Chapter 8
Design and Validation of the Experience Base
Model for Software Process Improvement

Great ideas often have three stages of reaction — first, “it’s crazy and don’t waste my time.” Second, “It’s possible, but it’s not worth doing.” And finally, “I’ve always said it was a good idea”
— Arthur C. Clarke
8.1 Introduction

SPI decisions in software engineering organisations are either based on the personal experience of software engineers or are motivated by technological hypes and trends in the field, which often fade away with the passage of time without delivering any long-term tangible benefits to the organisation.

In pursuance of the core objective of the present research, a model has been designed and thoroughly validated so as to support the SPI activities of software engineering organisations with the capturing, modelling, storage and reuse of the software experiences. The proposed model facilitates the software engineering organisations in capturing, accumulating and making explicit the experience through which the organisation can plan and hence make the process improvement effectively in future. The model will help an organisation establish a systematic process improvement by doing away with ad-hoc practices and uninformed decision making.

8.2 Experience Base Model

The model, based on the Experience Factory concept, is named as Experience Base Model (or EBM in short). The EBM captures and manages up-to-date experience about software engineering items or objects which may include any technique, method or tool used for software engineering. Programming languages, methods for systems analysis and design, reusable code libraries, testing techniques, tools for configuration management, version control, and software metrics are typical examples of software engineering reusable objects. These objects also include methods and models used for software development like the Waterfall model, Spiral or Prototyping model etc. Basili et al. (1994) refer to these software engineering objects as software engineering technologies.

Experience about such software engineering objects, referred to as software experience in this model, are the core elements of the EBM. Specifically, the EBM model interweaves the following three kinds of elements:
Design of Experience Base Model for Software Process Improvement

8.3 Conceptual Schema of the EBM Model

The EBM is based on a conceptual model of usage of software experience in software engineering projects. Figure 8.1 exhibits the conceptual schema of the EBM model.

The salient feature of the EBM concept is the distinction of software experience from the software engineering process. A software experience includes any method, technique or tool used for software engineering whereas a software process is defined as the sequence of steps required to develop or maintain software, aiming at providing the technical and management framework for applying methods, techniques, tools and people to the software task (Humphrey, 1995). The SEI defines a software process as “a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products, e.g., project plans, design documents, code, test cases, and user manuals” (SEI, 1993).

a) Definitions of software engineering reusable objects or technologies which act as a basis for establishing systematic and effective software engineering practices in an organisation leading to process improvements, called software experiences.

b) Experience about the impact of the usage of software experience on the quality of the software product or some attributes of software development project, called usage impact.

c) Experience and knowledge about the critical success factors of software objects in specific context situations, called critical context.

The knowledge of the impacts of the usage of software experience helps a software engineering organisation in selecting those software experiences that support the SPI goals of the organisation.

The knowledge about critical success factors guides the SPI teams in identifying the most effective experiences in a given project profile.
Furthermore, the concept of software experience is not restricted to software development technologies but also includes all other technologies that can be used for supporting any software engineering activity. In this sense, project management techniques, tools for recording and analysing customer feedback, organisation structures, KM approaches etc. are also covered under the notion of software experience. Process technologies such as measurement methods, software design methods, software inspection methods, and knowledge elicitation techniques are part of the software experience definition.

The essence of the EBM concept as depicted through the conceptual model (cf. Figure 8.1) can be explained as follows:

1. Software experience directly impacts the process performance and indirectly impacts the software product attributes. Use of relevant and up-to-date experience results in software process improvement and better product quality.

2. One or more experiences, relevant for a process, cause the impact of a process on a software product attribute. The trio of software experience, SE process and software product are the central part of software experience usage impact.
3. Different software experiences and process areas have different patterns of impact on product attributes and project performance.

4. Software process impact on product attributes or project performance varies across different context situations characterised by the relevant critical context factors (CCF).

The major goals of the EBM model are to capture and use experience about the (1) impacts and (2) the critical success factors of software experience application. These are referred to as software experience usage impact (usage impact in short) and software experience usage critical context (critical context in short) respectively.

8.3.1 Software Experience Usage Impact (Usage Impact)

The purpose of ‘Usage Impact’ is to explain for which software engineering process a given software experience can be used and for which quality attribute of a product or performance attribute of a project, this software experience is critical. Usage impact is used for evaluating the success of a software experience, i.e., we can claim that a software experience has been applied successfully for a given process, only if it has been relevant for achieving the desired product quality attribute.

8.3.2 Software Experience Usage Critical Context (Critical Context)

The proposed model lays special emphasis on the ‘critical context’ of the usage of the software experience. Critical context is defined as the collection of all the attributes of the contextual dependencies in a situation in which a certain software experience is applied for a certain process which impacts the success of the software experience usage with regard to a certain success criterion or a certain quality attribute of a product. Therefore, we can say that the concept of critical context is bound to a certain software experience, a software engineering process in which the experience is used, and to a software product quality attribute that provides a measure of the success of the software experience usage.
Critical context can be represented as a tuple \(<\text{critical\_attribute}, \text{value}\>\) for all the relevant success factors. For instance, programming language used, experience of software engineers and project size are the critical attributes. The possible values can be some ordinal scale (e.g. project size = small/medium/large); nominal scale (e.g. language used = C++/Java/C#/…) or some quantitative measure like number of years for ‘experience of software engineer’ attribute.

### 8.3.2.1 Essential versus Critical Context Factors

It is however important to note that every software experience has a set of essential factors that are prerequisite for that experience in the sense that these factors must always be provided when the experience is to be applied, i.e., they are indispensable for the usage of a particular experience. Such context factors are called Essential Context Factors and are different from Critical Context Factors (cf. Figure 8.1). The essential factors associated with a software experience can be the following:

- a) A minimum amount of staff skills and competencies including training and technical knowledge

- b) Provision of required resources like personnel, tools and techniques, and time

- c) Availability of required inputs like documents

The critical context factors are different from the essential context factors as the former are not necessary as such for the usage of the software experience but they are critical for the attainment of the desired goals using the software experience. The critical factors of success for a given experience vary with the processes, product attributes and all other elements of software experience usage impact.

The distinction between essential and critical context factors can be equated with the dichotomy of required versus desired factors. The essential factors of software experience are not specifically addressed in the EBM model because they are always directly linked to a software experience and do not vary with change in the process or desired product attributes.
8.3.3 Software Product

The software product here refers to the quality attributes of the software product that are relevant for the measurement of the success of software experience. The product quality attributes can be defined as per some formal quality model or alternatively can be defined by the organisation itself in its own precise terms. For the purpose of the EBM model, product quality attributes have been defined in accordance with the international standard ISO/IEC 9126-1:2001 (ISO9126-01), now replaced by ISO/IEC 25010:2011 (ISO25010-11) which defines reliability, maintainability, functionality, usability, efficiency and portability as the important quality attributes of a software product.

Furthermore, the term software product also includes project attributes that can act as critical factors of success for the software experience usage. Examples of project attributes can be cost, timeliness, and productivity.

The product factor is very important as SPI goals are defined either in terms of some quality attributes of a software product to be developed or in terms of performance attributes of the overall project.

8.3.4 Software Engineering Process

The term software process represents a software engineering task. Software engineering process has been defined as a set of activities, methods, practices, and transformations that people use to develop and maintain software and the associated products, e.g., project plans, design documents, code, test cases, and user manuals (SEI, 1993). The software engineering process is also defined in terms of the lifecycle phases that cover requirements analysis, software design, coding, testing etc. The software engineering process defined here also includes other tasks such as project management, quality assurance, and software and hardware acquisition.

Software experiences are used to accomplish tasks in some software engineering process.

8.4 High Level Schema of EBM
The high level schema of EBM is shown in Figure 8.2.

![Figure 8.2: High Level Schema of the EBM Concept](image)

It has the following components:

- **Software Experience Base (SEB)** – Experience repository that contains collections of software experience packages.

- **Software Experience Packages (SEPs)** – Core experience items which store experience relevant for the application of software engineering objects.

- Experience management processes for the acquisition, selection and updation of SEPs:
  - An acquisition method that collects and models experience about the usage of software engineering reusable objects.
  - A selection method that deploys SEB (and SEPs contained in it) for supporting SPI initiative in an organisation.
  - An updation method that analyses software projects in which software experience of interest is applied and modifies or refines the SEP of the applied software experience.
8.5 Structure of Software Experience Base

The SEB is a repository which stores and manages SEPs related to software experience. A modular and open schema is used to organise SEPs in a SEB so that they can be easily customised by any software engineering organisation in the light of the organisation’s specific needs and goals of the experience management. The proposed architecture of SEB is general in nature and can be used to implement other kinds of software engineering repositories and experience bases.

Figure 8.3 represents the overall structure of the SEB proposed in the Experience Base Model for SPI in a software engineering organisation.

The following are the main components of a SEB:

- Software Experience Definition
- Software Experience Packages (SEPs)
- Meta Knowledge Base (Meta KB)
- Taxonomies
- Project Repository (PR)
- Maintenance and Control Information
Figure 8.3: Overall Structure of SEB
Software Experience Definition and SEPs are the core components of a SEB. Software experience definition stores the key knowledge items of interest to the EBM user who wants to access it with the help of the associated SEP as every software experience definition is associated with the corresponding SEP in the SEB.

The key terms used in SEB are clearly defined and explicitly represented in Meta KB part of the SEB. Therefore, the Meta KB is the dictionary of key terms which are referenced from all SEPs throughout the SEB.

The software experiences and key terms used in SEB are structured and organised by user-defined taxonomies.

Experience about the project in which a software experience has been used is stored in a separate part of the SEB, called the Project Repository (PR).

Maintenance and Control part of the SEB provides information required for the maintenance and systematic updation of the SEPs and other parts of the SEB.

The components of the SEB are elaborated in the following text.

8.5.1 Software Experience Definition

The software experience definition is the building block of the EBM model and is also the central part of a SEP. Software experience is represented and defined in a detailed and operational way so as to enable a user to apply the experience easily. No specific format is required for the representation of a software experience. Usually software experience definition can be provided as a text document in a highly structured natural language and in a format that suits the particular experience being defined. Process guidebooks, design pattern guidebooks, programming language reference manuals, and templates for SRS are examples of software experience definition.

In some cases, a formal language can also be used to define software experience. Using Backus-Naur form for defining the syntax of a programming language is an example of such a formal language that can also be used to define software experience
in a SEP. But the schema should provide for flexibility and enhancements as software experience definitions may need to be revised from time to time.

8.5.2 Software Experience Package (SEP)

As discussed earlier, SEPs are the core element of the EBM model. They store experience about the impacts and critical context dependencies of the software experience.

The concept of Software Experience Package has been drawn from the ‘Experience Package’ (EP) concept of Basili and Rombach (1991) and Basili et al., 2001b, which is the central element of the Experience Factory approach proposed by them. EP packages the reusable experience to be used in software projects.

The software experience package has been defined as the unit of experience, within a SEB, storing contents related to a software experience.

The overall structure of a SEP is shown in Figure 8.4. It has following four components:

- SEP Header
- Software Experience Definition
- Software Experience Usage Impact
- Software Experience Critical Context

We will now discuss the components of a SEP in detail.

8.5.2.1 SEP Header
The Header part of the SEP stores the information required for the maintenance and administration of the SEPs. This part of the SEP contains:

- **SEP_Id**: SEP_Id is the unique identification tag assigned to a SEP. Every SEP in a SEB is assigned a unique identity number or Id.

- **SEP_Name**: A logical name is assigned to each SEP which gives a clear and unambiguous description about the contents of the SEP.

- **Version**: Required for updation and configuration management.

- **SEP_State**: This slot indicates the status of the SEP. A SEP could be placed in one of the many possible states – initial, confirmed, validated etc. The *initial* state indicates that the SEP is in the preliminary state in the SEB and still has not been confirmed or validated through use. A SEP in *confirmed* state indicates that the said SEP has been used sufficient number of times by software engineers and it has been found useful. The *validated* state means that the SEP has been tested empirically and is fully mature.

  In fact these states reveal the reliability factor of a SEP which is relevant for the selection decision of the SEP by software engineers. The appropriate values depicting different SEP states along with their respective interpretations must be defined during the installation of the SEB.

- **SEB_Name**: Stores the name of the parent SEB.
**Created By/ Owner:** Lists author(s) who created the contents of the SEP.

**Source:** Displays the source(s) of information.

**Last Update/ History:** Reveals the currency of the SEP and it is important for possible revisions of the SEP.

---

<table>
<thead>
<tr>
<th>SEP Header</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SEP_Id</td>
<td></td>
</tr>
<tr>
<td>SEP_Name</td>
<td></td>
</tr>
<tr>
<td>Version</td>
<td></td>
</tr>
<tr>
<td>State</td>
<td></td>
</tr>
<tr>
<td>SEB_Name</td>
<td></td>
</tr>
<tr>
<td>Created by / Owner</td>
<td></td>
</tr>
<tr>
<td>Source</td>
<td></td>
</tr>
<tr>
<td>Last Update / History</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Experience Usage Impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Experience</td>
</tr>
<tr>
<td>SE Process</td>
</tr>
<tr>
<td>SE Product</td>
</tr>
<tr>
<td>Role</td>
</tr>
<tr>
<td>Domain</td>
</tr>
<tr>
<td>Environment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Software Experience Usage Critical Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Critical attribute&lt;sub&gt;1&lt;/sub&gt; Value</td>
</tr>
<tr>
<td>Critical attribute&lt;sub&gt;2&lt;/sub&gt; Value</td>
</tr>
<tr>
<td>: :</td>
</tr>
<tr>
<td>Critical attribute&lt;sub&gt;n&lt;/sub&gt; Value</td>
</tr>
</tbody>
</table>

**Figure 8.4: Structure and Schema of a SEP**

---

### 8.5.2.2 Software Experience Usage Impact

The Software Experience Usage Impact has six parts (cf. Figure 8.4). These are:

- Software Experience
- Software Engineering Process
- Software Engineering Product
- Role
- Domain
- Environment
The central part of a software experience usage impact is the combination of software experience, software engineering process and product/project attributes (cf. Figure 8.1). The Role, Domain and Environment are required to qualify this connection. They make it easy to define software experience impact from different perspectives (roles) or compare them in different application domains or organisational environments.

**Software Experience** slot of the Usage Impact is stored with the name of the software engineering object and provides a link to the software experience definition.

**Software Engineering Process** slot specifies the name of the process or task for which the software experience is to be used.

**Software Engineering Product** slot may store the name of the software product or can specify the product quality attribute for which software experience can be effective.

**Role** defines the formal position from whose perspective the specified relationship is valid. Examples of the values the ‘Role’ slot can fill are Project Manager, Tester, Web designer, DBA and so on.

**Domain** slot defines the business domain for which the software experience usage has been defined.

**Environment** stores the organisational environment, e.g. a specific software company or a department within a company.

It is also important to note that the terms used in the Usage Impact should be precise and unambiguous. In fact they are drawn from a fixed, predefined vocabulary which is made public in the environment of the SEP during the design and installation of the SEB in the organisation. These concepts and definitions are stored in the Meta KB part of the SEB.

**8.5.2.3 Software Experience Critical Context**
The Software Experience Critical Context is specified by a collection of \(<\text{critical\_attribute}, \text{value}\>\) tuples that are called critical context variables. The conceptual model of software experience critical context is presented in UML notation in Figure 8.5.

Critical context is defined in two layers – the Abstract layer and the Operative layer. The Abstract layer (top layer) of the critical context contains the critical factors that are defined in common intuitive or abstract terms, easy to understand by software engineers of the organisation. But such abstract terms can be ambiguous; therefore, the Operative layer (bottom layer) of the critical context defines these abstract contexts in precise and standard defined terms.

8.5.2.3.1 Critical Context Variables

A context variable is a \(<\text{critical\_attribute}, \text{value}\>\) tuple that specifies an attribute of a software project. A context variable has two parts: a) Abstract critical context factor
and b) value(s). Here the abstract critical context factor represents the type definition of the particular attribute and the value(s) part contains the actual value(s) of the attribute. One abstract factor may have one or more values assigned to it. For instance, Project team size and Management commitment are the critical context factors which may assume values as Project team size = large; and Management commitment = high.

Note that the actual values will depend upon the abstract context factors. The values will change with change of the abstract critical context factors.

8.5.2.3.2 Critical Context Factor

A critical context factor represents an aspect of a context situation like Project team size or Management involvement. Both abstract and operative context factors have following parts:

- CCF_No.
- CCF_Title
- CCF_Description
- Range of values

Each context factor is assigned a unique Id called the CCF_No. and a logical title, called CCF_Title. The CCF_Description slot contains the detailed definition of the context factor. The range of all possible values for the given context factor is specified in the ‘Range of values’ slot.

There are two kinds of critical context factors – a) Abstract context factors and b) Operative context factors. Abstract context factors are part of the top layer of the software experience critical context whereas Operative context factors are part of the bottom layer of the critical context part of the SEP.

8.5.2.3.3 Abstract Context Factors
An abstract context factor has all the components of a context factor along with one more element (operator) called ‘Conversion.’ An abstract context factor specifies the abstract view that an experienced software engineer has on a software development project. We use abstract context factors to define critical factors like large project size or high management commitment that can be quite intuitive and ambiguous for a person who is not well versed with a particular organisation environment. For instance, large project size may mean different things to software engineers coming from different software engineering organisations.

### 8.5.2.3.4 Conversion

The ‘Conversion’ operator disambiguates the abstract context factor so that it becomes operational for use in the selection of software experience. It maps an abstract context factor like Project size to one or more operative context factors like number of software engineers in the team and the number of development sites. Every abstract context factor must have a conversion. Table 8.1 shows the example of conversion between abstract context factor and operative context factors.

<table>
<thead>
<tr>
<th>Abstract Context Factor</th>
<th>Operative Context Factors</th>
<th>Conversion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project size</td>
<td>No. of staff in project + No. of development sites</td>
<td>Semantic or logical</td>
</tr>
<tr>
<td>Programming language</td>
<td>C++, Java, C#, ...</td>
<td>Direct</td>
</tr>
<tr>
<td>Amount of code reuse</td>
<td>Total No. of modules and No. of modules reused</td>
<td>Mathematical</td>
</tr>
<tr>
<td>Software engineer’s experience</td>
<td>Years of software development experience</td>
<td>Semantic or logical</td>
</tr>
</tbody>
</table>

### 8.5.2.3.5 Operative Context Factors

An operative context factor has the following four basic components of a general context factor:

- CCF_No.
- CCF_Title
- CCF_Description
- Range of values
One or more operative context factors are linked with one abstract context factor through ‘Conversion.’ Also one operative context factor may be used in several conversions.

In nutshell, the software experience critical context is a vector of critical context variables consisting of one abstract context factor and a value assigned to it out of the context factor’s ‘Range of values’ slot.

8.5.3 Meta KB

Every term used in a SEP should be defined precisely and without any ambiguity. Explicit representation of all the terms, concepts and definitions related to software experience makes the references from all the SEPs possible. This is done in order to avoid redundancies and resultant inconsistencies associated with duplicate definitions of the key terms used in SEPs. The Meta KB part of the SEB stores the definitions of all key terms and concepts used in SEPs and other parts of the SEB. The concepts defined in the Mata KB include: software engineering process, product attributes, roles, domain and environment part of the SEPs. Also the explicit definitions of the SEP_State are stored in Meta KB.

Software experiences are also represented in a similar manner. They are treated as separate artefacts and their definitions are usually much more complex than the definitions of other concepts used in a SEB. As mentioned earlier, software experience can be defined using simple text or alternatively can also be defined as structured text. Figure 8.6 gives an example of a process software experience using structured text.

Likewise product attributes can be defined using some formal standard like ISO/IEC 9126-1:2001 (replaced by ISO/IEC 25010:2011) standard of product quality attributes or Software Engineering Book of Knowledge\(^2\) etc. For example, the ISO/IEC 9126-1 standard defines six overall product quality attributes – reliability, maintainability, functionality, usability, efficiency and portability, which can be used in the Meta KB.

\(^2\)www.swebok.org
### Process Description
An overview giving quick reference for the software experience

#### Documents and Artefacts:
- Input
- Output
- Intermediary
Specifications of various documents involved in the process

#### Roles involved
Various roles involved in the process are specified

#### Process Outline
Process steps are outlined

#### Process Start Condition(s)
Condition(s) for entry to process are specified

#### Process Stop Condition(s)
Condition(s) for exit from process are specified

#### Defined Process Steps
- Goal of the step
- Input products
- Output products
- Intermediary products
- Process-step start condition
- Process-step stop condition
- Role
- Sequence of activities
- Next process step
Defined for each step of the process

#### References to Literature
Information on further knowledge sources

#### Document Templates
Templates of documents used in the process

#### Software Support Tools
Software tools that can be used in the process

**Figure 8.6: Definition of Process Software Experience in Structured Text**

### 8.5.4 Taxonomies

To organise a large amount of concepts contained in the software experience definitions, user-defined taxonomies are used. Taxonomies also ease the retrieval and search of SEPs. Taxonomies are specified for each type of concept involved in a SEP, i.e., taxonomies are defined for software engineering process, product attributes as well as for software engineering objects or technologies. A taxonomy is divided into categories arranged in a hierarchy. Every category contains elements of the concept type of the taxonomy. Taxonomies can be defined for the following types of concepts:

- Software engineering objects or technologies
- Software engineering process
- Software engineering product attributes
- Critical context factors
- Others like Roles, Domain, and SEP_State
An organisation can either implement predefined standard taxonomies or can define its own taxonomies based on the concepts and their classifications popular in the organisation.

### 8.5.5 Project Repository (PR)

This part of the SEB contains information on the use of different software experiences in different software development projects. This information helps the user in selecting a particular software experience for a software development project or for the SPI initiative. The software engineer can review the experience from the application of a software experience object or technology in the previous projects and can base the technology selection decision on it.

PR is made up of two components – Software Experience Usage Case and Project Profiles (cf. Figure 8.7). A Software Experience Usage Case connects a software experience with a Project Profile in which it has been used. The Software Experience Usage Case contains the concept of *Experience_use_result* (Sx_Use_Result) which indicates whether the application of software experience in the project had been successful or not.

Project Profile part of the PR contains information like documents, templates and artefacts of the projects handled, measurement data etc.

![Figure 8.7: Representation Schema of Project Repository in UML Notation](image)

### 8.5.6 Maintenance and Control Information
This part of the SEB contains administrative information required for control and maintenance of SEB. It contains the following items:

**SEB_Header:** Contains Name, Id, Version of SEB etc.

**SEB_State:** Preliminary/Advanced/Mature etc. (organisation-defined taxonomy)

**SEB_Owner:** Access rights are specified; Author(s) are identified.

**SEB_History:** Change history should be logged by administrator

### 8.6 Representation Schema of the EBM

A conceptual object model and representation schema of the Experience Base Model is presented below in the form of UML class diagrams. The object model illustrates additional detailed aspects of the EBM representation schema. The object model and the representation schema are conceptual in nature and serve as a guide for the development of the EBM in an organisation. The class diagrams are abstract in nature. The minimum required attributes have been included in the respective classes. Functionality can be added at the implementation stage. Being the conceptual model, the implementation details have not been discussed.

#### 8.6.1 Software Experience Base (SEB)

The SEB is the container object of all other objects. The representation schema of SEB is shown in Figure 8.8. It consists of the following six parts:

1. Software Experience Definition
2. Software Experience Package (SEP)
3. Meta KB
4. Taxonomy System
5. Project Repository (PR)
Software Experience Definition, SEP and Meta KB are mandatory. This is indicated by the cardinality of the concepts that require at least one such instance.

Taxonomy System and Project Repository are optional.

Every SEB always contain one instance of Maintenance and Control Info.

8.6.2 Software Experience Definition

Software Experience Definition is not specified further because no specific format is required for defining Software Experience. It can be defined as a natural language text document.

8.6.3 Software Experience Package (SEP)
SEP has four parts: SEP Header, Software Experience Definition, Software Experience Usage Impact and Critical Context. The representation schema of SEP is shown in Figure 8.9.

8.6.4 SEP Header

The SEP Header stores SEP Id, Name, Version, State, SEB_Name, Author, Source and Last Update Information about the SEP. Each part has a textual representation.

8.6.5 Software Experience Usage Impact

The Usage Impact has six parts:

1. Software Experience
2. SE Process
3. SE Product
4. Role
5. Domain
6. Environment

The instance of each part is definitions of key terms and concepts taken from Meta KB, which consists of a name and a textual definition.
8.6.6 Software Experience Usage Critical Context

Software Experience Critical Context includes the Abstract Context Model and the Abstract Context Variablisation (cf. Figure 8.10).
The Abstract Context Model, like a data type definition, defines the structure of the context model. It consists of a list of abstract context factors.

The Abstract Context Variablisation, like a value of a data type, assigns concrete values (i.e. context variables) to the abstract context factors.

One Context Model can be part of more than one SEP. But a Context Variablisation is always related to exactly one SEP.

8.6.7 Context Model and Context Variablisation

The Context Model and the Context Variablisation are abstract super-classes of abstract and operative context model and context variablisation (cf. Figure 8.11).

One Abstract Context Model can be variablised by more than one Context Variablisation.

A SEP does not have an Operative Context Variablisation. The Operative Context Variablisation is only included in the Project Repository for variablising a project context.
8.6.8 Abstract and Operative Context Model

Context Models consist of Critical Context Factors. Also it contains the definition of a value type that specifies the domain of possible values. Here, two different kinds of values are defined: Range type and Enumeration type. There could also be additional value types defined in a SEB.

8.6.9 Context Variable

A Context Variable associates a Critical Context Factor with values.

Context Variablisation is a vector of values, one for each critical context factor to be variablised.
8.6.10 Critical Context Factor

A Critical Context Factor contains four parts: CCF_No., CCF_Title, CCF_Description and Range of values. The CCF_Description is a textual definition. The ‘Range of values’ includes both range type as well as enumeration type.

8.6.11 Meta KB

The Meta KB is a repository of all the terms and concepts stored in different parts of the SEB. Representation Schema of Meta KB is shown in Figure 8.12.

Every key term has the Name and textual Definition of the instance.


8.6.12 Taxonomy System

A Taxonomy system includes one or more taxonomy, which consist of a hierarchy of categories (cf. Figure 8.8).

Each Item type of Software Experience Usage Impact as well as Context Factors are organised in taxonomies. The Taxonomy system is used to define several alternative taxonomies for such concepts.
Figure 8.12: Representation Schema of Meta KB
8.6.13 Project Repository (PR)

Project Repository is the storehouse of information about the projects in which experiences have been used. This information is critical for the evaluation and re-packaging of SEPs. Most of the information stored in PR part of the SEB stems from the experience selection process and is not contained in other parts of the SEB. The representation schema of PR is shown in Figure 8.13.

PR is made up of two components – Project Profile and Software Experience Usage Case. Software Experience Usage Case documents that an experience of interest has been applied in a specific project. It has `Sx_Use_Result` component which stores, in natural language text, the result of the application of the experience in a particular project. It documents whether the use of software experience was viewed successful or not.

The Software Experience Usage Critical Context is the Operative Context Variablisation of the project during the selection of the Software Experience. It is a result of the experience selection process as documented in Software Experience Selection Log.

The Software Experience Selection Log contains the main results of the experience selection process which are important for evaluation of the experience usage success.
It documents the information based on which the experience was considered fit for the specified Project Profile.

### 8.6.14 Maintenance and Control Info

Every SEB has one Maintenance and Control Info module which is required for the administrative purpose. It has following four parts (cf. Figure 8.8):

1. **SEB_Header**
2. **SEB_State**
3. **SEB_Owner**
4. **SEB_History**

**SEB_Header** contains information like Name, Id, Version, and storage characteristics of the SEB.

**SEB_State** stores the current state of the SEB. It draws its value from the pre-defined organisation taxonomy.

**SEB_Owner** is used to store the author(s) name(s) who are responsible for the creation, updation and maintenance of the SEB in the organisation. Also the access rights provided to various owners can be specified in case of a hierarchical team structure.

**SEB_History** is used to document the temporal dimensions of the SEB.

### 8.7 Validation of the EBM Model

An empirical validation of the proposed model requires that a software engineering organisation be chosen wherein the complete SEB can be developed, evolved and used for a considerable period of time. This presumes the existence of software experience repositories in the organisation in some form or the other. The empirical survey conducted (findings of which are presented in chapters 5 and 6) revealed the fact that no such organisation-wide experience repository or EB exists in any software
engineering organisation. This made the empirical validation or testing of the proposed model difficult. Hence an exploratory case study based on literature survey was chosen as the strategy for validating the EBM model proposed in this thesis. For this purpose, SEI C4 Software Technology Reference Guide (STRG) was found from the exhaustive literature search and was chosen for validation of the EBM model (SEI C4, 1997).

8.7.1 Case Study for Model Testing

SEI C4 Software Technology Reference Guide is a project and technical report based on this prototype, published by Software Engineering Institute, Pittsburg. STRG is a repository of 61 software engineering technology definitions, organised in two taxonomies – a) a taxonomy of software engineering processes in which a technology can be applied and b) a taxonomy of software product quality that the particular software engineering technology influence.

Examples of software technologies indexed in STRG include Requirements tracing, Java, cleanroom software engineering, cyclometric complexity, digital signatures, and client/server architectures. The STRG provides template for technology description which comprises of Name, Status, Note, Purpose, Origin, Technical detail, Usage considerations, Maturity of the technology as well as Costs and limitations, Dependencies, Alternatives, References and Information sources for a software engineering technology.

A structured natural language text has been used to describe each software engineering technology and its related context, typically spanning from four to six pages.

8.7.2 Justification of STRG

The choice of STRG for the validation of EBM model was governed by the following reasons:
a) The organisation schema of STRG is quite analogous to the core part of SEP schema introduced in this thesis.

b) Definitions of software technologies in STRG contain information about context dependencies of the technology, matching the software engineering critical context part of the SEPs.

8.7.3 Objectives of Model Testing

The major objectives of the model validation were to

a) Evaluate the structure of the SEB so as to test its ability to accommodate a large number of SEPs.

b) Test the adequacy of the representation schema with a considerably large number of varied SEPs.

8.7.4 Testing Methodology

The EBM model was tested by using STRG’s Technology Definitions to model the complete SEB. Following three-step methodology was adopted to model the SEB and to validate the EBM model.

a) The software experience usage impacts were derived from the STRG. For this purpose, STRG’s product quality taxonomy as well as process taxonomy was used as each technology definition in the STRG is linked to a collection of product qualities and processes. For each such combination, a software experience usage impact was defined in the first step of testing.

b) In the second step, the technology descriptions of the STRG were reviewed to identify the context information about each technology in the STRG. Specifically, tags such as Purpose and Origin, Technical detail, Usage considerations, and Costs and limitations of each technology were scanned to extract information about critical context factors of each software engineering
technology, which were then translated in context variables after de-contextualising it.

c) Finally, the context variables of each software engineering object were integrated into a context model for the software experience usage impact.

However it is important to mention here that the SEPs so developed from the STRG did not distinguish between operative and abstract context factors as the information contained in the STRG expresses the entire software engineering domain in generic terms. Hence there was no need for defining abstract and operative context factors separately. But when the generic SEB is to be implemented in a specific software engineering organisation, it has to be tailored to the organisation-specific environment and accordingly context factors have to be defined in terms of abstract and operative layers.

**8.7.5 Results of Model Testing**

Modelling the software engineering technologies of SEI C4 Software Technology Reference Guide using the EBM constructs proved that the organisational structure of SEB and the representation schema of SEP are effective and valid. It is proved that a large and diverse number of information about software engineering technologies along with related context can be easily transferred into a collection of coherent SEPs.

As a result of the modelling of STRG’s technology definitions into SEPs, taxonomies for software experience (cf. Table E1 in Appendix E), product attributes (cf. Table E2 in Appendix E), software engineering process (cf. Table E3 in Appendix E), and critical context factors (cf. Table E4 in Appendix E) were developed.

**8.8 Advantages/ Features of the Proposed Model**

The EBM model has many unique features which are explained in the following.
1. The model has been described in an open and general representation schema that can be implemented by any software engineering organisation by customising it in terms of the organisation’s specifics.

2. The EBM model has been presented using a modular representation schema and tabular structure based on critical_attribute and value pairs. This representation makes the model easy to understand and automate.

3. EBM is intended to be used by software engineers in a SE organisation. Software engineers in a particular organisation share intuitive terminology for the software engineering concepts and terms which imply intended meanings to the members of that organisation but may be ambiguous for newcomers to the company or project or to some other SE organisation. EBM takes account of this difference between the views of the concepts and terms referred to by different terminologies in software engineering organisations. This gap is filled by providing two-layer model of the software experience critical context.

4. The model supports all possible critical context factors associated with any software experience and allows an organisation to define and include its own context factors.

5. Clearly defined formal representation schema has been provided for various components of the EBM, making it simple to implement it.

6. The model has been based on standard and universally accepted standards and guidelines like Software Engineering Book of Knowledge and ISO/IEC 9126-1 Standard.

7. The EBM model has been woven into the general EM framework. Routines for experience acquisition and SEP selection have been defined.

8. EBM supports reuse of software experience across organisational boundaries.
9. Though the proposed model is based on the concept of *Experience Factory* but the model is much better than the EF concept in the following ways:

a) The EBM provides specialised support for the systematic selection of software experience during SPI initiative being taken by an organisation.

b) The original EF concept does not specify any representation schema for the application purpose for which a software experience can be applied (i.e., software experience usage impact) or the context factors which are critical for the success of this purpose (i.e., software experience critical context). Formal detailed representation schemas have been explained in EBM so as to guide the software engineering organisation planning for SPI activities using experience management.

c) EF concept does not make any provision for accommodating abstract and operative views of the critical contexts needed for knowledge acquisition from human experts.

### 8.9 Future Scope of the EBM

The EBM model can be extended in future to cover more and more parts of a software engineering organisation and to provide support for more software engineering activities. The EBM can be extended by widening its application scope by adding other kinds of artefacts like the following:

- Lessons learnt about software engineering objects or technologies
- Measurement data
- Entire project documents
- Online tutorials about software experience
- Online guidance of software engineering activities
- Reference materials for software experience
- Reusable artefacts to support experience usage
- Yellow pages
The EBM so extended in future can grow into a complete software engineering information system of a software engineering organisation. This will help the software engineering organisations achieve the same maturity level as other engineering sciences of today. This is also the overall goal of EM in software engineering. The proposed repository structure and representation schema can be used to extend and migrate from existing knowledge/experience repositories to a more comprehensive community-wide experience base in an organisation.