal of Software Engineering and its Application*, 9(3), 29-40.

Chapter 5: This chapter deals with the extension process blend of different agile methods based on the rich knowledge of the past usage of these methods under different requirement sets. The applicability of the method thus formed will be significantly improved than the existing methods because the extended method thus formed contains the required constituent of each method. Hence the chapter focuses on the development of a method extension process capable enough to support entire set of situational project-requirements.

This work has been published in **Dwivedi, R.** and Gupta, D. (2015). The Agile Method Engineering: Applying fuzzy logic for evaluating and configuring agile methods in practice. *In International Journal of Computer Aided and Engineering Technology*. (In Press).

Chapter 6: In this section, the conclusion of the thesis work and the future scope of the work is presented.

Chapter-2

Software development paradigm selection

This chapter presents the Decision Support System that provides the set of 22 project characteristics like *complexity, modularization of the task, business risk, technical risk, and programmer's capability*. These project characteristics define the overall context of the situation and comfort the exploration of a project that further helps in the selection of software development paradigm. Since the process of evaluation is complex in nature; the neural network has been used for the realization process.

2.1 Software Development Lifecycle Paradigm

(IEEE Standard 610.12, 1990) states that “The period of time that begins with the decision to develop a software product and ends when the software is delivered. This cycle typically includes a requirements phase, design phase, implementation phase, test phase, and sometimes, installation and checkout phase”.

A Methodology is a systematic way of developing the product using tools, techniques, strategies, and guidelines. It consists of- *Model of the product constraints applicable to product, steps that can be prefigured, transition criteria between steps and life cycle of the software development*. Method or methodology supports complete lifecycle or partial life cycle. For example, (Yourdon, 89) supports full lifecycle and Entity Relationship Diagram supports partial life cycle. Traditional software development methodologies are divided into two domains: - *Function oriented methods and Object-Oriented methods.* Function oriented methods distinguish between data and function. Functions are the active part (behaviour) whereas, data is the static part (affected by function). Examples of function-oriented methods are Yourdon structured analysis (Yourdon, 89), Structured Analysis /Structured Design (DeMarco, 78) etc.

Object-Oriented methods define ‘Objects’ as a cohesive unit that can encapsulate a ‘set of methods’ and ‘state’ to which methods can have access.

The Unified Modelling Language is a modern Object-Oriented method, and is a unification of different Object-Oriented methods proposed by (Rumbaugh et al., 91).

Traditional approaches involve a significant overhead in planning, designing and documenting the system (Livermore, 2008). When these heavyweight, plan-based development approaches were applied to small and medium-sized business systems, the overhead involved was so large that it sometimes dominated the software development process. More time was spent on how the system should be developed than on the program development (Livermore, 2008: Highsmith, 2002: Awad, 2005). Dissatisfaction with these traditional approaches led a number of software developers in 1990s to propose new *Agile methods*. These methods allow the development team to focus on the software itself rather than on its design and documentation part (Rizwan and Qureshi, 2012). Agile methods are intended to deliver working software quickly to customers and allow changes in requirements to be included in a later iteration of the software development process.

Well known agile approaches include Extreme Programming (Beck, 99a), Scrum (Schwaber and Beedle, 2002), Crystal (Cockburn, 2000), Adaptive Software development (Highsmith, 2000), DSDM (DSDM consoritium, 97) and Feature Driven Development (Hunt, 2006).

Agile methodology gives solutions to many of the limitations imposed by traditional methodologies, but has their own problems like *lack of Skilled Professionals, the initial* *project doesn't have a definite plan, the final product can be grossly different than what was initially intended* (Narur et al, 2005).

From the in-depth study, of the recent works where these methodologies (both agile and traditional) have been adapted and applied to software development in practice. It can be concluded that software paradigms have their own limitations- Traditional plan-based software development methodologies work extremely well if the requirements are static whereas for frequently changing project requirements these methodologies are often considered as slow and insensitive. For example - Large organisations like Nokia (Kahkonen, 2004), Motorola (Fitzgerald et al. 2003), Adobe (Green P., 2012) and Yahoo (Benefield, 2008) are using agile methodologies and found that they are not well suited for development systems with the development teams in different places and where there may be complex interactions with other hardware and software systems. Agile methods are also not recommended for critical systems development where a detailed analysis and documentation of all of the system requirements is necessary to understand their safety or security implications (Boehm and Turner, 2004).

So it can be concluded that before developing a development engine for engineering methods or before configuring methods (either agile or non-agile) one has to be clear with the selection of development paradigm. The base to decide the paradigm is the project characteristics such as requirement characteristics, development team expertise, user participation and associated risk in the project. Next section identifies the details of these characteristics.

2.2 Domain analysis.

The four major domains that show significant impact on deciding the software development methodology are - : characteristics of requirements, characteristics of development team, user participation and project type. These domains are identified based on the literature available on the characteristics of software development methodologies for instance, (Syed-Abdullah et al., 2007) identified the role and skills of development team in implementing the software development methodology (Sultan and Chan, 2000) identified that users and project requirements plays critical roles in acceptance and implementation of software development methodology. (Narur, et al., 2005) described that project type and associated risk is also an important domain for deciding the software development methodology

1. **Characteristics of Requirements: -**Working with project requirements is a challenge, for some situations projects requirements are volatile, difficult to understand or initially not complete. For others, they may be complex or critical. The selection of the software development methodology is highly dependent on the characteristics of the Requirements gathered in the requirement phase. Project characteristics like the volatility of requirements, complexity, a number of requirements gathered initially, etc. are designed to address this need.
2. **Characteristics of Development team: -** Software development selection depends upon the development experience of the team members. Some of the team members have less experience, some have experience but little domain knowledge and some have expertise in the project field but lacks the familiarity with the technology being used in the project. Training is also an important factor to consider, as to what extent the team has to be trained and how much recourses it requires in terms of time and cost leaves a significant impact on project development. Project characteristics like tool experience, application experience, programmer's capability are designed to address this need.
3. **User's participation: -** Selection of software development method also depends on the involvement of different users during the software development. User here refers as the stakeholders of the projects such as – end users or management team. Sometimes project

requirements demand user to be present at all phases of development life-cycle wheras, other classes of projects may not require. Project characteristics like clarity and completeness of requirements, necessary functions, and business risk are designed to address this need.

4. **Project type and associated risk: -** Project type and associated risk plays a significant role in paradigm selection. Project type here refers as complex and simple projects. For example, projects requiring strict deadlines, high reliability, human life risk are complex. Whereas, projects like payroll management and hotel reservation system are considered as simple projects. Some of the important attribute like project funding, strict deadlines, high reliability, risk in terms of money, people, etc. have a great impact on the decision of software development methodology (Boehm and Turner, 2004). Other project characteristics like technical risk and operational risk also play a significant role in selection of development paradigm.

The outcome set of project characteristics identified by analyzing these domains is used to categorize the suitability of software development methodology for the project- in-hand.

2.3 Decision Support System for Software development methodology selection

Decision support system is based on the *weighted project characteristics (Wi)* and on the *Input metrics of the project characteristics (Pi).* In the following section project characteristics are assigned weights. There are two types of weights:-

- *Initial Weight* –Initial weight remains constant for all projects and are assigned based on their suitability for the software paradigm, they are referred as weighted project characteristics (Wi).
- *Current Weight* – Current weight in this thesis is referred as input metrics for the

project characteristics (Pi) and is derived from the project-in-hand.

Sec 2.3.1 presents the initial weight assignment and sec 2.3.2 presents the current weight.

2.3.1 Project Characteristics and their Weight distribution parameter (Wi):-

Weight assignment is a critical task, there has to be some plausible mechanism behind it. For the current problem, the output range is divided into three-

1. Between '1 to 4' -Agile methodology is best suited for the current project.
2. Between '4 to 5' Hybrid methodology is best suited for the current project.
3. Between '5 to 8' Traditional methodology is best suited for the current project.

After analyzing the identified characteristics the 'decision' formed for assigning the weight to the project characteristics is:-

"Assignment should be done, in such a manner that the project characteristics having more support for traditional has given more weight and the project characteristics having more support for agile has given less weight."

The project characteristics are given below, to get better understanding the weight distribution criteria of some are also explained. The numerical value assigned to the weights ranges between 0.0 - 0.1.

Project Characteristic 1: Volatility of requirements

This project characteristic signifies the *frequency of changing requirements*. Agile methodology has dynamic characteristics they are also known as 'dynamic methods' and are capable enough to address these needs. Results very less weight i.e. 0.02 for this.

Project Characteristic 2: Complexity

The project that requires a detailed analysis and high documentation in the initial phase are complex projects. More value of this characteristic strongly supports traditional methodology to follow, results in very high weight i.e. 0.1 for this.

Project Characteristic 3: Business Risk

Business risk is related to return on investment and customer satisfaction. For example, suppose a customer is unsatisfied with the product after release and hence it has no market value then the organization should be able to release a new version, but it will be very time-consuming and costly for organization. In this case, the risk is high, and organization will suffer a massive loss.

More value of business risk supports for agile development because the customer here is always available while development and product are released in increments, not, in the end, so any deficiency can be detected early. Results very less weight i.e. 0.03 for this.

Project Characteristic 4: Technical Risk

Technical risk involves the non-availability of the developer, non-availability of technology that is tools, etc. during development. It may occur due to the failure of the tool during development or leaving of the developer before completion of the task.

Since, traditional methodology addresses this risk to a great extent. Results a high weight i.e. 0.08 for this.

Project Characteristic 5: Operational Risk

This is the risk involved due to the failure of some functionality of the project. If the impact of such failure is high, operational risk is high. For example, suppose in some safety system if any feature fails then their impact will be very high so, operational risk is high.

More value of this characteristic supports traditional methodology because these types of systems should be designed a systematic and properly defined way. Results very high weight i.e. 0.1 for this.

Project Characteristic 6: Flexibility

Flexibility is the ease with which an operational program can be modified. Agile methodology is best suited for the project having more flexibility because it will be easy to develop and deliver software in increments. Results very less weight i.e. 0.02 for this.

Project Characteristic 7: Modularization of Task

Modularization is paramount for quick and secure software development. If tasks are divided into modules, then it will be very easy to develop the modules in parallel for quick release.
Agile methodology is suitable for development if functions can be divided into modules. Results very less weight i.e. 0.02 for this.

Project Characteristic 8: Time to Market

This characteristic signifies the time (in months) before which at least first phase (least functionality) of the product must be released. Agile methodology delivers in very short sprints to the user. Results very less weight i.e. 0.02 for this.

Project Characteristic 9: Amount of requirement known initially

It is not possible to know all the requirements initially for several projects. Some requirements are visible only after using the minimum workable (first release) of software.
For a limited number of elicited requirements, the agile methodology should follow because the customer is always involved and they can add requirements at later stages. So, less weight is given to this metric. For example, *Unknown users* are unable to explain complete

requirements, they can only give an outline and can proceed to the depth after some iterations.

Project Characteristic 10: Clarity and Completeness of requirement

Clear and complete requirements are - well defined, clearly visible and require minimum further analysis. These types of requirements can be addressed by both traditional and agile. Results in a medium weight i.e. 0.05 for this.

Project Characteristic 11: Expandability

Expandability shows the ‘ease’ with which software can accommodate additions to its capacity. For example a large number of shoppers, visiting a shopping website at the discount time. The more value of this characteristic has more support for agile methodology. Results very less weight i.e. 0.02 for this.

Project Characteristic 12: Coupling

Coupling is the degree of dependency between functionalities. Coupling increases complexity and hence more value to it supports for traditional methodology. Results very high weight i.e. 0.09 for this.

Project Characteristic 13: Tool Experience

The year of work experience the developer has, on the tool to be used for the development. Since, agile development supports simple and automated tools to a large extent. Results a low weight i.e. 0.03 for this.

Project Characteristic 14: Platform volatility

How frequently the projects need to adapt the platform changes. Agile methodology creates self-contained modules they are developed to accept technical changes. Results a low weight i.e. 0.02 for this.

Project Characteristic 15: Application Experience

The work experiences the developer on the desired application. Since, agile supports collaborative and cooperative environment for the development. Collective ownership is also there; that provides the peer group support to the developers at each level. Results a low weight i.e. 0.02 for this.

Project Characteristic 16: Programmer's capability

It defines the programmer's capability to understand and develop the project. In agile, not only the developer but the complete technology team work together to achieve a common goal. Hence, getting a better understanding of the project. Results a low weight i.e. 0.03 for this.

Project Characteristic 17- Add-on Function

Percent of Add-on functions/ fancy functions/ exciting functions need to be developed. Since, agile supports user involvement at all phases of development. Hence, functionality can extend or customized according to users need to make him happy. Results a low weight i.e. 0.02 for this.

Project Characteristic 18- Necessary/ Critical Functions

These are critical functions that should be developed in a defined manner with proper documentation. Traditional methodology supports a well-documented environment. Results a high weight i.e. 0.09 for this.

Project Characteristic 19: Reuse of existing code

For the development of the current project, the amount of code that can be taken from existing code. In agile, modules/sprints are independent, self-contained components that can be inherited from other projects easily. Results a low weight i.e. 0.03 for this.

Project Characteristic 20: Develop for reuse

If a project is to be developed as a ‘base project’ for future use, then it should be well documented, defined and structured. The quality of such product should be very high. The traditional methodology is best suited for this need. Results a high weight i.e. 0.07 for this.

Project Characteristic 21: Platform experience

The developers experience needed for the platform to be used for the current project. It is required for both, so an average weight of 0.04 is assigned to it.

Project Characteristic 22: Team cohesion

Ease of communication and interaction among team members is known as team cohesion. It is necessary for a good end product in every paradigm. Results an average weight of 0.05 for this.

The set of project characteristics along with the weight distribution criteria has been shown in the Figure 2.1.

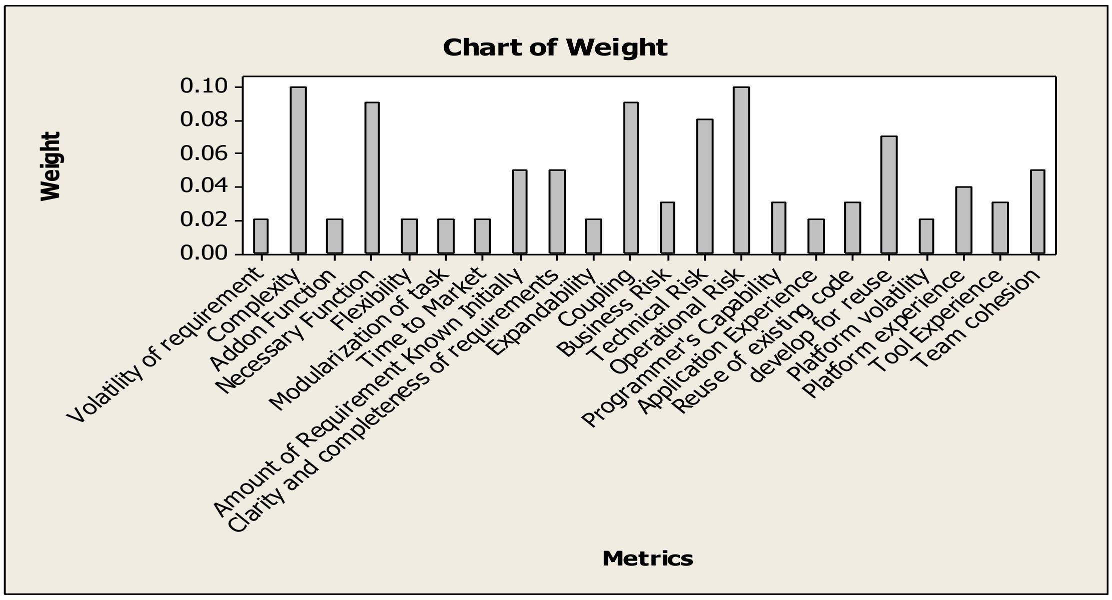


Figure 2.1:- Graph for weighted project characteristics.

2.3.2 Input metrics for the project characteristics (Pi)

As mentioned earlier, the decision support system depends on the weighted project characteristics and on the Input metrics of the identified project characteristics. Weights are constant. However, the input parameter varies from project to project.

The metrics for these project characteristics need to be decided. Since, all input metrics cannot be measured on the same scale; different measurement scales are drawn based on the behavior of the project characteristics.

Measurement parameter for identification of input values for different metrics:-

Metrics for - Volatility of requirements

Table 2.1: Metrics for “Volatility of requirement”

Percentage of volatile known requirement	**Category**
Less than 10%	Very Low
10-19%	Low
20-29%	Medium
30-39%	High
Greater than 39%	Very High

Metrics for - Complexity:-

Complexity for software methodology selection is defined as the initial research or documentation required before actually start a project.

Table 2.2: Metrics for “Complexity”

Initial Time	**Category**
Less than one week	Very Low
Less than 15 days	Low
Less than one month	Medium
Less than six months	High
More than six months	Very High

Metrics for - Business Risk

Business risk is influenced by numerous factors, including sales volume, per-unit price, input costs, competition, and overall economic climate and government regulations. So, the value of this metric should be chosen by considering all the above-said factors.

Metrics for - Technical Risk

Technical risks are range from software malfunctions to electrical failure to viruses that can completely shut down a firm’s operation. These are serious risks that a company must plan to face. The risk involved with installing new system also comes with technical risk. When firm shifts to a new system without appropriate integration, the new system is not able to accomplish all that was promised. Sometimes it even performs poorer than the system it replaces. The new system often requires employees to operate according to new processes. These may be difficult to learn, take training to execute correctly, or may even be outright resisted by employees who prefer the old way of doing business. So, the value of this metric should be chosen by considering all the above said factors wisely.

Metrics for - Operational Risk

Operational risks include failure to address priority conflicts, failure to resolve the responsibilities, insufficient resources, no proper subject training, no resource planning and lack of communication in the team. So, the value of this metric should be chosen appropriately. If the impact of such failure is very high then, operational risk is high. For example, suppose in some safety system if any functionality fails then their impact will be very great so, operational risk is high.

Metrics for - Flexibility

Table 2.3: Metrics for “Flexibility”.

Ease of modification	**Category**
Less than 5%	Very High
5-9%	High
10-14%	Medium
15-19%	Low
Greater than 20%	Very Low

Metrics for - Modularization of Task

Table 2.4: Metrics for “Modularization of task”.

Extent to which the project can be made modular	**Time to market**
Complete project	Very High
More than half functionality	High
Half of the functionality	Medium
Less than half	Low
Very minimum amount of modules	Very Low

Metrics for - Time to Market

Table 2.5: Metrics for “Time to market"

Time before the first release	**Time to market**
2 months	Very High
4 months	High
6 months	Medium
8 months	Low
Greater than 8 months	Very Low

Metrics for - Amount of Requirement known Initially

Table 2.6: Metrics for “Amount of requirements known initially.”

Amount of requirements known initially	**Category**
Less than 20%	Very Low
20-39%	Low
40-59%	Medium
60-79%	High
Greater than 79%	Very High

Metrics for - Clarity and Completeness of requirement

Table 2.7: Metrics for “Clarity and Completeness of requirements”

Amount of complete and consistent Requirements	**Category**
Less than 20%	Very Low
20-39%	Low
40-59%	Medium
60-79%	High
Greater than 79%	Very High

Metrics for - Expandability

This metric is chosen based on the ease of the modifications made to the software at later stages. The more value of this metric has more support for agile methodology. The effort required in addition to new functionality to the already working software. The value of this metric can be selected based on the effort estimation for the addition of new functionality.

Metrics for - Coupling

NCM=Number of modules to which a method is coupled.

Table 2.8: Metrics for “Coupling”

Value for NCM	**Modularisation of Task**
Less than 2	Very Low
2,3	Low
4,5	Medium
6,7	High
Greater than 7	Very High

Metrics for - Tool Experience

Table 2.9: Metrics for “Tool experience”

Developers experience on the tool to be used for project-in-hand. Time(in months)	**Platform Experience**
Less than 6 months	Very Low
6-12 months	Low
12-18 months	Medium
18-24 months	High
Greater than 24 months	Very High

Metrics for- Platform volatility

Table 2.10: Metrics for "Platform volatility"

Types of changes in platform	**Platform volatility**
Likely to evolve from one platform to another having different architectures (windows to Linux)	Very High
Likely to evolve from one platform to another having same architectures (Red hat to Ubuntu)	High
Likely to evolve from one platform to another having different version (windows XP service pack 2 to windows XP service pack 3) Or (Ubuntu 10 to Ubuntu 11)	Medium
No visible change found, but may require at later stage	Low
Never evolve	Very Low

Metrics for -Application Experience

Table 2.11: Metrics for "Application Experience"

Time (in months)	**Application Experience**
Less than 12 months	Very Low
12-24 months	Low
24-30 months	Medium
30-36 months	High
Greater than 36 months	Very High

Metrics for - Programmer's capability

This metric calculate the efficiency of the developer for developing the project in terms of knowledge, vision and dedication towards the work.

Metrics for -Add-on function

Table 2.12: Metrics for "Add-on function"

Percentage of functions developed as add-on functions	**Category**
Less than 20%	Very Low
20-39%	Low
40-59%	Medium
60-79%	High
Greater than 79%	Very High

Metrics for -Necessary functions

Table 2.13: Metrics for "Necessary function"

% of functions to be developed as necessary functions	**Category**
Less than 20%	Very Low
20-39%	Low
40-59%	Medium
60-79%	High
Greater than 79%	Very High

Metrics for -Reuse of existing code

Table 2.14: Metrics for “Reuse of existing code”

Reuse of Existing Code	**Category**
Less than 20%	Very Low
20-39%	Low
40-59%	Medium
60-79%	High
Greater than 79%	Very High

Metrics for -Develop for future reuse

Table 2.15: Metrics for “Develop for future reuse”

Purpose of the project	**Time to market**
Developed as a base project (developed only for reuse)	Very High
Probability of being used in another project is very high	High
It may require additional functionalities at a later stage	Medium
No open project found that requires code of present project-in-hand	Low
Never be reused	Very Low

Metrics for -Platform experience

Table 2.16: Metrics for "Platform experience"

Developers exp., on the platform used (in months)	**Platform Experience**
Less than 6 months	Very Low
6-12 months	Low
12-18 months	Medium
18-24 months	High
Greater than 24 months	Very High

Metrics for - Team Cohesion

Ease of communication and interaction among team members is known as team cohesion. The value for this metric can be chosen on the basis of the current situation of the organization and team members.

The characteristics of each metrics play a significant role in weight distribution. Weight distribution has been done based on the available literature (Sommerville, 2010: Fenton and Bieman, 2014) and from the knowledge and practical experience of various software developers in the leading software companies.

.

2.4 Proposed Algorithm

The following algorithm is used to predict the software development methodology for the current situation. The input to the algorithm is the set of project characteristics along with the identified weight, and the output is the value need to find the appropriate methodology. The algorithm is as follows:-

Step 1: - Assign input values to each metric for a given project from the five possible values.

Category	**Very Low**	**Low**	**Medium**	**High**	**Very High**
Value	1	2	3	5	8

Step 2:- Calculate S= $(W_i * P_i)_{\text{for } i=1 \text{ to } 22}$

Where, W_i is the weight assigned to the ith metrics that is fixed (constant)

and P_i is the input values chosen for ith matrices that are variable (project specific).

Step 3: - Select the suitable methodology for the given project on the basis of the value of S obtained in step2.

The output will range from 1 to 8. For the values, between '1 to 4', agile methodologies are a good solution for development and for '5 to 8' traditional methodologies will do well. For some projects, values lies between '4 and 5' indicating hybrid methodology may be used for the project-in-hand. The algorithm is demonstrated in the next section.

2.5 Case studies for Software Development Methodology Selection

2.5.1 Case study 1: On Mobile Application Development (MAD)

This section discusses the results that are obtained in the Mobile Application Development project. The dataset is formed by interviewing various developers of leading mobile companies. The data-set here is collected for the m-commerce development solutions for mobile application. The m-commerce delivers ideal mobile e-commerce solutions to the clients. The sample size is 40 developers of different organisations and the metrics formed the basis of the questionnaire for example – how volatile are the requirements etc. Values are assigned to each of the metric and the product of each input with their respective calculated weight. The questionnaire is given in appendix A and the results found are as follows

Table 2.17: Weight distribution, input values, their product and total sum for MAD

S. no.	Metrics	Weight (Wi)	Input Values (Pi)	S= Wi*Pi
1.	Volatility of requirement	0.02	Medium(3)	0.06
2.	Complexity	0.1	Very Low(1)	0.1
3.	Add-on Function	0.02	Medium(3)	0.06
4.	Necessary Function	0.09	Medium(3)	0.27
5.	Flexibility	0.02	High(5)	0.1
6.	Modularization of task	0.02	High(5)	0.1
7.	Time to Market	0.02	High(5)	0.1
8.	Amount of Requirement Known Initially	0.05	Medium(3)	0.15
9.	Clarity & completeness of requirements	0.05	High(5)	0.25
10.	Expandability	0.02	High(5)	0.1
11.	Coupling	0.09	Very Low(1)	0.09
12.	Business Risk	0.03	Very High(8)	0.24
13.	Technical Risk	0.08	Very Low(1)	0.08
14.	Operational Risk	0.1	Low(2)	0.2
15.	Programmer's Capability	0.03	Medium(3)	0.09
16.	Application Experience	0.02	Medium(3)	0.06
17.	Reuse of existing code	0.03	Medium(3)	0.09
18.	Develop for future use	0.07	Medium(3)	0.21
19.	Platform volatility	0.02	Low(2)	0.04
20.	Platform experience	0.04	Low(2)	0.08
21.	Tool Experience	0.03	Medium(3)	0.09
22.	Team cohesion	0.05	Medium(3)	0.15
	Total(Sum of product)			**2.71**

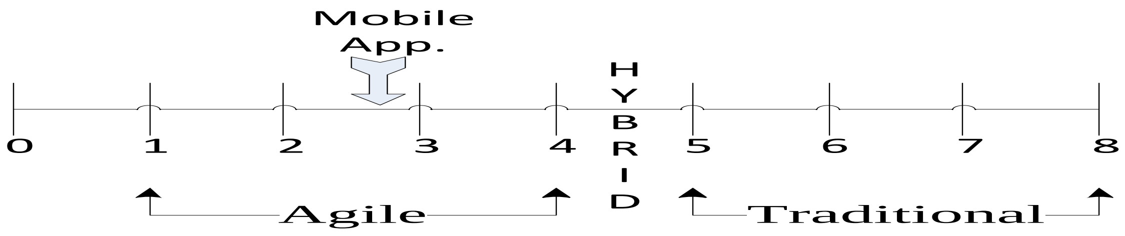


Figure 2.2:- Scale showing the output for MAD

Here for mobile application development, the output is 2.71, so it indicates that agile methodology is best suited for their development.

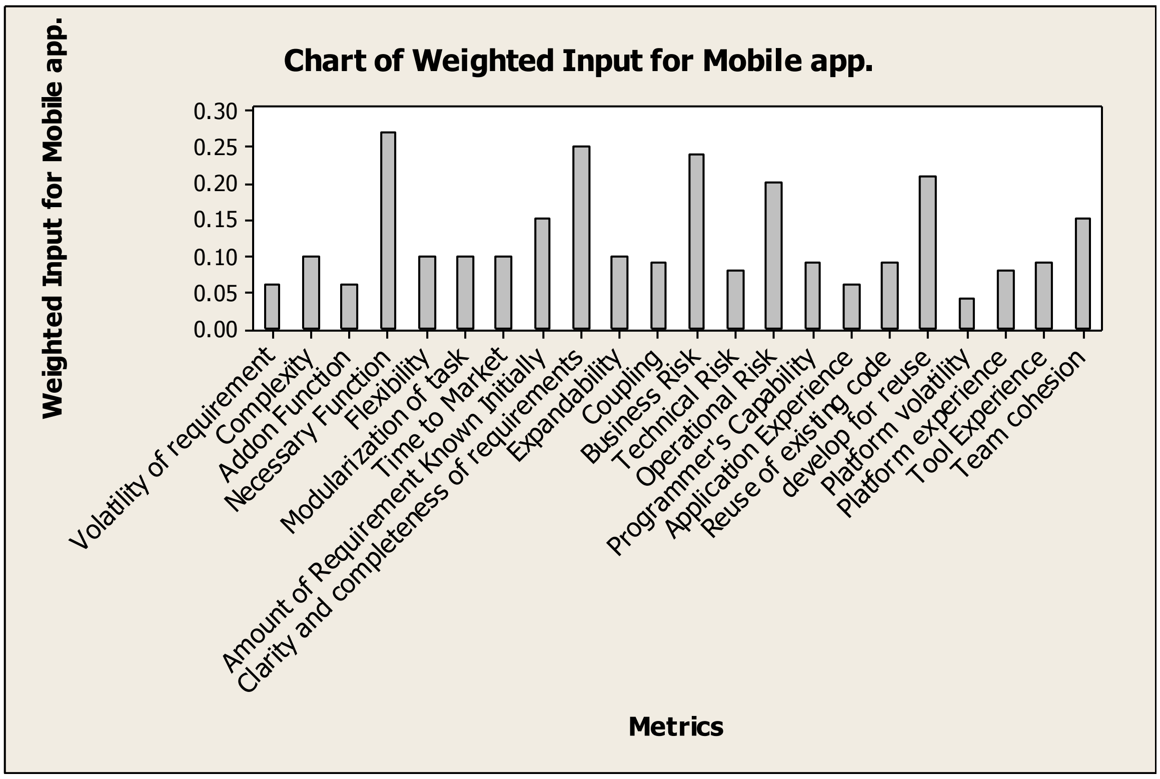


Figure 2.3:- Weighted input for MAD

2.5.2 Case Study 2: On Air Traffic Control (ATC)

For developing a system for Air Traffic Control and management, the data set produced by personally interviewing the ATC professionals at Airports Authority of India, INDIA. The questionnaire is given in appendix A and the results found are as follows

Table 2.18: Weight distribution, input values, their product and total sum for ATC

S. no.	Metrics	Weight (Wi)	Input Values (Pi)	S= Wi*Pi
1.	Volatility of requirement	0.02	Very Low(1)	0.02
2.	Complexity	0.1	Very high(8)	0.8
3.	Add-on Function	0.02	Low(2)	0.04
4.	Necessary Function	0.09	Very high(8)	0.72
5.	Flexibility	0.02	Low(2)	0.04
6.	Modularization of task	0.02	Very Low(1)	0.02
7.	Time to Market	0.02	Low(2)	0.4
8.	Amount of Requirement Known Initially	0.05	Very high(8)	0.4
9.	Clarity & completeness of requirements	0.05	Very high(8)	0.4
10.	Expandability	0.02	Low(2)	0.04
11.	Coupling	0.09	Very Low(1)	0.09
12.	Business Risk	0.03	Low(2)	0.06
13.	Technical Risk	0.08	Very High(8)	0.64
14.	Operational Risk	0.1	Very High(8)	0.8
15.	Programmer's Capability	0.03	Very High(8)	0.24
16.	Application Experience	0.02	Low(2)	0.04
17.	Reuse of existing code	0.03	Very Low(1)	0.03
18.	Develop for future use	0.07	High(5)	0.35
19.	Platform volatility	0.02	Low(2)	0.04
20.	Platform experience	0.04	Very High(8)	0.32
21.	Tool Experience	0.03	Very High(8)	0.24
22.	Team cohesion	0.05	Very High(8)	0.4
	Total(Sum of product)			**6.13**

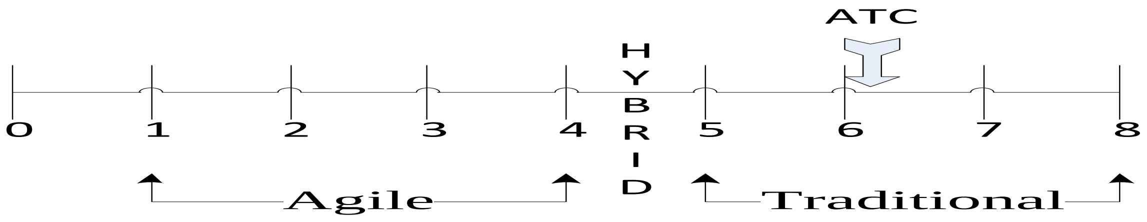


Figure 2.4: - Scale showing the output for ATC

The output of air traffic controller is 6.13, which indicates that traditional methodology is best suited for their development.

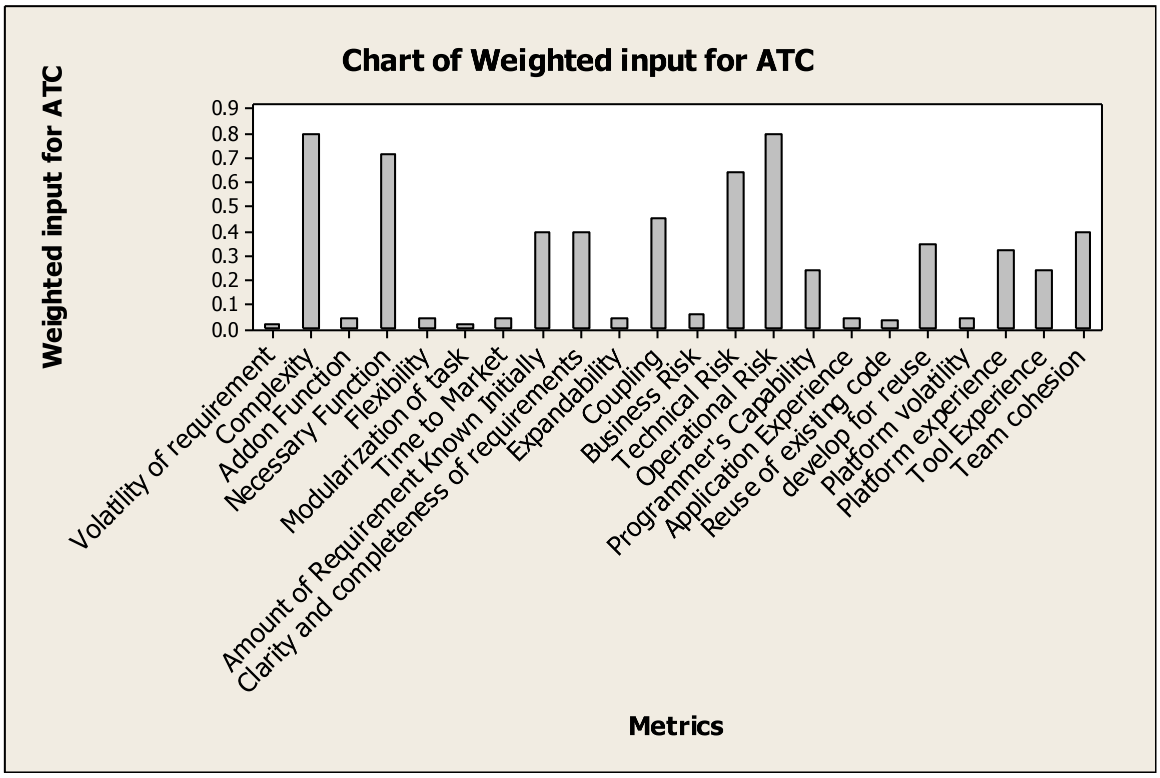


Figure 2.5: -Weighted input for ATC

2.6 Implementation Details

Neural networks process information in a similar way the human brain does. This is a layered architecture basically having a single layer, double layer or multiple layer neurons. In general for multilayer neural network, the layers are one input layer, one or more hidden layer, and one output layer. Each of the input has some associated weight and network learns by adjusting these weights.

The weight adjustment is done by providing a large number of samples (examples) having input and their target values. Ones the network is trained, it can be used as an efficient tool for the specified task. In this research, neural networks are trained for weight distribution. Initially, the weight distribution has been done manually, and the result has been obtained for the case project. Since the process requires a lot of complex calculations, neural networks are used for the purpose. This simplified the task of the developer to a great extent

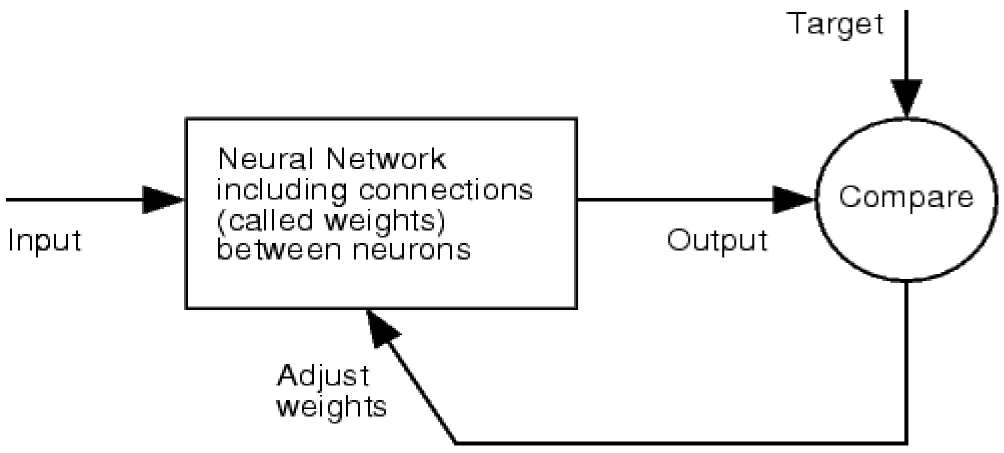

Target
Neural Network including connections (called weights) between neurons
Input
Output
Compare
Adjust weights

Figure 2.6:- General Architecture of Neural Network

Problem mapped as neural network

The decision support system is simulated by three layer feed-forward back propagation neural network having input, hidden and an output layer. Neural network tool available in Matlab toolbox is used for training and simulation. The network consists of twenty-two neurons at input layer (number of inputs), three neurons at hidden layer and one neuron in the output layer.

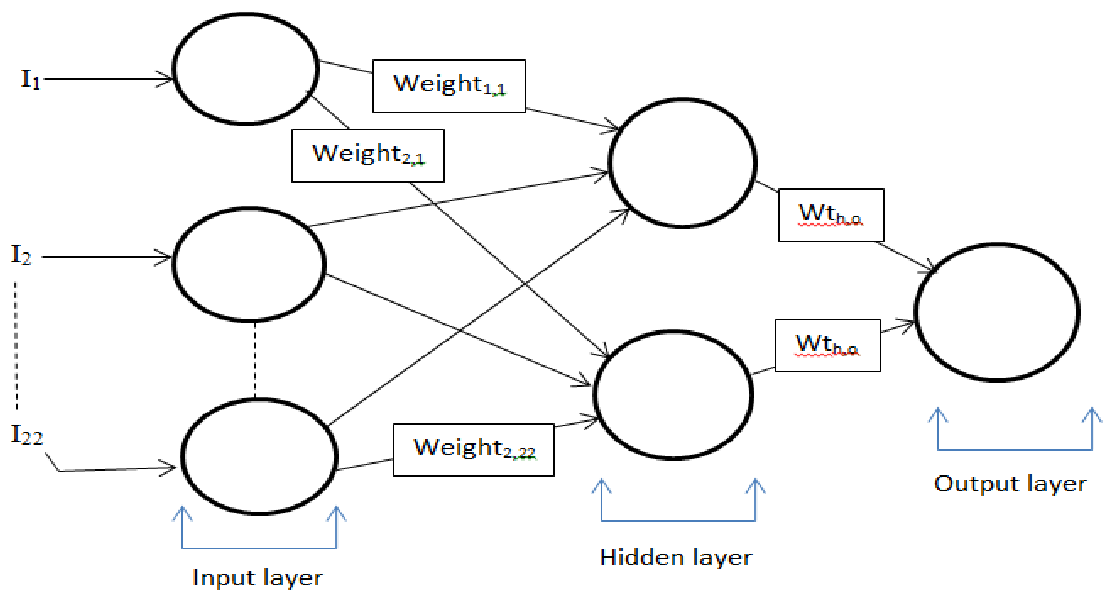


Figure 2.7: -Two-layer neural network.

Output is divided into three categories 1, 2 and 3.Here the mapping of the previous output range and present output range of the network is done as follows: - the output range from '1 to 4' is mapped as output '1', the output range from '4 to 5' is mapped to output '2' and the output range from '5 to 8' is mapped as output '3'. The output '1' of the network indicates that agile methodology is the best suitable development methodology for given project parameters, output '2' indicates that either agile or non-agile or a combination of both methodologies can be used for the particular project parameters. A '3' at the output indicates that non-agile methods are the best solution for the development of the project-in-hand.

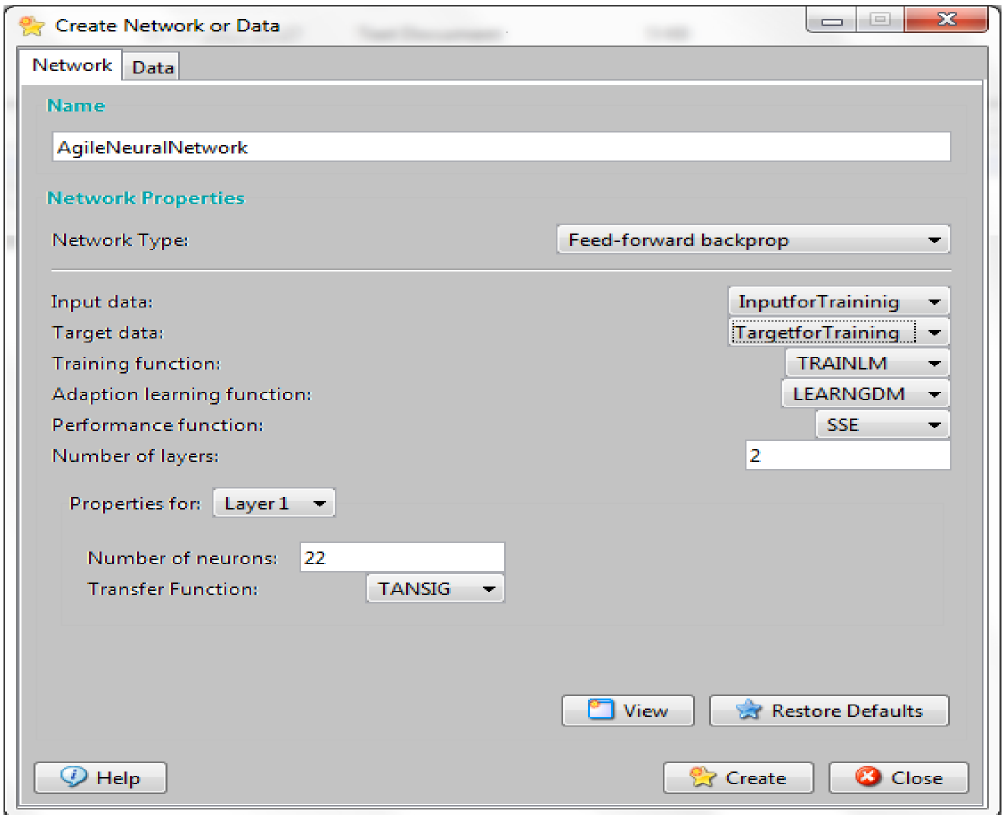
The ANN model

The model of ANN is specified by the three fundamental entities:

- The model's synaptic interconnections;
- The training or the learning rules adopted for updating and adjusting the connection weights;
- Their activation functions.

Connections or create networks: -An ANN consists of highly interconnected processing elements (neurons) such that for each processing element output is found to be connected through weights to other processing elements or to itself. The arrangement of neurons to form layers and the connection pattern formed within and between layers is called network architecture. There exist a number of neural network architectures for the research; the Feed-forward Back-propagation Neural Network was used. This is very popular neural network architecture because it can be useful in several different tasks. The first term, "feed-forward" describes the way patterns are processed and recalled by this neural network. Neurons are connected the only forward in a feed-forward neural network. There are connections from each layer of the neural network to the next layer (for instance, there are connections from the input to the hidden layer), but there are no such backward connections exist. The word "back-propagation" defines the way the neural network is trained. The form of training used by Back-propagation is supervised practice. In such a scenario, the network required sample inputs and estimated output to be provided. The estimated outputs being provided are then compared with the actual outputs for given set of input. Then the back-propagation algorithm for training takes a deliberate error using the estimated outputs, after which weights of various layers are adjusted backward i.e. from the output layer to the input layer.

For this problem network formed consists of twenty-two neurons at the input layer, two neurons in hidden layer and one neuron in the output layer. The next step is to train the network for self-adjusting the weights allotted for the connections


Create Network or Data

Network | Data

Name

AgileNeuralNetwork

Network Properties

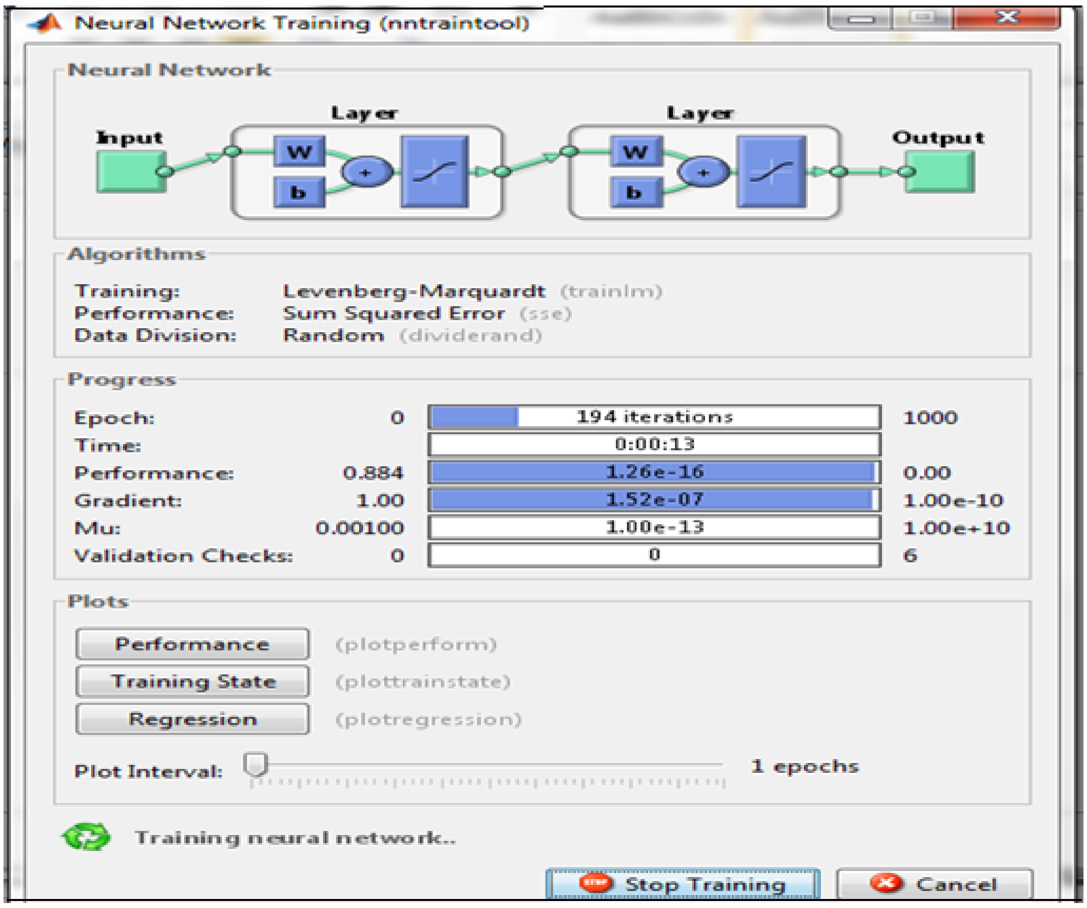
Network Type:	Feed-forward backprop
Input data:	InputforTraininig
Target data:	TargetforTraining
Training function:	TRAINLM
Adaption learning function:	LEARNGDM
Performance function:	SSE
Number of layers:	2

Properties for: Layer 1

Number of neurons:	22
Transfer Function:	TANSIG

Figure 2.8:- Screenshot for creating network

Train Network: - For training, the high number of inputs and their corresponding weights are provided, the data is generated for 4 projects- Mobile Application Development (MAD) project and for Air Traffic Control (ATC) project, ERP application in SMEs and Banking application details. Among these four the details of two case studies i.e. - Mobile Application Development and Air Traffic Control are given in sec 2.5.1 and 2.5.2 of this thesis. The epoch or a maximum number of iterations that the network can perform during training chosen here is 1000. Figure shows as


Neural Network Training (nntraintool)
Neural Network
Input
Layer
W
b
Layer
W
b
Output

Algorithms

Training: Levenberg-Marquardt (trainlm)
Performance: Sum Squared Error (sse)
Data Division: Random (dividerand)

Progress

```
Epoch:                 0       194 iterations      1000
Time:                          0:00:13
Performance:           0.884   1.26e-16            0.00
Gradient:              1.00    1.52e-07            1.00e-10
Mu:                    0.00100 1.00e-13            1.00e+10
Validation Checks:     0       0                   6
```

Plots
Performance (plotperform)
Training State (plottrainstate)
Regression (plotregression)
Plot Interval: 1 epochs
Training neural network..
Stop Training
Cancel

Figure 2.9:- Screenshot for training the network

The Activation functions in a neural network: -The activation function in a neural network species the output of a neuron to a given input. Neurons have switched that output a ‘1’ when they are sufficiently activated and a ‘0’ when not. There are a number of common activation functions in use with neural networks. For our research, the activation function used is a tangent sigmoid function. The equation for this function is tang (n) = 2 / (1 + EXP (-2*n)) - 1. The final adjusted weights by neural networks are given in the table below:-

Table 2.19: Final weight adjusted by neural network

	Metrics	Weight (W1)	Weight (W2)	Weight (W3)
1.	Volatility of requirements	0.19376	0.89165	-0.40272
2.	Complexity	0.51028	-0.030383	-0.68257
3.	Add-on function	-0.29233	-0.099261	0.067508
4.	Necessary Function	0.58891	-1.0546	0.41018
5.	Flexibility	0.30301	0.29065	0.3696
6.	Modularisation of task	-0.48088	-0.042034	0.42978
7.	Time to Market	0.17812	0.41011	0.44175
8.	Amount of requirement known initially	-0.3488	0.21002	0.30634
9.	Clarity & Completeness of Requirements	0.29833	0.4106	0.2556
10.	Expandability	-0.14357	0.27188	0.1635
11.	Coupling	0.33175	-0.90402	-0.070217
12.	Business Risk	-0.57239	-0.60949	0.597
13.	Technical Risk	0.31599	-0.38916	0.38757
14.	Operational Risk	-0.34957	-0.35567	-0.29739
15.	Programmer’s Capability	0.3903	-0.11019	-0.20646
16.	Application Experience	0.059763	0.21941	0.23257
17.	Reuse of Existing Code	0.20259	0.4015	-0.44519
18.	Develop for Reuse	0.29662	0.31654	-0.60989
19.	Platform Volatility	-0.27396	-0.035285	0.15748
20.	Platform Experience	-0.27396	-0.035285	0.15748
21.	Tool Experience	0.36862	-0.1564	-0.29208
22.	Team Cohesion	0.31846	0.26222	0.2154

Since the network architecture used is feed-forward back propagation neural network, it adjusts the weights of the various layers from the output layer to the input layer. Final weight of the network after training and adjustments are:-

Bias to hidden layer neuron: - [-1.3528; -0.40062; -1.2345]

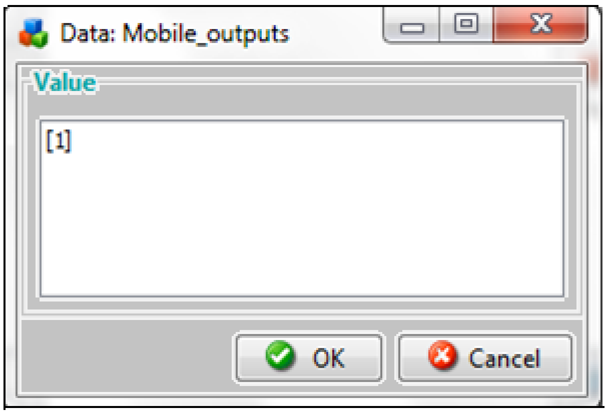
Weight from hidden layer to output layer: - [0.69954 -1.4034 -0.88563]

Bias to output layer neuron: - [0.11863].

Case 1:- Input data set for the case project - Mobile Application Development to the neural network tool

Input Data set:- {3;1;3;3;5;5;5;3;5;5;1;8;1;2;3;3;3;3;2;2;3;3}

Output Screen:-



[1]

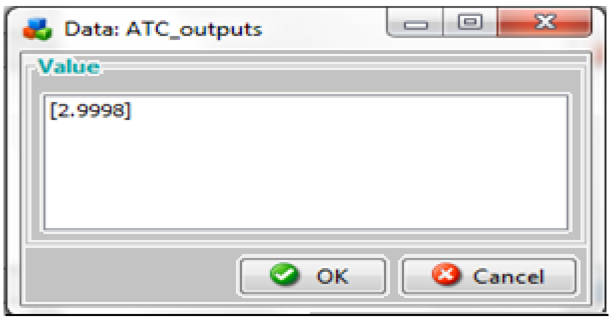
Figure 2.10:- Output Screen for MAD

Target:-‘1’ (i.e Agile). – For, Mobile Application Development

Case 2: - Input data set for the case project-Air Traffic Controller to the neural network tool.

Input Data Set:- 1;8;2;8;2;1;2;8;8;2;5;2;8;8;8;2;1;5;2;8;8;8

Output Screen:-



[2.9998]

Figure 2.11:- Output Screen for ATC

Target:-3 (i.e Traditional) – For, Air Traffic Control.

Summary

In this chapter, a set of 22 project characteristics is defined that needs to be assessed to achieve the goal of- Deciding the appropriate software development methodology for the situation-in-hand. Weights are assigned to these characteristics to identify their impact on the software development methodology. Further, it was observed that all project characteristics cannot be measured on the same scale; metrics is designed for deciding the input criteria for these project characteristics. Initially, the complex task of weight distribution has been done manually. In the later stages, neural networks are used for weight distribution; neural networks made the prediction process more accurate and simplified the task of the developer to a great extent.

The proposed technique is helpful for the organizations to save on huge losses incurred by the failure of projects due to the wrong selection of software development methodology.

The next chapter explains the method configuration process for traditional methods. The chapter explores the issues that need to be addressed to configure methods and how these issues got solved to form a project-specific method.

Chapter 3

Method Configuration process-Traditional methods

The chapter presents a method configuration process for traditional methods in Information System Domain (ISD). The proposal relies on *configurable methods* of different granularity; this is in contrast to the previous approaches (Karlsson and Ågerfalk, 04; Wistrand and Karlsson, 04) that support one single '*Base method*' for the process. The configurable methods are the pre-made method configurations and are an efficient way to achieve *genericity and granularity*.

All the Meta concepts of 'configurable methods' are supported by *Configurable metamodel* that is specially designed to suppress details during the Method Configuration process and to emphasize the task of constructing the project-specific method.

The main contribution of the chapter is to solve following issues

1. ***First Issue***- To design a *Meta model* to be used to model the concepts of configured method.
2. ***Second Issue***- The second issue is what '*good component*' is and what the '*right granularity*' is.
3. ***Third Issue***- Is regarding the '*Selection of a configurable method component'* for the project –in-hand.
4. ***Fourth Issue***-Lastly, the *complete process of a configuration* to reach coherent desired method.

Solution Approach

The present research approaches the first issue by modifying the Decisional Metamodel of (Prakash, 97) to reach *Configurable Metamodel* capable enough to model the concepts of the configurable method. The Decisional metamodel is an instantiation of the Generic model (Prakash, 06). Therefore, the proposed metamodel is an instantiation of the generic model.

Secondly the thesis presents configurable method component are entire methods, whether *atomic or compound.* These configurable method components will have characteristics of *commonality and variability*. This is also the '*right granularity*' because such method provides the most fundamental, configurable definitions. Coming to the third issue, that of selection of method component ensuring appropriateness of the retrieved component for the current situation. As the method base in the proposed approach contains configurable methods, the retrieval operation will perform on *global properties of methods.* Since the retrieved components shall satisfy global method properties, the chance of retrieving relevant components becomes high. The retrieved component is then configured to form the situated method.

The chapter first describes the configurable Meta model. In Section 3.2 gives the atomic and compound configurable method construct. Section 3.3 defines the global properties of method and defines the storage and retrieval operations performed on the method base. In section 3.4, architecture of method configuration process is explained.

3.1 Configurable Meta Model

The Configurable Metamodel is obtained by modifying Decisional metamodel. The decisional Meta model has a generic model part that treats a method as a triplet < MB, Dep, E> where, MB is a set of method blocks, Dep is a set of dependencies between these, and E is the enactment algorithm. The Configurable metamodel introduces *commonality and variability* concepts as shown in Fig. 3.1. It is centered round Method blocks and Dependencies. 'E' is the procedure that exploits the given set of MB and Dep to produce the product. It cannot be configured but comes as a given with the Metamodel. The set 'Dep' establishes dependencies between instances of method blocks. Thus, if a method block is *common* then all dependencies in which it participates are relevant to the configured method. However, if a method block is a *variable* and not included in the configured method, then all dependencies in which these variants participate are meaningless. Since 'Dep' is configured by the act of inclusion/exclusion of method blocks, it is not to be directly configured by the method engineer and is treated as not configurable in the Metamodel.

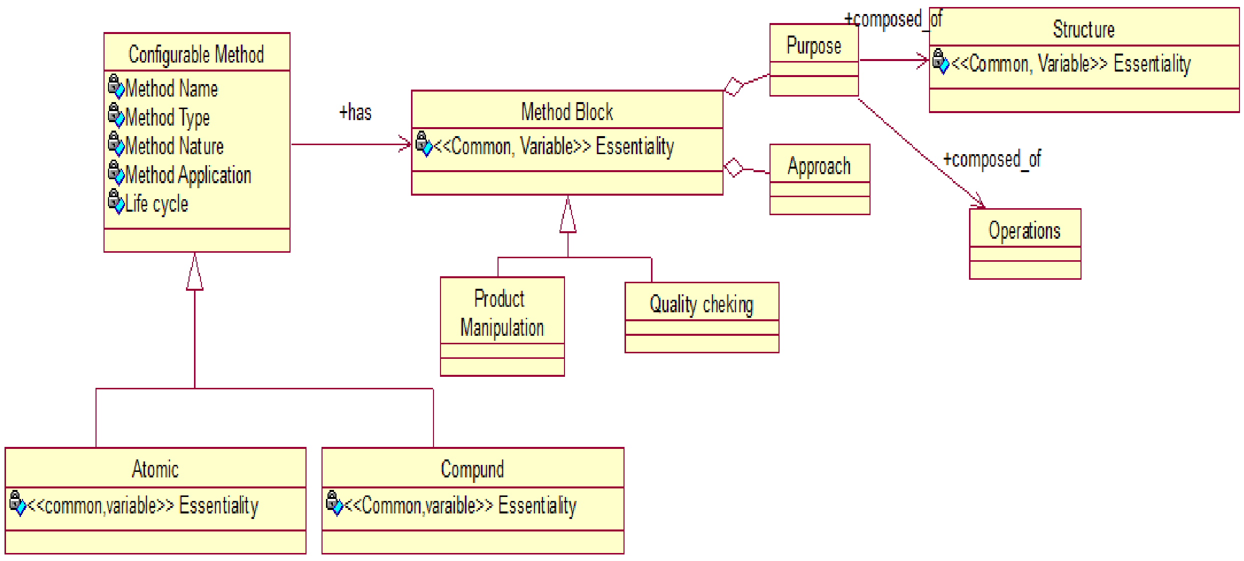


Figure 3.1:- Configurable Metamodel

Fig. 3.1 shows the presence of an attribute called essentiality. Essentiality = *common* specifies commonality whereas Essentiality = *variable* specifies a variant. The Configurable metamodel shows that method may be- *common* or *variable*. This has particular relevance for compound methods, for example, UML which is compound method consisting is a unification of atomic methods:

< Use Case Diagram, Activity Diagram, Class Diagram, Sequence Diagram, Collaboration Diagram, State Chart Diagram, Deployment Diagram, Component Diagram>.

It is possible to declare Use Case Diagram (UCD) and Class Diagram (CD) as a *common* and another component as a *variable*. Any method configured from this shall necessarily have a UCD and CD components whereas the others are optional. In contrast, an atomic configurable method can only be common.

Within a method, it is possible for *method blocks* to be either *common or variable*. This is shown by the essentiality attribute of the concept method block in the configurable metamodel. Thus in the foregoing example, the *common* class diagram can have its individual concepts as *common or variable*. For example, we may define an *object class* as *common* but an *operation* of the class as a *variable*. Similarly, in Use Case Diagram, we can define an *actor*, *use case* as *common* whereas, *generalization* can be *variable*.

Since, the concept of the configurable metamodel is an instantiation of the generic model proposed in (Prakash, 06). The instantiation relevant to the purpose is shown in Table 3.1.

Table 3.1:- Instantiation of Decisional Metamodel

Generic Model Concept	**Decisional Meta Model Concept**
Method block	Decision
Objective	Purpose
Product type	Structure
Process type	Operation

A method block is an aggregate of **Purpose and Approach**. For simplicity, let us ignore the notion of an approach. Thus, a method block reduces to a purpose. Now, in a purpose, there is a structure part and an operation part.

Purpose= <structure, operation>

The ‘operation part’ is used to *create, delete and modify* the ‘structure part’ and is given in the Metamodel. Thus, they are not configurable. The only configurability lies in the ‘structure part’. This results in the ‘purpose’ and consequently, the ‘method block’ to be configurable.

Again, however, this configurability can be algorithmically determined, only that subset of purposes is included in the situated method which is built on the included concept structures. Thus, there is no need for the method engineer to do explicitly this configuration.

In the rest of this section, the Configurable metamodel is described in detail.

3.1.1 Structure

There are two kinds of structures, those whose instances can be created and destroyed by application engineers and those whose instances are pre-defined. The former are called conceptual structures, and the latter is called fixed structures. Conceptual structures constitute the set of concepts in terms of which a product is expressed. Fixed structures are those that are defined once, by a method engineer. The fixed structures are method constraint such as *completeness and conformity* which cannot be created or destroyed by the application engineer.

1. Conceptual Structures

Conceptual structures are partitioned into two dimensions. The first dimension classifies them as either *atomic or compound*. The second dimension represents conceptual structures into disjoint classes of structures called *constraint, definitional, constructional, link, and collection of concepts* respectively.

Simple constructional structures cannot be decomposed into other components. *Links* are conceptual structures that are used to build collections of concepts from given concepts. For example, ISA and aggregation are links, as they develop abstraction hierarchies. *Collections* *of concepts* are constructed whenever constructional structures are connected by links. Aggregations, specialization hierarchies, and subtype hierarchies are examples of a collection of concepts. A collection of the concept is complex if it is defined out of other collections. *Definitional* structures determine the properties of conceptual structures. *Constraints* impose application-related constraints on conceptual structures. The presence of the attribute, *essentiality*, in configurable Metamodel shows that conceptual structures are configurable.

2. Fixed Structures

Fixed structures deal with the constraints that are used to enforce quality of conceptual structures. They are defined by the method engineer to help the application engineer in

Creating *well-defined and well-formed* conceptual structures. In their simplest form, they are the method constraints of *completeness, consistency, conformity, and fidelity*. For example, a *relationship* is complete provide the *entity* class is associated with it, or a conformity constraint can be there on the *arity of relationships*: every relationship must be *binary*. Similarly, there are *compositional constraints* which are specified between conceptual structures of the different atomic methods of a compound method. A structure of one of these cannot compose any arbitrary structure of the other. Such composition is governed by constraints that control the product resulting from the use of compound methods. The method engineer defines these constraints at the time the compound method is described. For example, in UML, *function* in Class Diagram must be a *use case* in Use Case Diagram.

3.1.2 The Operation

'Operations' identify the set of process types that operate on product types to provide product manipulation and verification capability to application engineers. Operations are classified into two four classes as follows:

1. ***Basic Life Cycle:*** For each conceptual structure, there are operations to *create,* and *delete* it.
2. ***Relational:*** These allow different structures to be related to one another. *These are attached, join,* couple, *associate,* relate, *apply* and their inverses.
3. ***Integration, This class of operations,*** is defined for compound methods. These operations are *export, import, correspond, convert* and their inverse operations.
4. ***Constraint Enforcement:*** For each conceptual structure, of a method and the method constraint applicable to it, a *method constraint enforcement* operation is defined.

3.1.3 Purposes

As mentioned above, a purpose is defined as:-

<S, O>

Where 'S', is a non-empty set of conceptual structures and 'O' is an operation. Purposes can be *primitive, complex and abstract*.

1. *Primitive* – These are elementary and non-decomposable. The primitive purposes are *the basic life cycle, relational, integration, and method constraint enforcement purposes*.
2. *Complex* - These are composed out of other purposes and represent aggregated objectives like *split a relationship* and *convert entity into an attribute*.
3. *Abstract* - These are formed when common properties of purposes are abstracted out into higher-level purposes like *validate the schema* or *improve the object class*.

3.1.4 Dependencies

Method concepts (MC_i) in a method are dependent upon one another. In the generic model (Prakash, 06), these dependencies are defined on two main properties namely, **Urgency** and

Necessity (As shown in table 3.2). Urgency refers to the **time** at which the dependent method concept, MC_2, is to be enacted. If MC_2 is to be enacted **immediately** after MC_1 is enacted then this attribute takes on the value *Immediate*. If MC_2 can be enacted any time, **immediately or at any later moment**, after MC_1 has been enacted, then urgency takes on the value *Deferred*. Necessity refers to whether or not the dependent method concept MC_2 is necessary to be enacted after MC_1 has been enacted. If it is **necessary** to enact MC_2, then this attribute takes the value *Must* otherwise it has the value *Can*. Combining these two properties, four possibilities as shown in Table 3.2.

Table 3.2:- Types of Dependencies

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

In configurable Metamodel, four kinds of dependencies are defined to the other dependencies of the generic model.

1. *Requirement dependency*: Requirement dependency says that when a particular manipulation purpose is performed, there must associate some constraints that have to be related to it. This corresponds to dependency *type '3'* of the generic model.
2. *Removal Dependency*: removal dependency is the inverse of requirement dependency. It says that when a particular manipulation purpose is performed, then certain purposes are not to be performed. This corresponds to dependency *type '1'* of the generic model.
3. *Activate dependency*: It says that a purpose activates another purpose. The activate dependency is of *type '4'* of the generic model.

4. *Inactivate dependency*: Inactivate dependency is the inverse of the activate dependency. It says that when an individual manipulation purpose is performed, then certain manipulation purposes *cannot* be performed. , inactivate dependency is of *'type 1'*.

3.2 Configurable Method

As stated above, the Configurable Meta-model can model any method as *Configurable Method*. The configurable method is defined as "*An abstraction of a method that identifies the essentiality of the method concepts and its relationships*". The crucial part of this definition is the '*Essentiality of the method concepts'*. For the process, this is accomplished by using Conceptual structure knowledge (see sec. 3.1.1). Conceptual structures are classified in seven categories- simple definitional, complex definitional, simple constructional, complex constructional, simple collection of concepts, complex collection of concepts and links. They provide a set of guidelines for identifying the *Essentiality* in a method. Guidelines are:

Guideline 1: The S*imple definitional* and *Simple constructions* are essential building blocks of a method; hence considered them as *common* for all methods.

Guideline 2: *Complex definitional* and *Complex constructional* may also be considered as common for some projects. The *commonality* in them are project-specific, Method engineer needs to use his knowledge for deciding the *commonality* and *variability* in them.

Guideline 3: The rest of the conceptual structures are considered as *Variables*.

Earlier, there were not any explicit rules exists for the identifying *commonality* and *variability* in the methods. These simple guidelines facilitate the task of the Method Engineer to a great extent and allow him to concentrate on configuring project-specific methods.

The primary process of defining configurable methods is as follows:

1. Define the scope of the configurable method by identifying its method concepts.

2. If the method is compound, then define the essentiality property of each method component; else determine its essentiality as *common*.
3. For every method concept in an atomic method, define the essentiality property using above guidelines.

This process is top down in the sense, that first *essentiality* is established for the method and then proceeds down to determine *essentiality* of coarse grained concepts. Section 3.2.1 and 3.2.2 presents an atomic and compound configurable method respectively.

3.2.1 Atomic Configurable Method

Entity- Relationship(ER) method expressed in configurable Meta model (Atomic Method)

A. Method Nature Part: Describes the *method name* and *method characteristics*

Method Name (*12 characters*) <***ER method***>

Method Type (*Atomic/Compound*) <***Atomic***>

Method Nature (*Constructional/Transformational*) <***Constructional***>

Method Application (*Data/Process/Behaviour Oriented*) <***Data Oriented***>

Method life cycle (Requirement/Design/Testing/Complete life cycle) <***Design Phase***>

B. Method Conceptual Model: Method conceptual model stores the method concepts, instantiation of conceptual structures and essentiality of each method concept in a method. Since atomic method belongs to one product model only, there is single conceptual model for each atomic method.

<Concept Name> : <Type> <Essentiality>

<Entity>: <Simple Constructional> <Common >

<Relationship>: <Simple Constructional> <Common>

<Cardinality>: <Simple Definitional> <Common>

<Primary Key>: <Complex Definitional> <Common>

<Attribute> : <Complex Definitional> <Common>

<Multiplicity of Attribute>: <Complex Definitional><Variable>

<Role>: <Simple Definitional> <Common>

<Functionality>: <Simple Definitional> <Common>

<N-ary Relationships>: <Complex Constructional> <Variable>

C. **Method Purposes and dependencies:** Along with the method nature part and method conceptual part, purposes and dependencies in the method are also stored.

<Purpose>: <*Basic life cycle, Relational, Constraint Enforcement*>

<Dependencies>: <*Activate, Requirement, Inactivate, Removal*>

3.2.2 Compound Configurable Method

Unified Modelling Language expressed in configurable Meta model (Compound Method)

A. Method Nature Part: Describes method name and method characteristics of the method. In compound methods, method nature part also defines the method components within the method

Method Name (*12 characters*) <***UML method***>

Method Type (*Atomic/Compound*) <***Compound***>

Method Components < *Class Diagram, Use Case Diagram, Sequence Diagram, Collaboration Diagram, State Chart Diagram, Component Diagram, Deployment diagram, activity diagram, object diagram>*

Method Nature (*Constructional/Transformational*) <***Transformational***>

Method Application (*Data/Process/Behaviour oriented*) <***Process Oriented***>

Method life cycle (*Requirement/Design/Testing/Complete life cycle*)<***Design Phase***>

B. **Method Component Model:** Since compound methods consist of more than one atomic method, they need a separate component model to be well expressed. Method component model of compound methods, stores the constituent atomic methods together with their essentiality.

<Method Component Name>:	**<Essentiality>**
< Class Diagram>:	*< Common>*
< Use Case Diagram>:	*<Common>*
<Sequence Diagram>:	*<Variable>*
< Collaboration Diagram>:	*< Variable>*
< State Chart Diagram> :	*< Variable>*
<Component Diagram>:	*<Variable>*
<Deployment diagram>:	*<Variable>*
<Activity diagram>:	*<Variable>*

C. **Method Conceptual Model(s): -** Since compound methods composed of atomic methods, there is a separate conceptual model for each atomic method defined in the compound method. Following is the method conceptual model for method Class Diagram.

<Method Concept Name>	**<Type>**	**<Essentiality>**
<Class>	< Simple Constructional>	< Common>
<Data_type>	< Simple Definitional>	<Common>
<Association>	<Complex Constructional >	<Variable>
<Aggregation>	<Simple Collection of concepts >	< Variable>
<Operation>	< Complex Definitional>	< Variable>
<Generalization>	<Simple Collection of Concepts >	< Variable>
<Generalization_link>	< Link>	<Variable>

<Aggregation_link>	<Link >	< Variable>
<Cardinality>	< Simple Definitional>	< Common>
<Degree of association>	<Complex Definitional >	< Variable>
<Multiplicity>	< Complex Definitional>	< Variable>

UML has eight atomic methods in the method component model consequently eight conceptual models are needed to express complete UML as a configurable method.

D. **Method Purposes and dependencies.** Purposes and dependencies for each conceptual model are defined. In case of compound methods, together with the operations available in the atomic method, a separate class of operations is also defined. The class is named as *Integration class* and it deals with the structure belonging to different product models of the compound method.

<Purpose>: <*Basic life cycle, Relational, Integration class, Constraint Enforcement* >

<**Dependencies**>: <*Activate, Requirement, Inactivate, Removal*>

3.3 Retrieval of Methods

The configurable methods will be selected to form project-specific method, based on the **global properties of the methods**. Global properties of methods provide a broad indication of the family of methods that can be produced. These properties and are proposed in (Prakash and Goyal 2007) are as follows:

3.3.1 Global Properties of Method

1. **Method Nature-** A method can be *data oriented, process oriented or behavior oriented***.** **Data oriented methods** emphasize the complete and thorough analysis of data and its relationships. Examples of data oriented methods are ER (Chen, 76) and Natural-language Information Analysis Method-NIAM (Verheijen and Bekkum, 82).

Process-oriented methods place emphasis on activities of an application domain, their interrelationships and decompositions (Ross and Schoman, 77; Lundeberg et al.,81). Examples of process-oriented methods are SSADM (Goodland and Karel, 1999), SADT (Marca and McGowan, 87), JSD (Jackson, 82) and ISAC (Lundeburg et al., 79). **Behavior-oriented methods** focus on the dynamic nature of the data by analyzing and understanding the events in the real world which impact data recorded in the IS. Examples of such methods are: - REMORA (Rolland and Richard, 82), TAXIS (Mylopoulos et al.,1980) and OBCM (Tao et al., 2006).

2. **The life cycle of method-** This address the part of the software development life cycle catered by the Method, for example, ER method is used to elicit requirements, Data flow diagrams are used for designing and there are methods like- Unified Modelling Language that used to model the complete life cycle.
3. **Method type-** Methods are classified as *atomic and compound methods*. Atomic methods are those that are expressed in exactly one product model. For ex. Entity-Relationship method. On the other hand, a compound method consists of other atomic methods integrated to form a compound method. For example Unified Modeling Language.
4. **Method Application-** Methods can be *Transformational or Constructional* - A transformational method is used for transforming a product, expressed in one or more product models, into a product of another product model(s). Whereas, constructional method is used whenever a new product is to be constructed.

3.3.2 Method Base

Method base is probably the most important prerequisite for the method configuration process. It is a formal representation of how a configurable and configured method component is stored. The construction of a method base is a vital activity as it presents the foundation for creating desired methods and thus has to be done prior to the method configuration process.

The design of the method base is shown in fig.3.2. It is divided into two parts *Method Configurable Part (MCP) and Configured Method Part (CMP).* Configurable methods are stored in Method Configuration Part from this part 'list of suitable methods' are retrieved and 'most appropriate' method is selected.

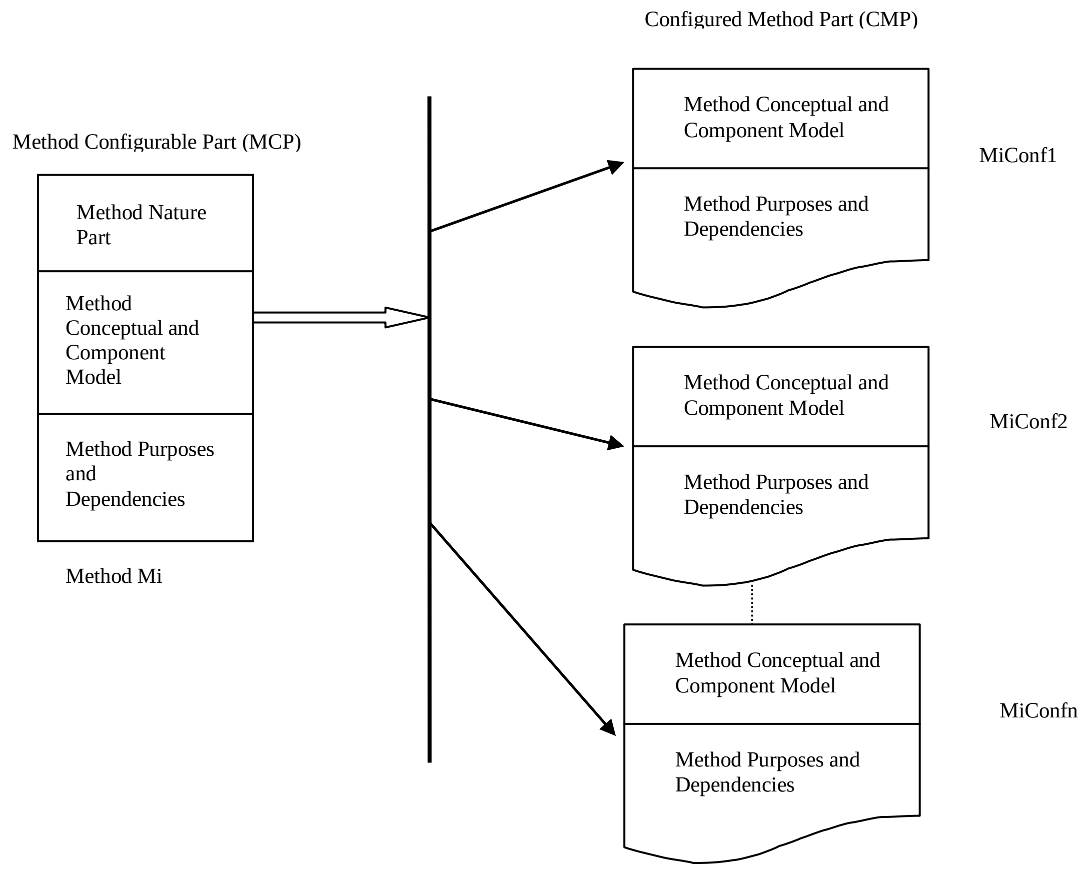

Configured Method Part (CMP)
Method Configurable Part (MCP)
Method Nature Part
Method Conceptual and Component Model
Method Purposes and Dependencies
Method Mi
Method Conceptual and Component Model
Method Purposes and Dependencies
MiConf1
Method Conceptual and Component Model
Method Purposes and Dependencies
MiConf2
Method Conceptual and Component Model
Method Purposes and Dependencies
MiConfn

Figure 3.2:- Design of the method base.

The *Method Configurable Part* of method base stores both *atomic configurable method* and *compound configurable method.* The atomic configurable model and compound configurable model is described in sec 3.2.1 and 3.2.2. Here, the focus is on Configured Method Part. The *Configured Method Part* stores the *configured methods* formed after method configuration process. These configured methods can be further retrieved for future use.

1. *Configured Method Part for Atomic methods*

Following is the configured method formed (*ERconf)* from ER method having only *Single valued attribute and Binary relationships.* To form ERconf, the product entities <n-ary relationship> and <multivalued attribute> will remove the Method conceptual Model of ER and the purposes and dependencies are modified accordingly. *ERconf* is stored in Configured Method Part as

ERconf stored in Configured Method Part

A. **Method Conceptual Model**

<Entity>: <Common>

<Relationship> :< Common>

<Cardinality>: <Common>

<Primary Key>: <Common>

<Attribute> : <Common>

<Role>: <Common>

<Functionality>: <Common>

B. **Method Purposes and dependencies:** In the configured method, some method concepts have been neglected and are not becoming the part of the desired method. Consequently, purposes and dependencies of the original conceptual model shall be engineered to form the coherent method. The method formed is then stored in the Configured Method Part of the method base. Engineering process for atomic methods will be discussed in (sec. 3.4).

2. *Configured Method Part for Compound methods*

Following is the configured method formed *UMLconf* from UML method having only two atomic methods *Class Diagram and Use Case Diagram*. *UMLconf* is stored in Configured Method Part as

A. Method Component Model

< *Class Diagram*>

< *Use Case Diagram*>

B. Method Conceptual Model(s)

<*Conceptual Model of class diagram*>

<*Conceptual Model of Use Case Diagram*>

C. Method Purposes and dependencies.

In the configured method, some method components have been neglected and are not becoming the part of the desired method. Consequently, purposes and dependencies of the original component model shall be engineered to form the coherent method. The method formed is then stored in the CMP of the method base. Engineering process for compound methods will be discussed in (sec. 3.4).

The configured method is an implementation of configurable method stored in the MCP of the method base; *the method nature part* of configured method formed remains same as of configurable method. Thus, the method structure of the configured method stored in Configured Method Part comprises of modified conceptual and component model along with the modified set of purposes and dependencies.

Retrieval from the method base

Configurable method components are retrieved from the method base, by mapping project characteristics with the method characteristics. In this thesis, method characteristics are defined as global properties of the methods as described in. sec 3.3.1**.** Table 3.3 shows the mapping between the methods and method characteristics.

Table 3.3:- Mapping between methods and global properties

Global Properties		**Methods**								
		M1	**M2**	**M3**	**M4**	**M5**	**M6**	**M7**	**M8**	**M9**
Method Nature (MN)	Data Oriented	✓				✓		✓		
	Process oriented		✓		✓				✓	
	Behaviour Oriented			✓			✓			✓
Life cycle of method (MLF)	Requirement phase	✓				✓		✓		
	Design phase		✓		✓				✓	
	Complete lifecycle phase		✓				✓			✓
Method type (MTP)	Atomic	✓				✓		✓		
	Compound		✓		✓				✓	
Method Application (MAP)	Constructional	✓				✓		✓		
	Transformational		✓		✓				✓	

A number of operations can be performed on the method base, for example,

Operations on Method Base

The basic operations on method base can be formalized by defining operators: the *storage* operator for storing the configurable methods in the method base, *storeconf* operator used to store configured method in the method base, the *retrieve* operator used to retrieve the list of 'eligible methods' and *select* operator for the selection of 'most appropriate' method.

- **Store a new configurable method:** Initially, the method base is populated with the new configurable method is in the Method Configurable Part of the method base using 'store' operator. The new method stored is completely defined by *method nature part, conceptual and component model and purposes and dependencies* of the method.

  Store Mi <MN, MLF, MTP, MAP>.

  For example: Store M1 <data oriented, requirement phase, atomic, constructional>.

  Stores the method M1 in Method Configurable Part with method characteristics as MN=*Data Oriented*, MLF=*Requirement phase*, MTP= *Atomic* and MAP =*Constructional.*

- **Store a configured method:** After the method configuration process, (see sec. 3.4) the configured method formed will, store in the Configured Method Part of the method base.

  Store_conf (Miconf) = CMP of method Base.

  For example: Store_conf <M1conf>.

  Store method <M1conf> in the Configured Method Part.

- **Retrieve configurable method:** The 'retrieve' operation mapped the attributes given, with the method characteristics of the configurable methods stored in the Method Nature Part. It retrieves the list of methods that satisfies these attributes.

  Retrieve M_{list} <MN= '', MLF= '', MTP= '' and MAP= ''>.

  For example: Retrieve M_{list} < MN= 'Data oriented', MLF= 'Requirement phase', MTP= 'Atomic' and MAP= 'Constructional'.

  Gives a list of atomic methods that are constructional and data oriented and are designed for requirement engineering phase.

- **Select a method:** The 'Select' operation selects the configurable model of the 'most appropriate' method chosen by the method engineer from the list of eligible methods retrieved from the method base through retrieve operation.

Select <Mi from M_{list}>.

For example: Select Method M1 from <M1, M5, M7>

Select the Method M1 from the list of eligible methods <M1, M5, M7>

- **Select the conceptual models of the compound method:** This operation will select all the conceptual models of the specified compound method.

  Select <M_{con_model}> where <MTP = 'Compound' and MLF = ''>.

  For example: Select <M_{con_model}> where <MTP = 'Compound' and MLF = Complete_life_cycle'>

  Selects all the conceptual models of method M2.

- **Update:** Update is used to update any of the method record stored in the method base.

  Update <MN= '', MLF= '', MTP= '' MAP= ''> where Method name = ''.

  For example: Update <MN= 'process oriented', MLF= 'design phase', MTP= 'atomic' MAP= 'transformational'> where method name = 'M2'.

  Updates the record method M2.

- **Delete:** Delete the method record from the method base.

  Delete <Method Mi>.

  For example Delete <Method M2**>**

  Delete the record method M2.

3.4 Architecture of Method Configuration Process

The process of Method Configuration is depicted in figure 3.3.

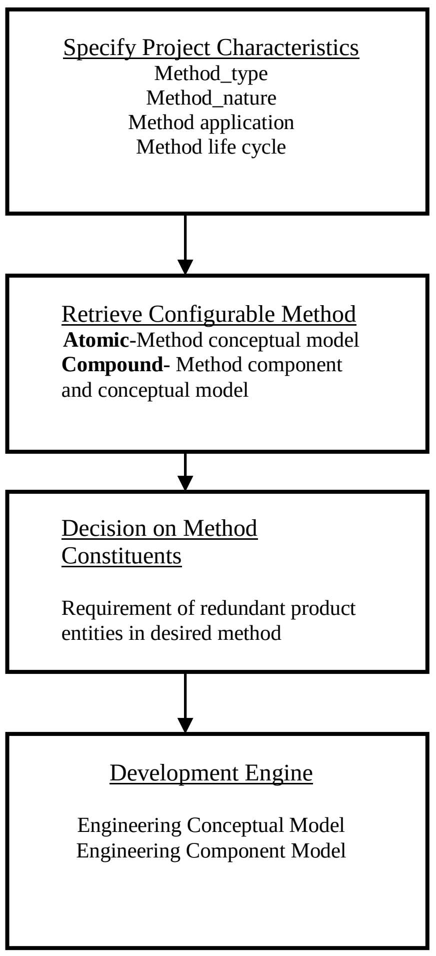

Specify Project Characteristics
Method_type
Method_nature
Method application
Method life cycle
Retrieve Configurable Method
Atomic-Method conceptual model
Compound- Method component and conceptual model
Decision on Method Constituents
Requirement of redundant product entities in desired method
Development Engine
Engineering Conceptual Model
Engineering Component Model

Figure 3.3:- Method Configuration architecture.

Step 1: Specify project characteristics

For developing a project, project characteristics are elicited. For example, for Air Traffic Control, the elicited project characteristics are:

Method_type = <Atomic>

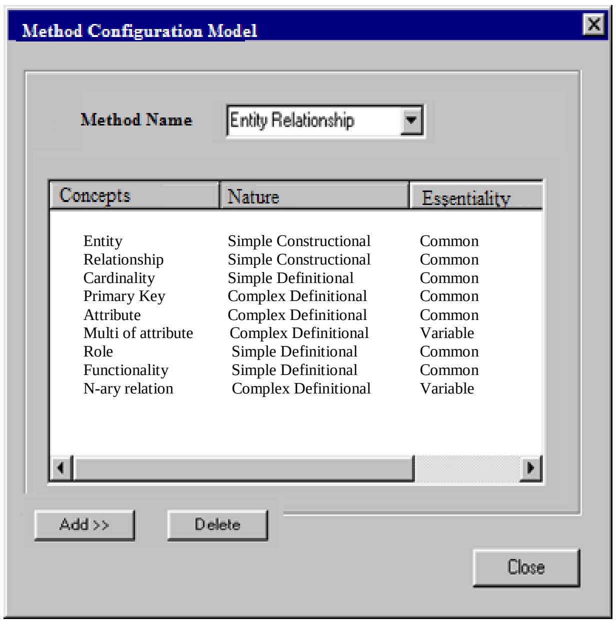
Method_nature = <Constructional>

Method_application =<Data-Oriented>

Method_life_cycle = <Design Phase>

Step 2: Retrieve Configurable method

The retrieval operation is done by mapping the project characteristics with method characteristics. Like, for the above problem ER method is selected from the list of methods retrieved from the method base. The method configurable model of ER is shown below

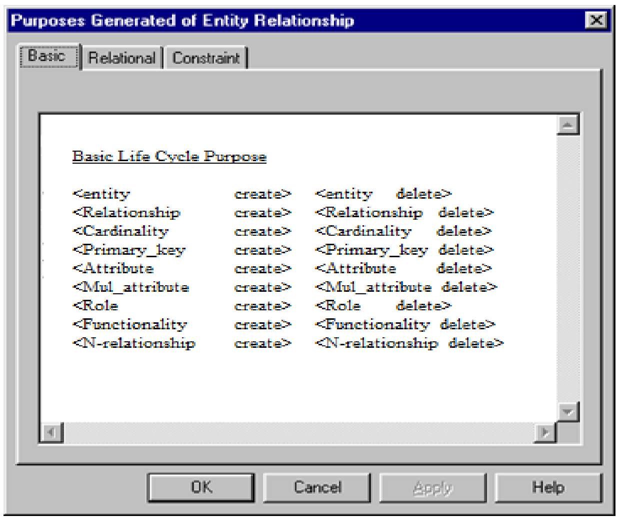


Method Name: Entity Relationship

Concepts	Nature	Essentiality
Entity	Simple Constructional	Common
Relationship	Simple Constructional	Common
Cardinality	Simple Definitional	Common
Primary Key	Complex Definitional	Common
Attribute	Complex Definitional	Common
Multi of attribute	Complex Definitional	Variable
Role	Simple Definitional	Common
Functionality	Simple Definitional	Common
N-ary relation	Complex Definitional	Variable

Figure 3.4:- ER configurable model.

Apart from above, purposes and dependencies for the configurable method are also generated. The basic life cycle purposes are shown below,



Basic Life Cycle Purpose

```
<entity              create>   <entity    delete>
<Relationship        create>   <Relationship  delete>
<Cardinality         create>   <Cardinality   delete>
<Primary_key         create>   <Primary_key  delete>
<Attribute           create>   <Attribute     delete>
<Mul_attribute       create>   <Mul_attribute  delete>
<Role                create>   <Role      delete>
<Functionality       create>   <Functionality  delete>
<N-relationship      create>   <N-relationship  delete>
```

Figure 3.5:- View of basic life cycle purposes

similarly, all purposes and dependencies are generated automatically.

Step 3: Decision on method constituents

Project requirements are complex in nature, and there cannot exist only one set of requirements, for every project. This simply reflects the change in the composition of the selected method. However, this negotiation of the composition is done in the method configuration process. The ER configurable model in sec 3.2.1 contains the list of the available set of common and variables for the method. A method derived from ER *must have* the notion of entity, relationship, cardinality, role and functionality. However, notion of a multiplicity of attributes and n-ary relationships are *variables*. The configuration process includes all the common concepts of ER but selects variable ones based on the need for the specific project being handled. Table 3.4 shows two ER configured methods obtained as instances of the configuration process.

Table 3.4:- Instances of ER configured method.

	Configured Methods from ER
1	All ER *common* concepts and *single valued attributes*.
2	All ER *common* concepts and *binary relationships*

Step 4: Engineering Conceptual/Component model

Since, some variable method concepts have been ignored and are not becoming a part of the desired method, the purposes and dependencies shall be modified to form the project-specific method. The chapter now presents algorithms that modify the purposes and dependencies for method configuration process.

The algorithm that supports the process is relatively simple. It starts with a method concept in the base method (typically it would be a simple definitional) and ends when there is no link left that would connect the deleted method concept further with any other method concept present in the configured method. If such links are found, they are examined for constraints they might have. When a particular link has no constraints or when constraints exist but are satisfied that the concept at the end of that link is processed in the same way using recursion.

<u>Engineering configured method from Atomic methods</u>

```
PROCEDURE engineering atomic method (cm, $s_i$)
// cm-*conceptual model*, $s_i$-*set of elements in cm*, $s_i$=<$C_i$ *or* $V_i$>
// $C_i$-*method concepts with essentiality as common*.
// $V_i$ -*method concepts with essentiality as variable.*
//Exclusion of a product entity $V_i$ requires the following deletion to be done in the chunk of purposes and dependencies generated at the time of method creation.
Begin:
Find all the purposes ($p_i$) where $V_i$ participates
For all pi
        *delete* < $V_i$, *forward purpose*> *and delete* < $V_i$, *inverse purpose*>
Find all fixed structure ($f_i$) where $V_i$ participates as a sub-concept or as a super concept
        *delete* < $V_i$, {*completeness, conformity and fidelity*}>
// when a project-specific method is created using the algorithm above, the dependency list should be checked for the completeness and coherency of the method formed.
For each deleted purpose $p_i$
Check
            {   Deletion of a purpose deactivates its inverse.
            Deletion of a purpose requires the deletion of all its method constraint and
            completeness purposes for all its super-concept as well as the method
            constraint fidelity purposes for all its sub-concept.
        }
For each deleted variable $V_i$
Check
        {
            If a product entity is deleted, all the purposes *activated* as a result of its
            creation become *inactivated*.
        }
END
```

Engineering configured method from Compound methods

The input to the process is the method component model that contains the set of method components along with essentiality property.

```
PROCEDURE engineering compound method (mc, $s_i$)
// mc-*component model*, si-*set of elements in mc*, $s_i$=< $C_i$ *or* $V_i$ >
// $C_i$ -*method components with essentiality as common.*
// $V_i$ -*method components with essentiality as variable.*
// Method components in a compound method communicate with each other through the
operations defined in the Integration class.
/*These operations are represented by a triplet, < $V_i$, $MC_i$, O> where, $V_i$ is the variable
method component to be excluded, $MC_i$ is the method component to which $V_i$ is
communicating, and O is the communicating operation. */
Begin:
```

```
//Disable all communication coming-In and Out from the deleted method component.
For each deleted variable $V_i$
Disable
        {< $V_i$, $MC_i$, export> and its inverse < $V_i$, $MC_i$, withdraw>
         < $V_i$, $MC_i$, import> and its inverse < $V_i$, $MC_i$, dump>
         < $V_i$, $MC_i$, correspond> and its inverse < $V_i$, $MC_i$, seperate>
         < $V_i$, $MC_i$, convert> and its inverse < $V_i$, $MC_i$, deconvert>
        }
END
```

Since method conceptual model exists in the next lower level of granularity of the method component model. Deletion of a method component automatically results in the exclusion of its method conceptual model.

3.5 Method Extension

The method configuration approach presented in this thesis consists of one basic process i.e. method configuration and one extended process i.e. method extension. The thesis proposes that the configuration process is the essential to the process and method extension should be attempted only when configurability fails to deliver the desired method. The failure can happen if a method concept is missing to make a perfect desired method or a method needs to extend its functionality to other phases of software development.

During method extension, the method engineer will either selects or integrates the missing product entities to form the desired method or assemble the method with another appropriate method to achieve the desired goal. For method extension, the external view of the method that represents only the functionality of the method is considered.

The new product entity must enter into the chunk of purposes i.e. Basic life cycle purpose, Relational purpose, Constraint Enforcement Purpose and Integration class purposes. Since the set of purposes gets modified the dependencies and the constraints, have to mutate accordingly. The details of the process will be discussed in chapter 5

Summary

Presently the methods used for software development do not project- specific and does not adapt to the requirements of given project. This issue has been raised for a long time leading to several proposed solutions to make the methods more situation specific and suiting to requirements of the project. The most of these solutions are proposed by method engineers after continued research. However, very few of them have been practically implemented in bits and pieces.

Method configuration in this thesis has been treated as a specific kind of method engineering. A Method Configuration process with respect to the instantiation of a Configurable Metamodel and Method configuration to the finest level of granularity has been envisaged in the present research. The fundamental part of the configurable meta-model is the configurable method as a means to facilitate efficient and rationally motivated modularization of systems development method. The benefits of using the configurable method are that the process can be performed more efficiently since pre-made method configurations are available and can be used over and over again. Hence, there is no need to perform a complete configuration for each new project.

As explained in chapter 2, the present research is focussing on two paradigms for software development- traditional methods and agile methods. The next chapter presents the method configuration process for agile methods. The process is further illustrated, with the help of two practical case studies.

Chapter 4

Configuring Agile Methods

In the last few years agile methodologies has generated a lot of interest among practitioners and lately also in the academia. (Qumer and Henderson-Sellers, 2006) points out that a considerable number of agile methods have been introduced to create "*people focussed, communications-oriented, flexible(ready to adapt to expected or unexpected change at any time), speedy (encourages rapid and iterative development of the product in small releases), lean (focuses on shortening timeframe and cost and on improved quality), responsive(reacts appropriately to expected and unexpected changes) and learning (focuses on improvement during and after product development)*" methods.

(Beck, 99) introduced the Extreme Programming method -better known as *XP* has widely acknowledged as the starting point of various agile software development approaches. There are also a number of other methods either invented or rediscovered since then, that appear to belong to the same family of methods. Such methods or methodologies are *Scrum* (Schwaber and Beedle, 2002), *Feature Driven Development (*Coad et al., 2000)*, Crystal methods* (Cockburn, 2000) *and DSDM (*DSDM consoritium, 97) etc. These methods have a well-defined structure that includes process, practices, roles and responsibilities.

- *Process*-Description of phases in the product-life-cycle.
- *Practices*-They are concrete activities and work products that a method defines to be used in the process.
- *Roles and responsibilities*- Allocation of specific roles through which the software production in a development team is carried out.

Agile methods are gaining popularity and are welcomed by managers and developers, now a day's software companies are extensively using these methods. (Miller and Lee, 2001) describes the characteristic of agile software process as – *"modularity, iterative with short cycles, time-bound-within cycles, adaptive with possible new risks, incremental process approach, people-oriented and collaborative working style"*. These characteristics ensure the fast delivery of software projects within given time-span.

In previous years many situational models have been developed, but no model has proved to be successful at effectively deliver the tailored/configured light-weight methods fulfilling the organisation requirements**.** Research in the field of agile methodology is growing. There are many published articles on various aspects of these, but probably due to they are being seen more as a practical approach rather than an academic methodology, most researches focus on experiences of using these methodologies in industrial domains and empirical findings on its practices.

This chapter presents an **Agile Method Engineering** process, to form project specific method. The framework is given below:

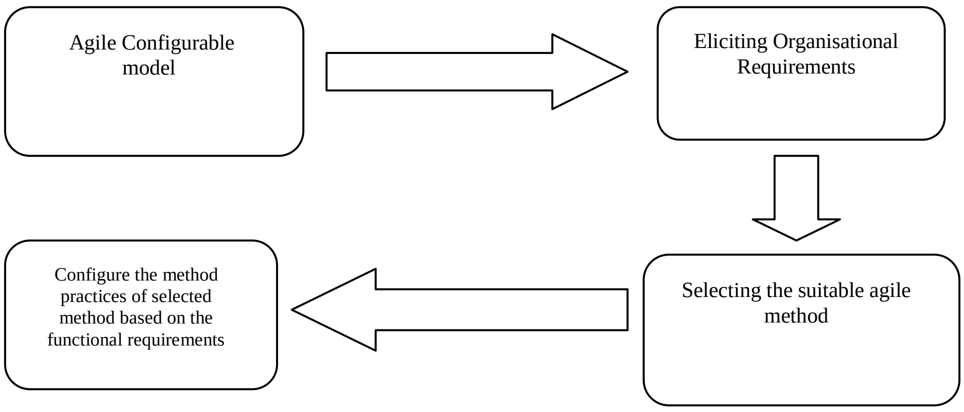

Agile Configurable model
Eliciting Organisational Requirements
Selecting the suitable agile method
Configure the method practices of selected method based on the functional requirements

Figure 4.1:- Agile Method Engineering Framework

The Agile Method engineering process starts with – first, defining the agile methods as agile configurable models (sec. 4.1). Similar to traditional configurable models, agile configurable models also supports an *Essentiality* attribute; this essentiality attribute can take two values either *common* or *variable*. The agile values defined in agile manifesto along with practical and theoretical experience of various developers and academicians forms the basis for defining the commonality and variability in these methods. The project characteristics are gathered in the form of organisational requirements (sec 4.2). Suitable methods are retrieved by mapping these organisational requirements with the agile method characteristics. Here, fuzzy rules are defined to solve the overlapping nature of the methods in accordance with the organisational requirements. The method retrieved is further configured to form project-specific methods (sec.4.3). The chapter offers to extract case-specific functional requirement for the current agile project. These functional requirements provide support to the method engineer for deciding the constituents of the configured method (sec 4.4). The process is illustrated with the help of two case studies (sec 4.5).

The next section will define the essentialities in agile methods and presents the configurable model of various agile methods.

4.1 Essentialities in agile methods

As described in chapter 1, a method has two aspects- product and process. The product aspect provides features for the product development whereas; process aspect is the route that needs to be followed to ensure the efficiency of the product development. The literature survey on various agile methods reveals that there exist many significant operational differences between the process aspects of these methods. Thus it is difficult to produce a generic model of an agile method configuration process with sufficient granularity to be useful for the purpose. This moved the research, to the practices or the product aspect of these methods. The agile practices are centred on the *Agile values* defined in **(**Agile Manifesto, 2001). To preserve agility, all popular agile methods found in literature, satisfies these agile values. So the present research considers these agile values as the basis for defining essentialities in the agile methods. In the next sub-section the chapter presents agile values defined in agile manifesto and the corresponding method practices.

4.1.1 Mapping between agile values and agile method practices

The ‘*agile movement*’ in software industry saw the ray of light when agile software development ‘manifesto’ got published by a group of software practitioners and consultant in 2001. The focal values honoured by the agilest are presented in the (Agile Manifesto, 2001):

The four core agile values defined in the manifesto are:

- Individuals and interactions over processes and tools.
- Working software over comprehensive documentation.
- Customer collaboration.
- Responding to change.

These agile values states that- the project managers have to let the development team and project user cooperate together or collaborate together. Developers need to concentrate on project delivery and not on project documentation. However, comprehensive documentation is valuable; don’t abandon the documentation completely. A balance need to be created between project delivery and project documentation. Customer collaboration requirements cannot be fully collected at the beginning of the software cycle. Therefore a continual customer or stakeholder involvement is very important, responding to change.

The practices of agile methods are divided into four groups corresponding to the four core agile values. Table 4.1 adapts from (Qumer and Henderson-Sellers, 2008b) shows the mapping between the practices of popular agile methods with the agile values. This mapping shows the support of agile values by agile methods. The mapping between the agile values and the practices of agile methods provide a guideline to define the *essentialities* in the method. The guideline is presented below:

Guideline: *To satisfy all agile values, at least one practice corresponding to an agile value must be considered as common to the method.*

Table 4.1:- Mapping between agile values and agile method practices.

Agile Values	XP	Scrum	FDD	ASD	DSDM	Crystal
Individuals and Interactions over processes and tools	1.Pair Programming 2.Collective Ownership 3.On-Site Customer 4.The planning game	1.Scrum Teams 2.Daily Scrum Meeting 3.Sprint Planning meeting	1. Domain Object Modelling. 2.Individual Class Ownership 3.Feature Teams 4.Inspection	1.Adaptive Management Model 2.Collaborative teams 3.Joint Application Development by independent agents 4.Customer Focus Group reviews	1. Empowered Teams. 2.Active User Involvement	1.Holistic Diversity and Strategy 2.Parallelism and Flux 3.User Viewings
Working Software over comprehensive documentation	1.Testing 2.Short releases 3.Continuous Integration	1. Sprint 2.Sprint Review	1.Developing By Feature 2.Inspection 3.Regular Builds 4.Reporting/Visibility of results	1.Developing by Components 2.Software Inspection 3.Project Post mortem	1.Frequent Product Delivery 2.Iterative and Incremental development 3.Integrated testing	1. Monitoring of a progress. 2.Revision and Review
Customer Collaboration over contract negotiation	1.The Planning Game 2.On-Site Customer	1.Sprint planning meeting 2 .Product Backlog	1.Domain Object Modelling	1.Adaptive Management Model 2.Joint Application Development	1.Collaboration and Cooperation among stakeholders 2.Requirements are baseline at a high level	1.Staging 2.User Viewings
Responding to change over following a plan	1.Metaphor 2.Simple Design 3.Refactoring 4. Coding standard	1.Sprint Planning meeting 2. Sprint Review 3. Sprint Retrospective 4. Scrum of Scrums	1. Domain Object Modelling. 2.Configuration Management	1.Adaptive Cycle Planning 2.Customer Focus group reviews	1.Reversible Changes	1.Reflection workshops 2.Methodology Tuning

Since the guideline is defined at a higher level it needs to be explored further to identify - **'commonality among the group of practices corresponding to an agile value'.** For the purpose, the practical and the theoretical experiences of various software developers and users are gathered and examined. The next sub-section presents the major outcomes of the research, used to decide the *essentialities* in method practices.

4.1.2 Determining the essentialities in agile methods.

The widely accepted agile methods- Extreme Programming, Scrum and DSDM were introduced in the early and mid 1990's and have been found well documented. There exists a

number of literature and experience support for them. Other methods that are also included in this research are FDD, crystal and ASD. However, less is known about their actual usage in real world but these methods has generated and maintained their own active research and user communities. Thus, they can be classified as "active" and are thus included in this research.
These methods have a well defined process and a set of practices that need to implement the process. To avert a repetition of arguments in the research and to present the effort contextually, we avoid exhibiting a review of the process, practices, roles and responsibilities of all the above methods. However, only the relevant points are briefly discussed and are presented in a nutshell. Interested readers are referred to (Abrahamsson, P. et al., 2002) to get a detailed overview on the agile methods.

Extreme Programming (XP)

XP has evolved from the "problems caused by traditional development models" (Beck, 1999a). (Haungs, 2001) first started as "simply an opportunity to get the job done". After a number of successful trials in (Beck, 1999b) the XP was "*well-documented*" on the key principles and practices used. The term "extreme" comes from taking the commonsense principles to extreme levels.

The Beck, defined that key features of XP are - customer driven development, small teams, daily builds and the special features that makes it distinct from others is '*refactoring*'-the ongoing redesign of the system to improve its performance and responsiveness to change.

(Qumer and Henderson-Sellers, 2008b) identifies that only XP discusses *code style* and standard; other agile methods don't specify it explicitly.

(Fitzgerald et al., 2006) found that developers at Intel Shannon formed a customized method, of XP. They took pair programming, testing, metaphor, collective ownership, refactoring, coding standards and simple design as the part of the customized method formed. Leaving behind planning game, small release, continuous integration, 40-hrs week and on-site customer. The customized method thus, formed behaves extremely well as compared to the original method. Similarly, in the literature another case study by (Rizwan and Qureshi, 2012) was found support the configuration of this method.

Thus, from the practical experiences and the available literature on XP, the essentiality of this method is defined as:

Common = {pair programming, testing, the planning game, metaphor, refactoring, coding style}.

Variables = {collective ownership, on-site customer, short releases, continuous integration, simple design}.

The next method under consideration is scrum. The term 'scrum' originally derives from a game strategy of Rug-by where it denotes "getting an out-of-ball back into the game".

Scrum

The scrum approach has been developed for managing the software development process. (Schwaber and beedle, 2002) identifies two situations in which scrum can be adopted: an existing project and a new project. For an existing project, the introduction of scrum is started with *daily scrum meetings* with a *scrum master*; the goal of first *sprint* should be "*to demonstrate any piece of user's functionality on the selected technology*". This will help the team to believe in itself.

Whereas, for a new project, Schwaber and beedle, suggests first working with the team and customer for several days to understand the requirements and develop an initial '*product backlog*'. The goal "*to demonstrate key process of user's functionality on the selected technology*". In this situation, apart from other practices they suggest product backlog as a must for the purpose.

By personally interviewing, software developers in the HCL technologies currently working on the leading projects like banking and aviation. It was found that they consider practices like-*product backlog, sprint, sprint planning meeting and daily scrum meeting* as *common* to their process.

(Scharff and verma, 2010) conducted a survey to verify the effectiveness of scrum for the development of mobile application. In their research they found that *sprints, product backlog and sprint backlog* as the most essential practice needed to be address during the development in this domain.

Thus, from the practical experiences and the available literature on Scrum, the essentiality of this method is defined as:

Common = {scrum teams, sprints, sprint planning meeting}.

Variables = {Daily scrum meeting, sprint review, product backlog, sprint retrospective, scrum of scrums}.

Crystal

(Cockburn, 2000) states that the crystal family of methodologies includes a number of different methodologies to address the diversities in the software projects in terms of size and criticality in the system.

(Qumer and Henderson-Sellers, 2008b) founds that out of all the popular agile methods like XP, scrum, FDD and ASD none of the method supports the 'leanness' attribute. Leanness attribute describes the cost effectiveness in the method. However, crystal practices like '*reflection workshop*' and '*monitoring of a progress*' supports the leanness that makes it distinct from other methods. Thus in deciding the essentiality in crystal these practices that makes crystal a unique method must be considered as common.

The essentiality in crystal is defined as:

Common = {reflecting workshops, holistic diversity and strategy, monitoring a progress, staging}.

Variables = {Parallelism and flux, user viewings, revision and review, methodology tuning}.

DSDM

Since its origin in 1994, the DSDM (Dynamic Systems Development Method) has gradually gained importance in the United Kingdom (Stapleton, 1997).

(Abrahamsson, P. et al., 2002) described that the fundamental idea behind DSDM is that "*instead of fixing the amount of functionality in a product, the teams are empowered enough to fix the release time and then adjust the amount of functionality accordingly*" thus keeping more stress on *empowered teams* and *frequent product delivery.*

Besides, this the essentiality is also computed keeping in mind that the DSDM is the only methodology to explicitly specify a collaborative and cooperative business culture.

The essentiality in DSDM is defined as:

Common = {Empowered teams, frequent product delivery, collaboration and cooperation among stakeholders, reversible changes}.

Variables = {Active user involvement, Iterative and incremental development, Integrating testing, requirements are baseline at a higher level}.

ASD

Adaptive software development (ASD) was developed by James A. Highsmith and published in (Highsmith, 2000). ASD focuses mainly on the problems in developing complex and large systems. According to Highsmith, "*The method strongly encourages planned, iterative development with constant prototyping*". Basically, (Highsmith, 2000), expresses three main practices – *component based development*; *adaptive cycle planning* and *customer focus group* *reviews*. Rest of the practices are left flexible and were defined as “what could be done rather than what should be done”.

The essentiality in ASD is defined as:

Common = {adaptive management model, customer focus group reviews, developing by components, adaptive cycle planning}.

Variables = {collaborative teams, joint application development by independent agents, software inspection, project post mortem}.

4.1.3 Configurable model for agile methods

The agile values, and the researches, and practical experiences are examined to decide commonalities in the method practices corresponding to an agile value. Table 4.1 is now modified by defining commonality and variability in the methods. The outcome is shown in table 4.2. A ‘C’ corresponding to an agile practice indicates that essentiality=common for the practice and ‘V’ indicates essentiality=variable.

Table 4.2:- Commonality and variability in popular agile methods

Agile Values	XP		Scrum		FDD		ASD		DSDM		Crystal	
Individuals and Interactions over processes and tools	**Pair Programming**	C	**Scrum Teams**	C	**Domain Object Modelling.**	C	**Adaptive Management Model**	C	**Empowered Teams.**	C	**Holistic Diversity and Strategy**	C
	Collective Ownership	V	Daily Scrum Meeting	V	Individual Class Ownership	V	Collaborative teams	V	Active User Involvement	V	Parallelism and Flux	V
	On-Site Customer	V	**Sprint Planning meeting**	C	Feature Teams	V	.JAD by independent agents	V			User Viewings	V
	The planning game	C			Inspection	V	Customer Focus Group reviews	V				
Working Software over comprehensive documentation	**Testing**	C	**Sprint**	C	**Developing By Feature**	C	**Developing by Components**	C	**Frequent Product Delivery**	C	**Monitoring of a progress**	C
	Short releases	V	Sprint Review	V	Inspection	V	Software Inspection	V	Iterative and Incremental development	V	Revision and Review	V
	Continuous Integration	V			Regular Builds	V	Project Post mortem	V	Integrated testing	V		
					Reporting/Visibility of results	V						
Customer Collaboration over contract negotiation	**The Planning Game**	C	**Sprint planning meeting**	C	**Domain Object Modelling**	C	**Adaptive Management Model**	C	**Collaboration and Cooperation among stakeholders**	C	**Staging**	C
	.On-Site Customer	V	.Product Backlog	V			JAD	V	Requirements are baseline at a high level	V	User Viewings	V
Responding to change over following a plan	**Metaphor**	C	**Sprint Planning meeting**	C	**Domain Object Modelling**	C	**Adaptive Cycle Planning**	C	**Reversible Changes**	C	**Reflection workshops**	C
	Simple Design	V	Sprint Review	V	.Configuration Management	V	**Customer Focus group reviews**	C			Methodology Tuning	V
	Refactoring	C	Sprint Retrospective	V								
	Coding standard	C	Scrum of Scrums	V								

From the above table the configurable models of agile methods can be drawn. For example:

XP as an agile configurable model

<Pair-Programming>	<Common>
<The Planning game>	<Common>
<Short releases>	<Variable>
<Metaphor>	<Common>
<Collective ownership>	<Variable>
<On-site customer>	<Variable>
<Testing>	<Common>
<Continuous Integration>	<Variable>
<Simple Design>	<Variable>
<Coding Standard>	<Common>
<Refactoring>	<Common>

Similarly, configurable models of all agile methods can be drawn. Now just as the traditional method configuration process yields a family of configured methods, so also agile method configuration process produces a family of methods. For example:

Table 4.3:- Instances of XP configured method

	Configured Methods of XP
1.	All XP common concepts, with on-site customer and collective ownership.
2.	All XP common concepts, with simple design and continuous integration.

4.2 Gathering Project requirements as Organisational Requirements

Before taking up the issue of organisational requirements, it is important to understand the organisational environment. An organisational environment is a demarcated environment where the software engineering method or its resulting artefacts are used. Depending upon the environment under consideration, the organisational requirements are gathered.

The method engineer is role responsible for gathering organisational requirements; such competence should already exist in-house. Since, there is a difference between organisation-specific environment and ideal environment about standard software engineering methods. A method engineer is familiar with the organization, in which the software engineering method is going to exist in. Hence, he or she possesses general knowledge about methods, as well as organizational knowledge. The later is significant because systems engineering methods can exist in different realms (Goldkuhl and Braff 2002).

Table 4.4 shows the set of Organisational requirements and the corresponding agile methods support that gives as an input to Fuzzy Logic Controller to find the membership metrics. However, for future purpose the set can be extended and many other fine-grain features can be added. We adapted the table from research by (Qumer and Henderson-Sellers, 2008a).

Since requirements can be conceptualised in many ways, the present research prefer to use fuzzy logic controller to handle the vagueness in the elicited requirements. As described by (Marcelloni and Akshi, 97) in fuzzy logic, concept of vagueness is introduced by the definition of fuzzy set.

Table 4.4:- Organisational requirements and corresponding method support.

Characteristics	***Values***	***Methods support***
Task extent	*Small*	*XP, SCRUM, FDD, DSDM, Crystal*
	Medium	*XP, SCRUM, FDD, Crystal*
	Large	*FDD, ASD*
	Complex	*ASD*
Group Size	*Less than 10*	*XP, SCRUM, Crystal*
	Multiple Teams	*SCRUM, DSDM, Crystal*
	No limits	*FDD*
Progress Approach	*Iterative*	*XP, SCRUM, FDD,ASD, DSDM, Crystal*
	Rapid Development	*XP, SCRUM,ASD, DSDM, Crystal*
	Distributed Development	*ASD*
Code Style	*Clean and Simple*	*XP*
	Not Specified	*SCRUM, FDD,ASD, DSDM, Crystal*
Expertise Environment	*Quick Feedback*	*XP*
	Not Specified	*SCRUM, FDD,ASD, DSDM, Crystal*
Physical Environment	*Co-located teams*	*XP, ASD, Crystal*
	Distributed teams	*XP, ASD*
	Not Specified	*SCRUM, FDD, DSDM*
Industry customs	*Collaborative and Cooperation*	*XP, DSDM*
	Not Specified	*SCRUM, FDD, ASD, Crystal*
Abstraction Mechanism	*Object-oriented*	*XP, SCRUM, FDD, ASD, DSDM, Crystal*
	Component-oriented	*ASD, DSDM*

A fuzzy set S of a universe of discourse U is characterized by a membership function which associates with each element y of U a number of intervals which represents the grade of membership of y in S. based on the definition of fuzzy sets, the concept of linguistic variables is introduced to represent a language typically adopted by a human expert. A linguistic variable is a variable whose values, called linguistic values, have the form of phrases or sentences in a natural or artificial language. For instance, the relevance of an agile method for the situation-in-hand can be modelled as a linguistic variable which might assume linguistic values weakly, fairly and strongly relevant.

Why Machine Learning is required

(Zhang, 2003) founds that through machine learning many software engineering problems can be efficiently solved. The agile method selection models are expresses by using two-value logic. For instance, an agile method for a particular situation is either accepted or rejected. There are two major problems in the way of how rules are defined and applied in the current agile method engineering process. The first problem termed "*Selection problem*" is a natural result of the incapability of 2-value logic to express the approximate and in-exact nature of agile methods in a typical software development and management process. The research reveals that more than one agile method can support a specific characteristic value for ex. For the characteristics value-small task extent, the methods support exists is comprised of *XP, SCRUM, FDD, DSDM, Crystal* methods.

The second problem termed '*configuration problem*', arises because most of the method users tend to use parts of the methods rather than the complete method defined. Method Configuration approach makes possible for an organization to configure and extend agile methods to meet their specific needs; it allows the organization to make changes in the methods without losing the purpose to stay on the method.

Fuzzy rules are used to find the membership metrics of the methods under consideration. The purpose is to select the most suitable methods; FLC will assign membership to the methods depicting the degree of perfectness for the defined set of requirements.

An example for the set of rules formed is as follows:-

If <*Task extent*> is <*small*> then <*XP*> is a <*good* > method to select.

If <*Task extent*> is <*medium*> then <*XP*> is a < *good*> method to select.

If <*Task extent*> is <*large*> then <*FDD*> is a < *very good*> method to select.

If <*Task extent*> is <*complex*> then <*ASD*> is the <*only*> method to select.

If <*Task extent*> is <*small*> then <*Scrum*> is a < *good* > method to select.

If <*Task extent*> is <*smal*l> then <*FDD*> is a < *good* > method to select.

If <*Task exten*t> is <*small*> then <*DSDM*> is a < *good* > method to select.

If <*Task extent*> is <*small*> then <*Crystal*> is a < *good* > method to select.

If <*Task extent*> is <*medium*> then <*Scrum*> is a < *good* > method to select.

If <*Task extent*> is <*medium*> then <*FDD*> is a < *good* > method to select.

If <*Task extent*> is <*medium*> then <*DSDM*> is a < *good* > method to select.

If <*Task extent*> is <*large*> then <*ASD*> is a < *very good* > method to select.

Similarly, the rules are made for the entire domain. The shift from two-valued to fuzzy logic rules in software development is quite natural. This is because the design rules for the '*selection of most appropriate method*' are applied to solve the overlapping nature of methods (in behaviour and characteristic domain). *Like both XP and SCRUM may be used for development process and they support small and medium size projects.*

By using fuzzy rules complete set of organisational requirements are considered to calculate the member ship degree of each agile method under consideration.

4.3 Retrieving the suitable agile methods

As explained in sec. 4.2, the suitable methods are found by mapping the elicited organisational requirements with the method characteristics. For example, the set of elicited organisational requirements is given in table 4.5.

Table 4.5:- Set of elicited Organisation Requirements

	Characteristics	**Values**
R1	Task extent	Small
R2	Group size	Less than 10
R3	Development Style	Rapid Development
R4	Code Style	Clean and Simple
R5	Technology Environment	Quick Feedback
R6	Physical Environment	Distributed teams
R7	Business Culture	Collaborative and cooperation
R8	Abstraction Mechanism	Object-Oriented

The fuzzy logic controller will calculate the membership of agile methods, corresponding to the elicited organisational requirements and after evaluating the complete set of requirements the membership metric will be generated. For the above set of elicited requirements *SCRUM has membership of 83% and the method XP has membership degree of 68%* and so on. Once the candidate method got selected, the present agile method engineering process moves towards the method configuration process to form project-specific method.

4.4 Configuring the agile method

Despite, the reported success of agile methods, the significant concern regarding these methods is that – they lack the factor of 'Discipline' (Fuller and Croll (2004). Since these methods adhere to a set of practices rather than follow a common process for the development. Results in the "significant operational difference" between them. *Thus it is difficult to produce a generic model of an agile process with sufficient granularity to be useful for the purpose.*

Further, to configure agile methods and to provide support for selecting *variables* in these methods. *It would prefer to consider each agile project individually rather than to provide a generic mapping between the practices and set of guidelines as is done in case of traditional methods.*
Following is a case study of an agile project, after analysing the project; the functional requirements for the project are identified. These 'functional requirements' provide support to the method engineer for deciding the 'constituents of configured method'.

4.5 Case studies

To show the practical implementation of the proposed methodology, case studies are used as a research method. In chapter 2, it was found that for mobile application domain agile development methodology is preferred over traditional methodology. The following case study shows – how an agile method is configured to form project-specific method for mobile application domain.

Case Study 1: A large software project developed for a mobile company to produce a usage analysis tool for analysing the customer's requirements in this domain and intelligently studies the areas for the development in this domain. It involves a huge and highly experienced team for its development which are further distributed into small teams. The project uses the complex technology for the implementation. The average duration of the project was 1 year. There is a need for documented requirements, to track the progress of the project and further to help during the testing phase. Extracting the functional requirements from the case study (table 4.6):

Table 4.6:- Extracted Functional Requirements for case study 1.

Number	Requirements
R1	Large Software
R2	Complex Technology
R3	Experienced teams
R4	Distributed teams
R5	Documented Requirements
R6	Iterative Developments

In accordance, with the set of requirements weights are assigned to the method practices of the selected method (*Scrum* in this case, refer sec 4.3). Practices of Scrum:

Table 4.7:- Weighted practices of scrum for the case project 1.

Number	Practice	Weight
P1	Product Backlog	0.8
P2	Sprint Review	0.4
P3	Scrum teams	0.9
P4	Sprint	0.8
P5	Daily Scrum meeting	0.3
P6	Sprint planning meeting	0.6
P7	Sprint retrospective	0.0
P8	Scrum of Scrums	0.2

These weighted practices will provide a support system to the method engineer to select the *variables* in a method. The scrum process and configurable model of scrum (refer table 4.2) are given below:-

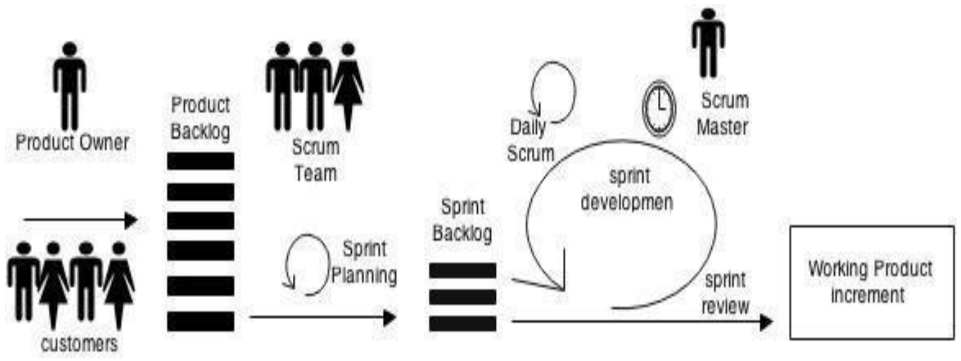


Figure 4.2:- Scrum Product model (AlMutairi et al., 2015)

The configurable model of scrum,

Common = {sprint, scrum planning meeting, scrum team}

Variable = {product backlog, sprint review, daily scrum meeting, sprint retrospective, scrum of scrums}

For the case project, the '*product backlog*' is found heavily weighted thus, among the set of variable it needs to be add in the configured method. The less weighted practices '*sprint review*' and '*daily scrum meeting*' and '*scrum of scrums*' can be tailored or modified for the purpose. However, '*sprint retrospective*' is removed from the configured method. Hence, the configured method formed for the current project is:

Configured Method Scrum for the case project: ***All Scrum 'common concepts' with 'product backlog'*** **and modified '*sprint review*', '*daily scrum*' and '*scrum of scrums*'**

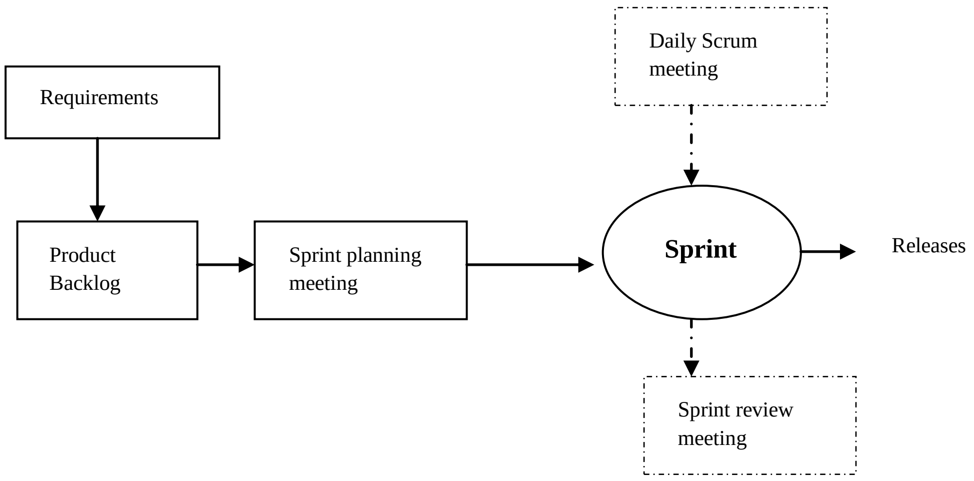


Figure 4.3:- Configured model of Scrum for case project 1.

Since, daily scrum meetings can be considered as *variable* to the method the effect on the process can be understood as,

Daily scrum meetings are organised to keep track of the progress of the scrum team. It monitors and plans the sprint development and ideally need to be held daily. Sprint planning meeting also serves a similar purpose with a less frequency of occurrence. For an experienced team, where the developers have worked on a similar project and users are clear and confident on their requirements, daily scrum meeting can be absorbed under sprint planning meeting. Similarly, sprint review meeting is done between all the stakeholders of the project at the end of each sprint. The purpose is to bring out new backlog items or even change the direction of the development. For an experienced group of stakeholders, these meetings can be avoided or at least the frequency can be reduced to some extent. However, scrum of scrums have a very less weight age because it holds rarely - most of the times other meetings and communication opportunities were considered to be sufficient.

Case Study 2**:** For illustration, the chapter is using the case study presented by (Hossain et al., 2011). The case study describes an EnergyInfo project. '*The project was the part of a large product development to control a power, energy and oil refinery system. The project was new and had moderate change requirements. The project manager hired a development team from a nearby country. The onshore management team's main task was to generate and maintain specifications provided to the offshore development team. The onshore team had over ten yrs of experience in software development. However, offshore were less experienced*'. The extracted set of functional requirements is:

Table 4.8:- Extracted Functional Requirements for case study 2.

Number	Requirements
R1	Large Software
R2	Complex Technology
R3	Experienced management team
R4	Less experienced development team
R4	Distributed teams
R5	Moderate changing requirements
R6	Iterative Developments

In accordance, with the set of requirements weights are assigned to the method practices of the Scrum are:

Table 4.9:- Weighted practices of scrum for the case project 2.

Number	Practice	Weight
P1	Product Backlog	0.9
P2	Sprint Review	0.9
P3	Scrum teams	0.9
P4	Sprint	0.8
P5	Daily Scrum meeting	0.5
P6	Sprint planning meeting	0.6
P7	Sprint retrospective	0.0
P8	Scrum of Scrums	0.0

For the case project, the '*product backlog*' is found heavily weighted thus, among the set of variable it needs to be add in the configured method. The high weight to '*sprint review*' is because the off shore needs to present the task to onshore team after each sprint. '*daily scrum meeting*' had a moderate weight because it is essential for off shore team, since they are less experienced, but is not required for onshore team. However, '*scrum of scrums*' and '*sprint* *retrospective*' are removed from the configured method. Hence, the configured method formed for the current project is:

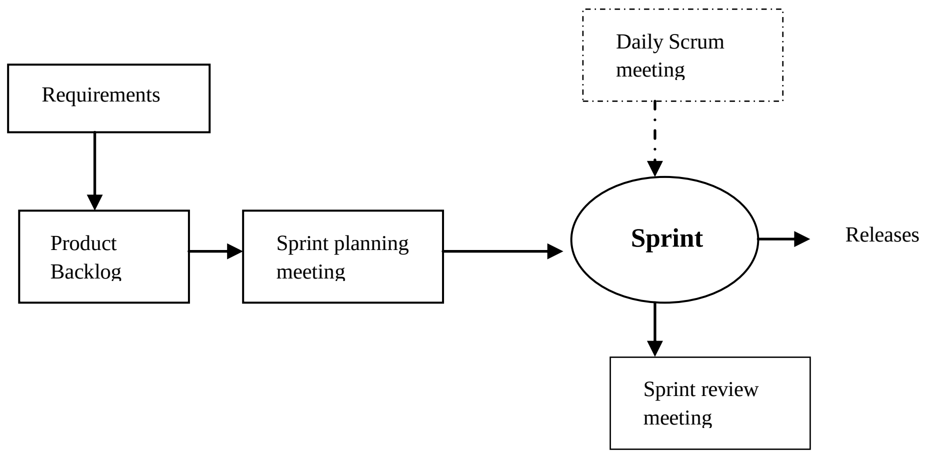
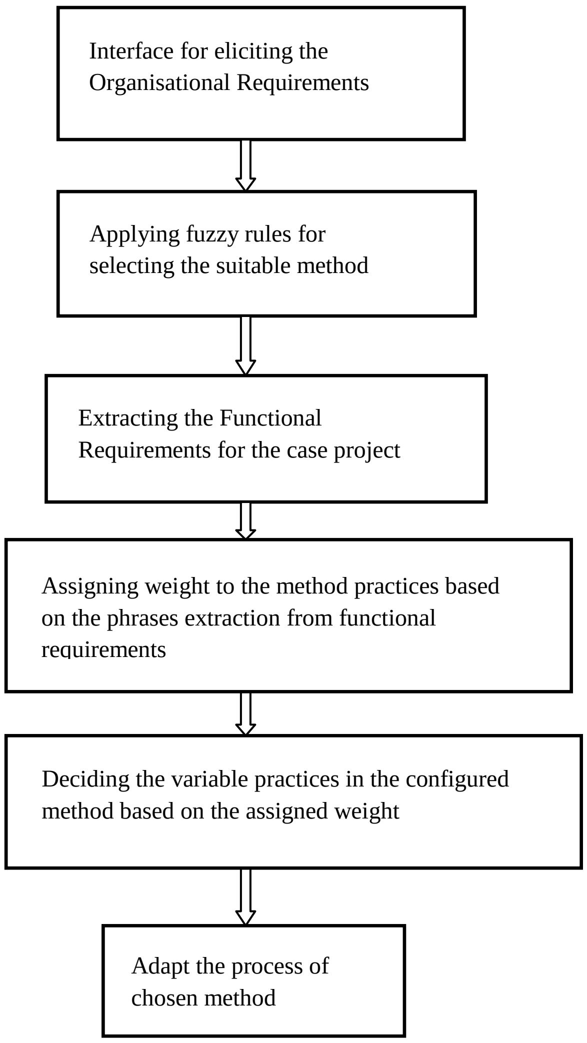


Figure 4.4:- Configured model of Scrum for case project 2.

Configured Method Scrum for the case project: *All Scrum 'common concepts' with 'product backlog'*, '*sprint review*' and modified '*daily scrum meeting*'.

4.6 Functional Architecture of the AME process

In order to provide the context of the proposed agile method engineering process, this section provides the functional architecture of the process along with the implementation details. The functional architecture is as follows



Interface for eliciting the Organisational Requirements
Applying fuzzy rules for selecting the suitable method
Extracting the Functional Requirements for the case project
Assigning weight to the method practices based on the phrases extraction from functional requirements
Deciding the variable practices in the configured method based on the assigned weight
Adapt the process of chosen method

Figure 4.5:- Functional Architecture of AME process

First, the interface for eliciting the organisational requirements has been provided to capture the organisation specific requirements hence, bridging the gap between the developers and method engineer. The interface is implemented using the .NET framework. The fuzzy rules are implemented in MATLAB to find the most suitable method in accordance with the elicited set of organisational requirements. As mentioned earlier, the functional requirements are extracted from the case projects that need to be developed in the organisation. These functional requirements are used for phrase extractions that are further mapped with the practices of the suitable method found. The mapping helps in assigning weight to the practices of the method. For example, the functional requirements like- large software and complex technology, needs high support from the ‘product backlog’ and ‘product review’ agile practices. Results, in assigning a very high weight corresponding to these agile practices.

The weighted practices provide the means to select the variable practices and hence deciding the method constituents. Once, the project-specific method is formed the process is adapted respectively.

Summary

In today’s dynamic market environment producing high quality software rapidly and effectively is crucial. In order to allow fast and reliable development process, several agile methodologies have been designed and are now quite popular. Software developers find these methods as interesting and are concentrating more and more on these light-weight methods. Through their practical experience in the field it was found that agile processes may individually be incomplete to support the whole development process well, hence their processes require to be tailored to meet the requirements.

This arise a need to apply method engineering principles and practices to agile methods. As mentioned in the chapter that these methods have a significant difference in their process thus, it is difficult to produce a generic model for them. They can only be adapted for the project-specific needs using configuration process.

The agile method engineering approach finds the degree of veracity of these methods for the specified set of requirements and configures them to form project specific methods. The method configuration process, supports configurable models, these models illustrates the essential component of agile methods and is an attempt to show that “being agile” is a specific combination of practices only.

This revolutionary approach opens the paths to utilize the revolution brought by the concept of agility. The process supports to specify the requirements in laymen language and finds the suitable agile methods for the same with the practices that need to be followed. The aim is to deliver project specific agile method for the current organisation requirement. Sometimes, method configuration alone fails to form project-specific method. Methods need to extend their capabilities by inheriting some product features of another method or by assembling their product entities with other methods to satisfy the complete set of requirements. The next chapter addresses this problem by presenting the method extension process.

Chapter 5

Method Extension

Certain software engineering methods supports partial phase of Software Development Life Cycle and may require to be extended for other phases also. For example, Data Flow Diagram (DFD) introduced by Gane and Sarson, in year 1979 (Gane and Sarson, 1979), focuses on the design phase of the software development and aims at showing the flow of data in a system. Later on, it may require extending DFD's for requirement phase. If this extension is done in an ad-hoc manner it may not results in a coherent and consistent method. In turn it could add to the project-risk list.

Using the proposed technique of Method Configuration, the chapter presents the process of method extension. Method extension should be attempted only when configurability fails to deliver the desired method. This failure can happen

- If a method concept is missing in the desired method.
- If a method needs to extend its functionality.

The thesis proposed that a method can be extended- **Either** by adding more concepts in it, the purpose is to make the method more efficient for the software development phase to which the method was initially designed for, **OR** to extend the method to get functional on other phases of software development.

5.1 Extending method with missing product entities

As described in chapter 3, the retrieved configurable method from the method base exists in one of these three forms

- The method is sufficient and complete to create the desired method.
- The retrieved method cannot lead towards the desired method. It is discarded, and another one is considered.
- The retrieved configurable method partially meets the requirements. In this case, method extension is to be performed.

During method extension, the method engineer selects and integrates the missing product entities to form the desired method. For method extension, the external view of the method that represents only the utility of the method is considered.

Steps for method extension:

- The Method engineer selects the missing method concepts.
- Instantiate the ‘is composed of’ or ‘is mapped to’ relationships in which the new method concept or component participates.
- Add the new method concept in the original method conceptual model or method component model of the method.
- The new method concept or component has essentiality =variable in the method conceptual model or method component model.
- Generate the modified set of purposes and dependencies.

The modified set of purposes and dependencies are generated based on the set of rules. The rules are adapted from the generic rules given by (Gupta and Prakash, 2001). To get better understanding, a flavour of these rules is given below:-

5.1.1 Rules for adding a new method concept in method conceptual model

Whenever a new method concept is added in the method conceptual model following operation, need to perform

- The new method concept must be imported from some other method.
- The new method concept must enter in the 'is composed of' relationship with some existing method concept in the method.
- It must satisfy the completeness, conformity and fidelity constraints.

For example, to add *Generalisation* with the *use-cases* in the basic Use Case Diagram that show user's interaction with the system **(**Rumbaugh J. et. al, 1991)**.** In this situation, the new method concept added, must enter into the chunk of purposes i.e. Basic life cycle purpose, Relational purpose, Constraint Enforcement Purpose and Integration class purposes. Since the set of purposes gets modified, the dependencies and the constraints have to mutate accordingly.

For every method concept S_i to be added define the purposes:

< $S_{i,}$ create> and < $S_{i,}$ delete >

<Generalisation, create> < Generalisation, delete>

Rules for Relational purposes

Rule-1: For all added method concepts S_i such that S_i *is composed of* S_j that already exists and S_i is not a collection of concepts, generate the relational purposes

< $S_{j,}$ S_i , O> and < $S_{j,}$ S_i , O'>

The operations O and O' are defined as follows:

- If S_j is simple definitional and S_i is complex definitional, then they are *attach* and *detach respectively*.
- If S_j is complex definitional and S_i is complex definitional, then they *join and dejoin* respectively.

- If S_j is simple constructional or complex constructional and S_i is complex constructional, then they are *associate* and *dissociate* respectively.
- If S_j is definitional and S_i is constructional or link then, they are a *couple* and *uncouple* respectively.

Rule-2: - If S_i is a collection of concepts and *is composed of* a s_k and S_j where S_k is of link type and S_j is of constructional type only then generate following purposes:

< S_{j1}, S_{j2}, s_k, S_i, relate> and < S_{j1}, S_{j2}, s_k, S_i, unrelate>

Where, S_{j1} and S_{j2} are two instances of S_j.

For our example, following purposes are generated

<use_case1, use_case2, extend_link, generalisation, relate>

<use_case1, use_case2, extend_link, generalisation, unrelate>

Method Conceptual model of Use Case $_{conf}$

Method Concepts	Essentiality
<Actor>	*<Common>*
<use-case>	*<Common>*
<Generalisation>	*<Variable>*
<assoc-link>	*<Variable>*
<include-link>	*<Variable>*
<extends-link>	*<Variable>*

5.1.2 Rules for adding a new method component in method compound model

Whenever a new method component is added, in the method component model of compound method. Following operation need to perform

- The new method component must be imported from some other method.
- The new method component must enter in the 'is mapped to' relationship with some method component exists in the method.
- It must satisfy the icompleteness, iconformity and ifidelity constraints.

For example: To extend the object model of OMT method with activity diagram of UML method.

<u>Generating Integration Purposes</u>

<u>Rule-1 :</u> For every method concept s_i of a method component M_1 that *is mapped to* the added method concept sj of other method component M_2, generate following integration purposes:

(i) M_2: <s_j, export >, <s_j, withdraw >

(ii) M_1: <s_j, import>, <s_j, dump>

For example,

UML:	**<activity diagram, export>, < activity diagram, withdraw>**
OMT:	**< activity diagram, import>, < activity diagram, dump>**

Rule-2: If type of the method component is constructional, then for method concept S_i of method M_1 which *is mapped to* method concept Sj in method component M_2 generate following integration purposes:

M_1:<s_j, s_i, correspond>, <s_j, s_i, separate>

For example,

OMT :< object model, activity diagram, correspond > < object model, activity diagram, separate>

Rule-3: If type of the method component is transformational, then for every method concept s_i of method component M_1 which *is mapped to* method concept sj in method component M_2 generate following integration purposes:

M_1: <s_j, s_i, convert>, <s_j, s_i, deconvert>

Compositional Constraint Purposes

Rule-1: For every compositional completeness structure, s_i_icompleteness, generate the purpose

<s_i, s_i_icompleteness, enforce_s_i_icompleteness>

Rule-2: For every compositional conformity structure, s_i_iconformity, generate the purpose

<s_i, s_i_iconformity, enforce_s_i_iconformity>

Rule-3: For every compositional fidelity structure, s_i_ifidelity, generate the purpose

<s_i , s_i _ifidelity, enforce_s_i _ifidelity>

For example, these rules will generate following purposes:

<activity_diagram, activity_diagram _icompleteness enforce_ activity_diagram _icompleteness>.

<activity_diagram, activity_diagram _iconsistency, enforce_ activity_diagram _iconsistency>.

< activity_diagram, activity_diagram _ifidelity, enforce_ activity_diagram _ifidelity>.

5.2 Extending the FDD- for complex project applications

In this section, the chapter presents how the agile method FDD, can be extended to form a project-specific method.

FDD is a modern agile approach, but it lacks to cover the entire software development process rather focuses on design and building phases. The FDD approach embodies iterative development with the best practices found to be effective in the industry. In the original framework of FDD, first reported by (Coad et al., 2000) and was further extended by (Palmer and Felsing, 2002). The FDD consists of five sequential processes during which the designing and building of the overall system are carried out.

The first process, Develop an overall model has a prerequisite that the domain expert are already aware of scope, context and requirements of the system to be built. The actors involved in this process are domain experts, chief architects and team members. The output of the process is class diagrams, sequence diagrams and the model notes. However, FDD does not explicitly address the issue of gathering and managing the requirements.

For broad and complex systems, a steady flow of elicited requirements is necessary for smooth functioning of the process. In order to meet this requirement, the FDD method should extend by adding a new process of Requirement Exploration.

The Requirement Exploration phase takes input from the stakeholders and output backs the requirement definitions to develop the overall model. The role of the requirement engineer is paramount since he handles the task of decomposing the requirements from coarse-grained to fine-grained level. Once the requirements are clearly defined, analysed and prioritised, the requirement definitions thus written are input to develop the overall model of the system. Based on the developed model feature list is formed, features represent the different activities within the specified domain area. Next, the effective plan for the sequencing and execution of the feature sets have to do, this noble planning act as an input for the designing and building of the features in practice. Since the case project is a large and complex project a slight variation in the planning phase is also needed. In the original FDD framework feature are planned, designed and then build but for a complex application, the situation is different complex applications are not made, they evolve. They require an adaptive planning process that must be capable of learning from the build features, the volume of information shall be collected, analyzed and applied for future planning of the new and critical features.

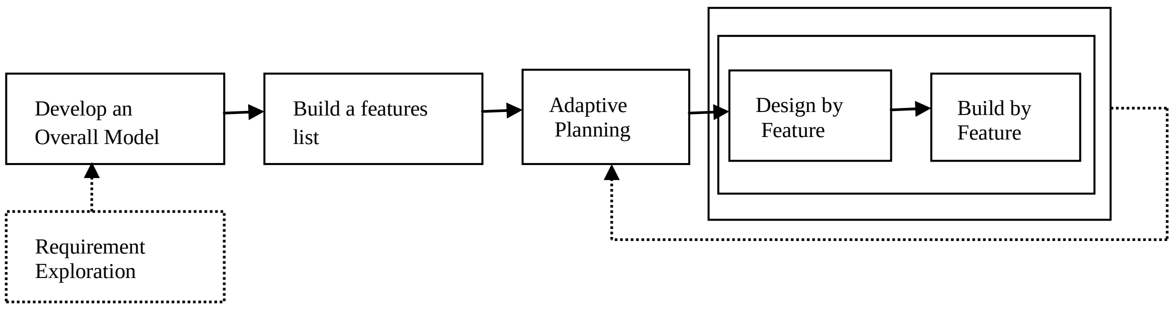


Figure 5.1:- Extended FDD process framework for complex project applications

Above is the diagrammatic view of the extended FDD process for the project in hand. The dotted lines present the new extended phases in the method to address the needs of the large and complex projects.

5.3 Method extending its functionality by assembling with other methods

As described earlier, that standard software engineering methods usually designed for delimited parts of software development life cycle, leaving other phases of SDLC in an ad-hoc environment. This results in a need to extend these methods with the functionalities of other methods and empowered them to support other phases of Software Development as well.

The process of assembly starts with the retrieval of highly weighted methods, as per the organisational requirements explained in chapter 4. The practices of highly weighted methods are further analysed with the extracted requirements of the case project. These requirements provide the guidelines to method engineer for selecting *variables*.

5.3.1 Assigning values to method practices

Assign ‘1’to all the common practices of the highly weighted methods, and a ‘0’to all the remnant practices termed as variables.

5.3.2 Assigning the colour scheme

In order to differentiate between the retrieved highly weighted configurable methods, a different colour is assigned to each method. Say ‘red’ colour to method M1 and ‘green’ colour to method M2 and so on.

5.3.3 Assembling the individual highly weighted configured methods

This process requires to perform logical OR operation but with an exception that if there are two 1's of different colour both will be considered and will be appended to the result. If the numbers of practices are not same, a 'don't care condition(X)' is appended. The obtained outcome is the situated latter method formed by assembling the individually configured method.

Consider '1' as a representation of *common* practices of method M1 and '1' is *common* for practices of method M2. A '0' represents the variable practice of either method.

There may arise 4 cases during the OR operation:

1. 1 OR 0: output will be 1 in method part.
2. 1 OR 0: output will be 1 in method part.
3. 0 OR 0: output will be 0 in method part.
4. 1 OR 1: output will be 1 in the method part and 1 in the method extension part.

5.3.4 Method Representation

The final output of the above step is represented into two parts: method part and method extended part. **Method part** includes the actual part that comes out of the OR operation whose length is the length of the maximum of the two method representations M1 and M2.

Method extension part plays a role for the 4th case of OR operation discussed above when at a particular position both methods have the common practices. In this situation, method extension part contains the *commons* of the second method.

5.4 Empirical grounding: The illustrations

An organisation needs to work on the following case study. As a prime need, the organisational requirements are elicited, and most suitable methods were found.

The two most highly weighted methods found by Fuzzy Logic Controller after applying fuzzy rules as described in the chapter 4 of this thesis, for further processing are:-

Table 5.1:- Retrieved Methods for the current organisational requirements.

Number	Method	Weight
M1	Dynamic System Development Method(DSDM)	0.9
M2	Feature Driven Development(FDD)	0.8

Case study: An organisation needs to upgrade the existing code into a large software project. It was being developed at a University that has many colleges located at various different places, and each college administration used the software for the academy management and placement management of the students. It involved the iterative and incremental development of the software. The project had seen an active user involvement during the development of the project because of the ever changing requirements of the customer. Since, it was needed by colleges at different locations, so teams were also spread in various locations for the software development, so it was a distributed development project.

Extracting the requirements,

Table 5.2:- Extracted Requirements of the case project.

Number	Requirement
R1	Upgrading code
R2	Large project
R3	Active user involvement
R4	Iterative Development
R5	Changing requirements

R6	Distributed development
R7	Object-oriented approach
R8	Incremental development

These requirements are further analyzed to select variables from the configurable model.

Table 5.3:- The configurable model of DSDM.

Number	Practice	Essentiality
P1	Active User involvement	Variable
P2	Empowered team	Common
P3	Frequent product delivery	Common
P4	Iterative and incremental delivery	Variable
P5	Reversible changes	Common
P6	Requirements are baselined at high level	Variable
P7	Integrated Testing	Variable
P8	Collaborative and cooperative approach shared by stakeholders	Common

Some of the configured methods formed are:

Table 5.4:- Instances of DSDM configured method

	Configured Methods of DSDM
1.	All DSDM common concepts with the empowered team and integrating testing.
2.	All DSDM common concepts with empowered team and iterative & incremental development
3	All DSDM common concepts with incremental development and integrating testing.

The method engineer will map these configured methods with the requirement set extracted out from the case project to decide the remnant values.

Similarly, the configurable method model of FDD is given below.

Table 5.5:- The configurable model of FDD.

Number	Practice	Weight
P1	Domain object modelling	Common
P2	Developing by feature	Common
P3	Inspection	Variable
P4	Individual class ownership	Variable
P5	Feature teams	Variable
P6	Regular builds	Variable
P7	Configuration management	Variable
P8	Progress reporting	Variable

Some of the configured methods of FDD are:

Table 5.6:- Instances of FDD configured method.

	Configured Methods of FDD
1.	All FDD common concepts with regular inspection and individual class ownership.
2.	All FDD common concepts with regular builds and progress reporting
3	All FDD common concepts with regular builds and individual class ownership.

Assembling the methods

As explained in sec 5.1.1. A‘1’ in the configurable method representation, shows that particular practice is *common,* and it should be included in the situated method whereas, a ‘0’ represents the variability in the configured method. Separate colour schemes are assigned to the methods for precise identification say, RED colour to the method DSDM and Green to the method FDD.

Method representations of Method M1 and M2 after all the calculations are:-

For Method M1 (DSDM)

0	1	1	0	1	0	0	1

For Method M2 (FDD)

1	1	0	0	0	0	0	0

If the length of the considered methods is not same, a 'don't care' condition is appended at the end of the Method representation; the purpose is to make equal lengths of considered methods. The situated final method is formed by performing the Oring operation on the considered methods.

Performing OR operation on the two method equivalents. -> M1 OR M2.

0	1	1	0	1	0	0	1
1	1	0	0	0	0	0	0

Generated Output: Assembled method formed having *commons* of both M1 and M2.

1	1	1	0	1	0	0	1	0	1	
1	2	3	4	5	6	7	8	1	2	

|--- ------Method part----------||---Method Extension part---

Understanding the output

The output method representation shows the practices that must be taken as represented by '1' of both colours. The red coloured 1's at position 2, 3, 5 and 8 signifies the presence of common practices of M1 in the assembled method formed. The green '1' at position first means the presence of first common practice of M2 in the assembled method formed; this completed the method part.

Now, in the method extension part, the green 1's at position 2 signifies the second common practice of method M2 in the assembled method formed. As explained in the case 4 of the Oring operation. Thus, the assembled method formed consists of the *common* practices of both the basic methods.

Summary

The chapter presented the Method Extension process- the process supports the assembling of different methods based on the rich knowledge of the past usage of these methods under different requirement sets. The applicability of the method thus formed will be significantly improved than the existing methods because the extended method contains the required constituent of more than one method.

Chapter 6

Conclusion, Contribution and Future work

The thesis brought out certain limitations in the current Software Engineering Methods. Firstly, it was observed that these methods are rigid and inflexible-resulting in their incapability to adapt according to the situation. The target organization needs to implement these methods in a way that is similar to implement a standard system. However, it has also been pointed out by method engineering community that there exist specific requirements for each project, for example, considering multiplicity of attributes and development team. Notice also, that while developing software, the organisation must initially, decide the suitable software development methodology for the current project characteristics. A study of available literature shows that software companies rarely address issue of software development methodology selection for the current project.

It was observed that software development community has adopted method configuration to form project-specific method for both agile methodology and traditional methodology but have not treated the 'notion of essentiality' in a method. The consequence of this is that

- The relationship between the original method and configured method is not fully explored. Thus, the extent to which a method can be configured, remain unanswered. This demands a full investigation into what can be configured into which method.
- Configurable Meta model have not been developed. It is therefore difficult to map the concepts of configurable model and to suppress the details of the method configuration process.

- Software companies are assuming ‘one development methodology fits for all’. This lead to an emphasis on determining the appropriate software development methodology for the current project.

Not only eliciting project characteristics for software development methodology selection is necessary but also the organizational requirements (for agile methods) and global properties (for traditional methods) for selecting suitable configurable method are important. While considering agile methods, the thesis argues that these methods have a significant operational difference in them. Therefore, functional requirements need to consider for configuring the agile method.

To sum up, the thesis found that there is a requirement to address the software development methodology selection problem and to develop a method configuration process supported by a configurable Meta model to configure the methods and turning them as project-specific methods.

In addressing these limitations, the thesis offers a solution for configuring methods for forming project-specific methods. In this regard, the thesis starts by a decision support system for software development methodology selection. The two broad category of methodology is considered: Traditional and Agile. Traditional methods are rigid and procedural oriented methods whereas, agile methods are light-weight and flexible. The thesis addresses the configuration process for both, the proposed process is analogous to system configuration. The system configuration is based on the construction of a ‘configurable system model’ that represents the essential system concepts and the interrelationship between these. Similarly, The task of configurability is first to create a new model called a configurable method model followed by selecting those parts of the configurable model that are relevant to the user’s requirement.

The generic framework presented in figure 6.1 presents the overview of the complete research. It is divided into three phases about the execution of the process during the formation of the project-specific method.

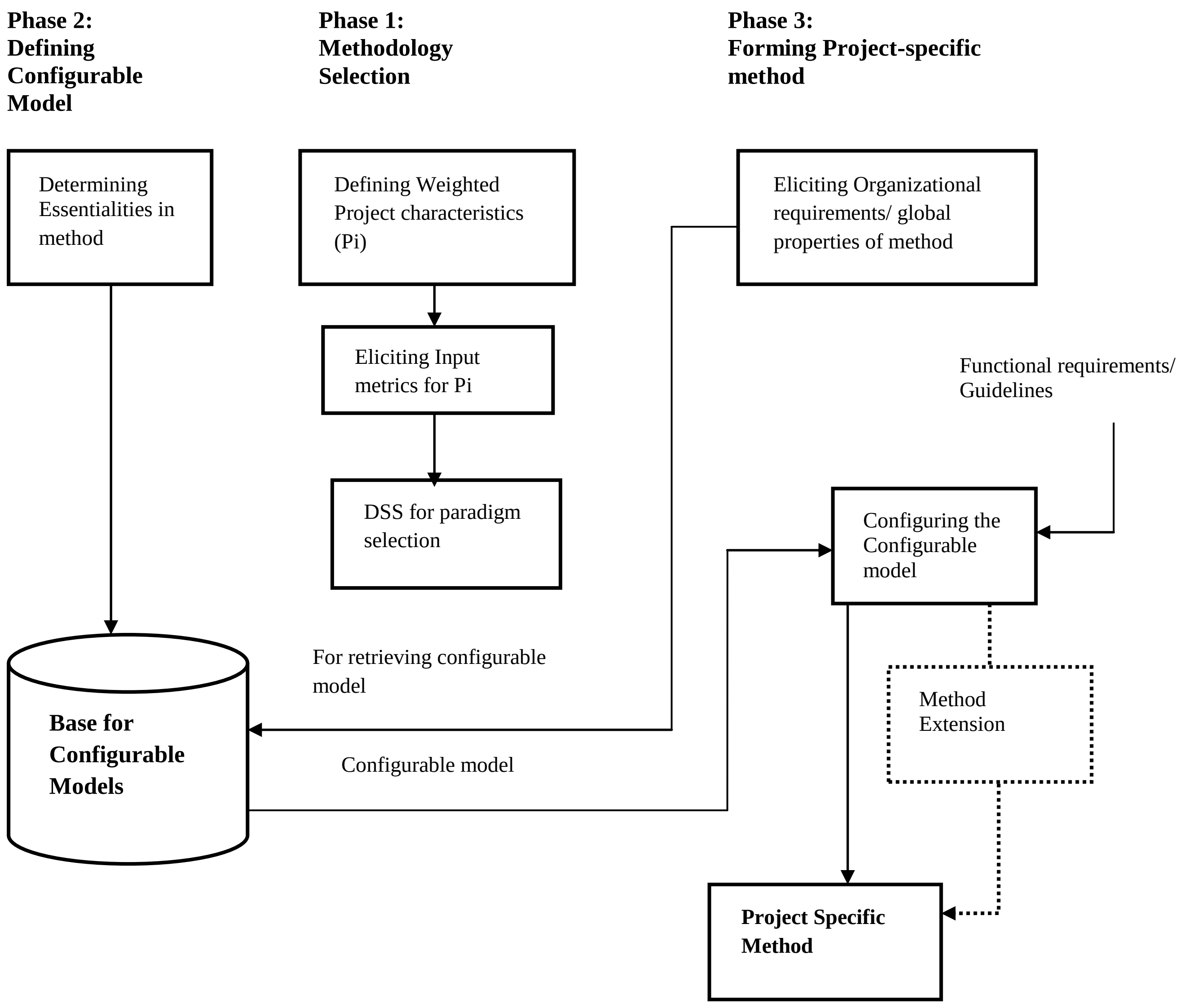

Phase 2: Defining Configurable Model
Phase 1: Methodology Selection
Phase 3: Forming Project-specific method
Determining Essentialities in method
Defining Weighted Project characteristics (Pi)
Eliciting Organizational requirements/ global properties of method
Eliciting Input metrics for Pi
Functional requirements/ Guidelines
DSS for paradigm selection
Configuring the Configurable model
For retrieving configurable model
Base for Configurable Models
Method Extension
Configurable model
Project Specific Method

Figure 6.1: The Generic Framework

In the first phase, Software development has been supported by providing a decision support system for methodology selection. The project characteristics and the input metrics for specific project define the overall context for the decision support system. Information associated at both levels is identified individually and later integrated to predict the appropriate methodological domain.

In the second phase, Methods are defined as configurable models. These configurable models support the notion of commonality and variability and are stored in the method base in the form of pre-made method configurations. The method characteristics of these configurable constructs are mapped with the organizational characteristics or global properties and suitable configurable model is retrieved.

In the third phase, the configurable model is configured to form project-specific method. The organizational requirements or global properties of methods are mapped with the method characteristics to retrieve the appropriate configurable model from the base. The retrieved configurable model is configured in accordance with the functional requirements or generic guidelines. If configurability fails to deliver the requisite method, method extension can be done. This is achieved by EITHER extending the method with some concepts or practices of other methods OR assembling the suitable methods to form project-specific methods.

Contribution of the thesis

A summary of the contributions made in the thesis is as follows:

1. **Decision support system** has been provided for solving the appropriate paradigm selection problem. The proposed set of 22 project characteristics and the input metrics for specific project, define the overall context of the decision support system. Information associated with both levels are identified individually and later integrated to predict the appropriate methodological domain.
2. **Configurable meta model** has been provided used to model the concepts of configurable model, suppress the minutiae of the method configuration process

and facilitate the task of method engineer. The Method Engineer is a role responsible for developing and maintaining the organisation specific method. (Karlsson, 2002).

3. **Providing methods as Configurable Models.** These configurable models have an essentiality attribute to the methods. This attribute defines the criteria of commonality and variability in a method. Hence, a method can now be configured without losing its original essence/purpose.
4. **Discovery the need of guidelines or functional requirements** to configure the project specific method. These generic guidelines or functional requirements empowers the method engineer and allow him to concentrate on configuring project-specific methods.
5. **Widen the scope of software engineering methods** by allowing them to extent their functionality. The methods can be extended by adding some new concepts or practices or can be expanded by assembling with other methods.

Future work and open problems

This thesis throws up a number of directions of future work as follows:

1. The present method configuration process suggests that forming project-specific methods consists of method configuration process and method extension process. Out of these, the present thesis has addressed the method configuration process in generic form. Thus there is a need to develop a generic process for method extension also.
2. There is also a need to integrate the configuration process for traditional methods and for agile methods. Again, the problem here is one is rigid and other is light-weight.

3. Integrating the different software development methodology is also necessary to satisfy the hybrid methodology demands. Literature suggests that there are situation that requires the use of hybrid methodology.
4. The present research focuses on traditional and agile processes of development, however in the future the research can be extended to consider the newer development processes like continual integration and continual delivery.
5. To make the process, of selecting the appropriate software development methodology more efficient, the data set can be extended, more machine learning algorithms can be used and more methodological domains can be considered.

References

1. Abrahamsson, P., Salo, O., Ronkainen, J. and Warsta, J. (2002). Agile Software Development Methods Review and Analysis. *VIT Publications*, Juhani Warsta, University of Oulu.
2. Agile Manifesto (2001) *Manifesto for Agile Software Development*, [online] http://www.agilealliance.org/the-alliance/the-agile-manifesto/ (accessed 14 March 2005).
3. Ahmadi, H., Moaven, S., Rashidi, H. and Habibi, J. (2008). Performing Assembly-Based Method Engineering by Architecture-Centric Method Engineering Approach. *In Second UKSIM European Symposium on Computer Modelling and Simulation,* IEEE Computer Society*, (*pp. 181-186).
4. AlMutairi A. M. and Qureshi M. R. J. (2015). The Proposal of Scaling the Roles in Scrum of Scrums for Distributed Large Projects. In *I.J. Information Technology and Computer Science,* 08, 68-74.
5. Attarzadeh I. (2008). In Project Management practices: The criteria for success or failure, *In communications of IBIMA*, Vol. 1.
6. Avison, D. E., (1996). Information Systems Development Methodologies: A Broader Perspective. *In Method Engineering. Principles of Method Construction and Tool Support. Procs. IFIP TC8, WG8.1/8.2 Working Conference on Method Engineering, 26-28, Atlanta, USA, S. Brinkkemper, K. Lyytinen, R.J. Welke, Eds. Chapman & Hall, London,* (pp. 263-277).
7. Awad, M. A. (2005). *A comparison between agile and traditional software development methodologies*. The University of Western Australia.

8. Basili, V.R. and Rombach, H.D. (1987). Tailoring the Software Process to Project Goals and Environments. *In Proceedings of the Ninth International Conference on Software Engineering*, March 30 – April 2 1987, Monterey, CA.
9. Beck, K. (1999a). Embracing change with extreme programming. *IEEE Computer Society Press*, Vol. 32, No. 10, pp.70–77.
10. Beck, K. (1999b). Extreme programming explained: Embrace change. Reading, Mass, Addison-Wesley.
11. Benefield G. (2008). Rolling out agile in large enterprise. *In Proceedings of the 41st Hawaii international conference on system sciences, HICSS,Hawaii*, IEEE computer society, (pp.461-462).
12. Boehm, B. and Turner, R. (2004). *Balancing Agility and Discipline: A Guide for the Perplexed*. Addison-Wesley, Boston, MA.
13. Booch G., (1994). *Object Oriented Analysis and Design with Applications*. Benjamin/Cummings Publishing Company Inc., Redwood City, CA, Second edition., Rational Method Engine.
14. Brinkkemper, S. (1996). Method engineering: Engineering of information systems development methods and tools. *Information & Software Technology*, 38(4), (pp. 275-280).
15. Brooks, F.P. Jr., (1987). No Silver Bullet: Essence and Accidents of Software Engineering. *IEEE Computer* 20(4), 1987, 10-19.
16. Cameron, J., (2002). Configurable development processes. *Communications of the ACM*, 45(3), 72–77.
17. Chen, P. (1976). The Entity-Relationship Model - Toward a Unified View of Data. *ACM Trans. Database Systems*, 1(1), 9-36.

18. Coad, P. and Yourdon E., (1991). *Object-Oriented Analysis,* 2nd ed., Prentice-Hall, Englewood cliffs, NJ, USA.
19. Coad, P., LeFebre, E. and DeLuca, J. (2000). *Java Modeling in Color with UML: Enterprise Components and Process*, Prentice Hall, Inc., Upper Saddle River, New Jersey.
20. Cockburn, A. (2000). *Writing effective use-cases. The crystal collection for software professionals.* Addison-Wesley professionals.
21. Coplien, J., Hoffman, D. and Weiss, D. (1998). Commonality and Variability in Software Engineering. *IEEE Software*, 15(6), 37-45.
22. Davenport T. H. (1998). Putting the enterprise into the enterprise system. *Harvard Business Review*, 76(4).
23. Debenham, J. and Henderson sellers, B. (2003). Designing agent based process systems-extending the OPF framework. *In V. Plekhanova (eds.), chapter VIII*, (pp 160-190).
24. DeMarco, T., (1978). *Structured Analysis and System Specification*, Yourdon Press, New York.
25. Dowson, M., (1998). Iteration in the software process. *In proceedings of 9th international conference of software engineering*.
26. DSDM consoritium, (1997). Dynamic System Development Method, version 3. Ashford engineering, *DSDM consortium*.
27. Engels G., Lewerentz, C., Schäfer, W., Schürr A. and Westfechtel B., (2010). Graph Transformations and Model-Driven Engineering: The Merits of Manfred Nagl. *Graph Transformations and Model-Driven Engineering*, Springer, (pp. 1-5).
28. Fenton, N. and Bieman, J. (2014). *Software Metrics: A Rigorous and Practical Approach*, Third Edition, CRC press, Taylor and Francis Group.

29. Fitzgerald, B., Russo, N. and O' Kane, T., (2003). Software development method tailoring at Motorola. C*ommunication of ACM*, 46(4), 64–70.
30. Fitzgerald, B., Hartnett, G. and Conboy, K., (2006). Customizing agile methods to software practices at Intel Shannon, *European Journal of Information Systems*, 15(2), 197–210.
31. Fuller A. and Croll P. (2004). Towards a generic model for agile process. *In constructing the Infrastructure for the Knowledge Economy*, Springer, (pp 179-185).
32. Gane C. and Sarson T. (1977). *Structured Systems Analysis: Tools and Techniques*. Mc Donnell Dougles systems Integration Company.
33. Glass, R.L., (2000). Process Diversity and a Computing Old Wives'/Husbands' Tale. *IEEE Software,* 17(4), 128-127.
34. Glass, R.L., (2004). Matching Methodology to Problem Domain. *Communications of the ACM,* 47(5), 19-21.
35. Goldkuhl, G. and Braf, E. (2002). Organisational Ability - constituents and congruencies. *In Coakes E., Willis D., Clarke S. eds., Knowledge Management in the SocioTechnical World,* Springer, London.
36. Goodland, M. and Riha, K., (1999). History of SSADM. *SSADM – an Introduction*. http://www.dcs.bbk.ac.uk/~steve/1/sld005.htm.
37. Green, P. (2012). Adobe premiere pro scrum adoption How an agile approach enabled success in a hyper-competitive landscape. *IEEE 2012,* agile conference.
38. Grosz, G., Rolland, C., Schwer, S., Souveyet, C., Plihon, V., Si-Said, S., Ben Achour, C. and Gnaho C. (1997). Modeling and engineering the requirements engineering process: an overview of the NATURE approach. *Requirements Engineering Journal* (2), 115–131.

39. Gupta, D. and Prakash, N. (2001). Engineering methods from their requirements specification. *Requirements Engineering Journal*, (6), 135–160.

40. Haire, B., Henderson-Sellers, B. and Lowe, D. (2001). Supporting web development in the OPEN process: Additional tasks. *In: Proceedings of 25th Annual International Computer Software and Applications Conference. COMPSAC*, IEEE Computer Society Press, Los Alamitos, CA, USA, 2001, (pp. 383–389).

41. Harmsen, F. and Brinkkemper S. (1993); Computer Aided Method Engineering based on existing MetaCASE Technology. *In Proceedings of 4th European Workshop on the Next Generation of CASE Tools (NGCT '93)*, Sorbonne, Paris, France, Memorandum Informatica, University of Twente, Holland, (pp. 93-32).

42. Harmsen F., Brinkkemper, S. and Oei, H. (1994). 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, (pp.169-194).

43. Harmsen A.F., (1997). *Situational Method Engineering*. Moret Ernst & Young.

44. Haungs, J. (2001). *Pair programming on the C3 project*. Computer 34(2): 118-119.

45. Henderson-Sellers, B., Haire, B. and Lowe, D. (2002). Using OPEN's deontic matrices for e-business. *Engineering Information Systems in the Internet Context*, C. Rolland, S. Brinkkemper, M. Saeki (Eds.), Kluwer Academic Publishers, Boston, USA, (pp. 9-30).

46. Henderson-Sellers, B., Gonzalez-Perez, C., Serour, M. K. and Firesmith, D. G. (2005). Method engineering and COTS evaluation. *ACM SIGSOFT Software Engineering Notes*, 30(4), 1-4.

47. Henderson-Sellers, B**.**, France, B., Georg, G. and Reddy, R. (2007). A method engineering approach to developing aspect-oriented modeling processes based on the OPEN process framework. *In Information and software technology*, 49 (7) 761-773.

48. Highsmith, J. A. (2000). *Adaptive Software Development: A collaborative approach to managing complex system*, NY, Dorset House Publishing.
49. Highsmith, J. A. (2002). *Agile Software Development Ecosystems*, Addison-Wesley.
50. Hossain E., Bannerman P.L. and Jeffrery R. (2011). Towards an understanding of tailoring scrum in global software development: A Multi- Case study. *In ICSSP*, USA.
51. Hunt, J. (2006). Feature driven development. *Agile Software Construction*, Springer, (pp.161–182).
52. Iacovelli, A., Carine, S. and Rolland C., (2008). Method as a Service (MaaS). *In RCIS,* (pp. 371-380).
53. IEEE Standard 610.12 (1990), IEEE Standard Glossary of Software Engineering Terminology.
54. Jackson, M. (1982). Software Development as an Engineering Problem. *Angewandte Informatik* 24(2), 96-103.
55. Jeffery, D.R. and Basili, V.R. (1988). Validating the Tame Resource Data Model. *In Proceedings of the 10th International Conference on Software Engineering*, April 11-15, Singapore.
56. Kahkonen, T. (2004). Agile methods for large organizations – building communities of practice. *In Proceedings of the Agile Development Conference, ADC*, IEEE Computer Society, (pp.2–11).
57. Karlsson F. (2002). *Meta Method for Method Configuration – A Rational Unified Process Case*, Faculty of Arts and Sciences thesis 61, Sweden.
58. Karlsson, F. and Ågerfalk, P.J., (2004). Method configuration: adapting to situational characteristics while creating reusable assets. *Information and Software Technology*, Vol. 46 (9), 629–633.

59. Kelly, S., Lyytinen, K., and Rossi, M., (1996). MetaEdit+: A fully configurable multi-user and multi-tool CASE and CAME environment. In *proceedings of the 8th International Conference, CAISE'96, Advanced Information Systems Engineering,* (pp. 1-21).

60. Kornyshova, E., Deneckere, R., and Salinesi, C., (2007). Method Chunks by Multicriteria Techniques: an Extension of the Assembly-based Approach. *In Proceedings of the IFIP WG 8.1 Working Conference*, Geneva, Switzerland, Springer, (pp. 64-78).

61. Koskinen, H. (1996). Designing multiple processes modelling language for flexible, enactable process models in a MetaCASE environment. *In Proceedings of 7th European Workshop on NGCT,* Seltheit, Farshchian (eds), Greece.

62. Kumar, K. and Welke, R.J. (1992). Methodology Engineering: A Proposal for Situation- Specific Methodology Construction. *In Challenges and Strategies for Research in Systems Development,* W.W. Cotterman, J.A. Senn, Eds. John Wiley & Sons: Chichester, UK, (pp. 257-269).

63. Livermore, J. A. (2008). Factors that significantly impact the implementation of an agile software development methodology. *Journal Of Software* , Vol.3(4).

64. Lundeburg, M., Goldkuhl, G., and Nilsson, A. (1979). A Systematic Approach to information Systems Development. *Information Systems*, 4(1), (pp. 1-12).

65. Marca D. and McGowan, C. (1987). Structured Analysis and Design Technique. McGraw-Hill, ISBN 0-07-040235-3.

66. Marcelloni, F. and Akshi, M. (1997). Applying Fuzzy Logic Techniques in Object-Oriented Software Development. *Object-Oriented Technology, LNCS*, 1357, (pp.295–298).

67. MetaCase Consulting, (1995). *User Manual for MetaEdit Personal 1.2: Customisable CASE Tool to meet your Requirements*, Micro Works, Finland.
68. Miller, D. and Lee J. (2001). The people make the process: commitment to employees, decision making and performance. *Journal of management* (27), 163-189.
69. Moon, M. and Yeom, K., (2005). An Approach to developing Domain Requirements as a Core Asset based on Commonality and Variability Analysis in a Product Line. *IEEE TSE*, 31(7), 551- 569.
70. Nerur, S., Mahapatra, R. and Mangalaraj, G. (2005). Challenges of migrating to agile methodologies, *Communications of the ACM*, 48 (5), 73–78.
71. Nguyen, V.P. and Henderson-Sellers, B. (2003). OPENPC: A tool to automate aspects of method engineering. *In 16th International Conference on Software and Systems Engineering and their Applications,* ICSSEA, Paris, France.
72. Palmer, S.R. and Felsing, J.M. (2002). *A Practical Guide to Feature-Driven-Development*, Prentice- Hall, Upper Saddle River, NJ.
73. Plihon V and Rolland C., (1995). Modeling way-of-working, *In: Proceedings of CASiSE'95*, Springer, Berlin Heildelberg New York.
74. Plihon V. (1996). An environment for method engineering. *Ph.D. Thesis*, University of Paris.
75. Prakash, N., (1994). A process view of methodologies. *In Advanced Information Systems Engineering (CAiSE-94),* Wijers, Brinkkemper and Wasserman (eds.), Springer Verlag, LNCS 811, (pp. 339- 352).
76. Prakash, N., (1997). Towards a formal definition of a method. *Requirements Engineering Journal*, 2(1), Springer Verlag, U.K., 23-50.
77. Prakash, N., (1999). On method statics and dynamics. *Information Systems Journal,* 24(8), Pergamon press, London, 613-637.

78. Prakash, N. (2006). On Generic Method Models. *Requirements Engineering Journal*, 11(4), 221-237.

79. Prakash, N. and Goyal, S.B. (2007). Towards a Life Cycle for Method Engineering. *In Proceedings Eleventh International Workshop on Exploring Modeling Methods in Systems Analysis and Design (EMMSAD'07)*, (pp. 27-36).

80. Prakash, N., Shrivastav M. and Gupta C. (2007). An Intention Driven Method Engineering Approach. *RCIS 2007*, (pp. 281-288).

81. Prakash, N. and Goyal, S.B. (2008). Method architecture for situational method engineering. *In RCIS 2008*. (pp. 325-336).

82. Qumer, A. and Henderson-Sellers, B. (2006). Crystallisation of agility-back to basics. *ICSOFT 2*, (pp.121–126).

83. Qumer, A. and Henderson-Sellers, B. (2008a). A framework to support the evaluation, adoption and improvement of agile methods in practice. *The Journal of Systems and Software*, 81(11), 1899–1919.

84. Qumer, A. and Henderson-Sellers, B. (2008b). An evaluation of the degree of agility in six agile methods and its applicability for method engineering, *Information and Software Technology*, (50), 280–295.

85. Ralyté, J., Rolland, C. (2001). An Assembly Process Model for Method Engineering. *In Proc.of CAiSE*, LNCS 2068, springer-verlag, Berlin, (pp. 267-283).

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

87. Ralyté, J. (2004). Towards Situational Methods for Information Systems Development: Engineering Reusable Method Chunks. *In Proceedings of the*

International Conference on Information Systems Development (ISD'04), Vilnius Technika, (pp.271-282).

88. Ralyté, J., Lamielle, X., Arni-Bloch, N. and Léonard M. (2008). A framework for supporting management in distributed information systems development. *RCIS 2008*, (pp. 381-392).
89. Rizwan, M. and Qureshi, J. (2012). Agile software development methodology for medium and large projects. *IET Software*, 6(4), 358–363.
90. Rolland C., Richard, C. (1982). The Remora Methodology for Information Systems Design and Management. *In proceedings of the IFIP TC8 conference on comparative review of information system design methodologies*, North Holland.
91. Rolland, C., Souveyet, C. and Moreno, M. (1995). An approach for defining ways of working. *Information Systems Journal*, 20(4), 337-359.
92. Rolland, C. and Plihon, V. (1996a). Using generic method chunks to generate process model fragments. *In Proceedings of the 2nd International Conference on Requirements Engineering (ICRE),* IEEE Computer Society Press, 1996, pp. 173 - 180.
93. Rolland, C. and Prakash, N. (1996b). A proposal for context-specific method engineering. *In Method Engineering Principles of Method Construction and Tool Support*, Brinkkemper, Lyytinen and Welke (eds.), Chapman and Hall, (pp. 191-208).
94. Rolland, C., Plihon, V. and Ralyté,J. (1998). Specifying the reuse context of scenario method chunks. *In: B. Pernici, C. Thanos (Eds.), Proceedings of the 10th International Conference on Advanced Information System Engineering (CAISE'98)*, Pisa, Italy, LNCS 1413, Springer, pp. 191–218.
95. Rolland, C. (2009). Method engineering: Towards methods as services. *Software Process Improvement and Practice*, 14(3), 143-164.

96. Ross D.T. and Schoman K.E. (1977). Structured analysis (SA): A language for Communicating Ideas. *IEEE Transactions on Software Engineering*, SE-3(1).

97. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991). *Object oriented modelling and design*, Prentice Hall International, Englewood cliffs, New Jersey.

98. Scharff, C. and Verma, R. (2010). Scrum to Support Mobile Application Development Projects in a Just-in-Time Learning Context. *In Proceedings of the ICSE Workshop on Cooperative and Human Aspects of Software Engineering*. Cape Town, South Africa, (pp. 25–31).

99. Schwaber, K. and Beedle, M. (2002). *Agile Software Development with Scrum*, Nouvelle editions.

100. Si-said, S., Grosz, G. and Rolland, C.(1996). Mentor, a computer aided requirements engineering environment. *In Proceedings of the 8th International Conference on Advanced Information Systems Engineering (CAISE'96)*. LNCS 1080, Springer.

101. Sivanandam, S. N. and Deepa, S. N. (2007). *Principles of soft computing*, John Wiley and sons.

102. Slooten, K. V. and Hodes, B. (1996). Characterizing IS development projects. *In Method Engineering: Principles of Method Construction and Tool Support*, S Brinkkemper (Ed.), *et al.*, Chapman and Hall, London, (pp. 29–44).

103. Smolander, K., TaLvanainen, V. and Lyytinen, K. (1987). How to Combine Tools and Methods in Practice- A Field Study. *In B. Steinholtz, A. Solvberg, and L. Bergman (eds.), Information Systems Engineering.* Berlin, SpringerVerlag, 1987, (pp. 195-214).

104. Smolander, K. (1991). OPRR: A Model for Modelling Systems Development Methods. *In Next Generation of CASE Tools*, IOS Press, Amsterdam.

105. Soffer, P., Golany, B. and Dori, D. (2003). ERP modelling: a comprehensive approach. *Information Systems,* 28(6), 2003.

106. Sommerville I. (1995). *Software Engineering*, Addison Wesley.

107. Sultan, F. and Chan, L. (2000). The adoption of new technology: the case of object-oriented computing in software companies, *IEEE Transactions on Engineering Management* 47 (1) 106–126.

108. Syed-Abdullah, S., Holcombe, M. and Gheorge, M. (2007). The impact of an agile methodology on the well being of development teams. *Empirical Software Engineering* 11, 145–169.

109. Moaven S., Habibi, J. and Ahmadi, H. (2008). Towards an Architectural-Centric Approach for Method Engineering. *In IASTED conference on Software Engineering*, Austria, (pp. 74-79).

110. Stapleton, J. (1997). *Dynamic system development method- the system in practice*. Addison Wesley.

111. Tao L., Fan, J., Li, X and Liu, L. Y. (2006). Observability Statement Coverage Based on Dynamic Factored Use-Definition Chains for Functional Verification. *Journal of Electronic Testing.* 22(3), 273-285.

112. Tuunanen, T. and Rossi, M. (2004). Engineering a Method for Wide Audience Requirements Elicitation and Integrating It to Software Development. *In Proceedings of the 37th Hawaii International Conference on System Sciences*, Hawaii, *(*pp. 1-10).

113. Verheijen, G.M.A. and Bekkum, J., V. (1982). NIAM: an Information Analysis Method. *In: T.W. Olle, H.G. Sol and A.A. Verrijn Stuart (Eds.), Information Systems Design Methodologies: A Comparative Review. Proceedings of the CRIS 82 conference*. North-Holland, Amsterdam.

114. Vlaanderen, K., Jansen, S., Brinkkemper, S and Jaspers, E. (2011).The agile requirement refinery: applying SCRUM principles to Software product management. *Information and Software Technology*, 53(1), 58–70.

115. Weiss, D.M. and Lai, C.T.R., (1999). *Software Product-line engineering: A Family based Sofwtare development Process,* Addison Wesley.

116. Wistrand, K. and Karlsson, F. (2004). Method components – rationale revealed. *In Advanced Information Systems Engineering 16th International Conference, CAiSE,* Proceedings, Springer-Verlag, Berlin, LNCS 3084, (pp.189–201).

117. Yourdon, E., (1989). *Modern Structured Analysis*, Prentice-Hall, London.

118. Zhang, D. (2003). Machine learning and software engineering. *Software Quality Journal*, 11(2), 87–119.

Appendix A – Questionnaire

1. Percentage of requirements for the project-in-hand that is volatile in nature?
    a. Less than 10%
    b. 10-19%
    c. 20-29%
    d. 30-39%
    e. Greater than 39%
2. Will the project –in-hand requires a detailed analysis and high documentation adding the complexity in the development process?
    a. Less than one week
    b. Less than 15 days
    c. Less than one month
    d. Less than six months
    e. More than six months
3. Will the project-in-hand has a high business risk value in terms of the return on investment and customer satisfaction?
    a. Low
    b. High
    c. Very High
4. Will the project-in-hand has a high technical risk value in terms of non-availability of the developer, non-availability of tools etc.
    a. Low
    b. High
    c. Very High

5. Will the project-in-hand has a high Operational risk value in terms of failure of some functionality of the project?
    a. Low
    b. High
    c. Very High
6. Percentage of the project needs to be developed for future modification?
    a. Less than 5%
    b. 5-9%
    c. 10-14%
    d. 15-19%
    e. Greater than 20%
7. Percentage of the project that requires an early release of the complete project?
    a. Complete Project
    b. More than half functionality
    c. Half of the functionality
    d. Less than half
    e. Very minimum amount of modules
8. Time before the first/initial release of the project- in- hand?
    a. 2 months
    b. 4 months
    c. 6 months
    d. 8 months
    e. Greater than 8 months
9. Percentage of requirements known initially for the project-in-hand?
    a. Less than 20%

b. 20-39%

c. 40-59%

d. 60-79%

e. Greater than 79%

10. Amount of requirements gathered for the project- in- hand that are clear and complete and need no further analysis by the developer?

    a. Less than 20%

    b. 20-39%

    c. 40-59%

    d. 60-79%

    e. Greater than 79%

11. Will the project-in-hand requires the scope for major extensions at some points?

    a. No

    b. At some points

    c. At many points

12. Percentage of the functions of the project those are highly dependent on each other?

    a. Less than 2

    b. 2,3

    c. 4,5

    d. 6,7

    e. Greater than 7

13. Amount of experience required by the developer on the tool to be used for the project-in-hand?

    a. Less than 6 months

    b. 6-12 months

   c. 12-18 months

   d. 18-24 months

   e. Greater than 24 months

14. Will the project-in-hand requires platform volatility?

   a. Likely to evolve from one platform to another having different architectures.

   b. Likely to evolve from one platform to another having same architectures.

   c. Likely to evolve from one platform to another having different versions.

   d. No visible change found, but may require at later stage.

   e. Never evolve.

15. Expertise required on the specific application for the project-in-hand?

   a. Less than 12 months

   b. 12-24 months

   c. 24- 30 months

   d. 30-36 months

   e. Greater than 36 months

16. Will the project-in-hand requires some special programmer's capability in terms of Knowledge, Vision and dedication to understand and develop the project?

   a. Very High

   b. High

   c. Low

17. Percentage of Add-on-functions/fancy functions/exciting functions need to be developed.

   a. Less than 20%

   b. 20-39%

   c. 40-59%

    d. 60-79%
    e. Greater than 79%

18. Part of the project that need to be developed in a defined manner with proper documentation.
    a. Less than 20%
    b. 20-39%
    c. 40-59%
    d. 60-79%
    e. Greater than 79%
19. Amount of modules need to be inherited from some parent project for the project-in-hand?
    a. Less than 20%
    b. 20-39%
    c. 40-59%
    d. 60-79%
    e. Greater than 79%
20. Does the project-in-hand will act as a base project for some future projects?
    a. Developed as a base project.
    b. Probability of being used in another project is very high.
    c. It may require additional functionalities at a later stage.
    d. No open project found that requires code of present project-in-hand.
    e. Never be reused.
21. Amount of developer's experience for the project-in-hand.
    a. Less than 6 months
    b. 6-12 months.

c. 12-18 months
d. 18-24 months
e. Greater than 24 months

22. Will the project-in-hand supports communication and interaction among team members?
    a. Very High.
    b. High
    c. Low.

Biography of the Author

Rinky Dwivedi is a research scholar in the Department of Computer Engineering at Delhi technological University (DTU) formerly, Delhi College of Engineering, New Delhi, India. She did her M.E. from Delhi College of Engineering New Delhi in 2008. Her research interests lie in the area of Method Engineering.

She has teaching experience of more than 10yrs. As a research scholar she has taught many courses to undergraduate students. These include Software Engineering, Operating System Programming Fundamentals: C and C++. She has also taken a short course for post graduate students on Advanced Software Engineering.

She is currently working as an Assistant Professor in Maharaja Surajmal Institute of Technology where she is involved in teaching activities as well as various administrative activities.