Modelling Critical Context in Software Engineering Experience Repository: A Conceptual Schema

Neeraj Sharma
Associate Professor
Department of Computer Science
Punjabi University, Patiala (India)

ABSTRACT

The experiential knowledge needs to be stored in some formalized system for the purpose of its reuse. Such knowledge repositories are referred to as experience bases in software engineering. Though the concept of software engineering knowledge repository is often discussed and advocated by the proponents of knowledge and experience management practices but there are no concrete studies available on how to structure and model such experiences in software engineering environments. Moreover, the critical context associated with software experience is often not captured and stored for the want of the formalization schema to model such context. In this paper, the conceptual schema of the experience repository is described. The paper also explains the components of the critical context and presents a conceptual model in UML class diagram.

General Terms

Software Engineering, Experience Repository

Keywords

Experience Base, Critical Context, Experience Factory

1. INTRODUCTION

The Software Engineering Experience Repository 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. These software engineering objects are also referred to as software engineering technologies [6]. Experience about such software engineering objects, referred to as software experience in the proposed model, are the core elements of the experience repository. Specifically, the proposed model interweaves the following three kinds of elements:

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 improvements in process, 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 process improvement goals of the organisation.

The knowledge about critical success factors guides the process improvement teams in identifying the most effective experiences in a given project profile.

2. LITERATURE SURVEY

In software engineering, Experience Factory (EF) approach is the most popular solution for capturing and managing the software engineering knowledge into knowledge repositories. The Experience Factory is a logical and/or physical organisation that supports project developments by analysing and synthesising all kinds of experience, acting as a repository for such experience, and supplying that experience to various projects on demand [6]. The focus of the EF is primarily on experiences as opposed to generalised knowledge (e.g. rules, heuristics etc.). The captured experiences are stored into a repository of experiences, called the Experience Base (EB), which stores the software engineering experience in a packaged form, called Experience Packages. The EF is an organisation that supports software projects conducted in, what we call the project organisation. Project organisation can be an independent full-fledged software development organisation or a unit or department within an organisation developing software [3]. In particular, the EF analyses and synthesises all kinds of experiences drawn from these projects, acts as a repository for such experiences by documenting, storing, qualifying, and updating them using a experience base, and supplies those experiences back to projects on demand. Literature is abound with many successful stories of EF implementations in many organisations, e.g. [11], [12] and [14]. Furthermore, experience factories covering different aspects of software engineering process are reported, e.g. [1] and [2].

We also find mention of domain-specific experience factories which include software development cost estimation, e.g. [10], data mining applications, e.g. [5], developing CBR applications, e.g. [4], [9], and ontology deployment, e.g. [13]. Software engineering knowledge representation and technology to support the development of software engineering repositories have been the main focus of research in this area. How to document and represent experiences in repository finds special emphasis in research.

There are studies which forewarn about the potential barriers in implementing EF approach in software engineering organisations. They conclude that EF implementations require a significant investment of time and efforts. Need to leverage alternate approaches to distribute knowledge have been stressed. A short term solution to this problem has been suggested in the form of 'Answer Garden' approach. An approach called the 'Knowledge Dust to Pearls' has been explained in [7]. This approach addresses some of the issues with knowledge management in software development and allows the software experience repository to become more useful more quickly than traditional approaches. This approach adds the short-term-oriented features from the 'Answer Garden' approach to the long-term and sophisticated features of the EF approach.

3. CONCEPTUAL SCHEMA OF EXPERIENCE REPOSITORY

The essence of the proposed model for experience repository can be explained as follows:

1. Software engineering experience directly impacts the process performance and indirectly impacts the software product attributes. Application 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 characterized by the relevant critical context factors (CCF).

The major goals of the experience repository 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.

3.1 Software Experience Usage Impact

The purpose of software experience '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.

3.2 Software Experience Usage Critical Context

The proposed schema 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.

4. SOFTWARE EXPERIENCE CONTAINER

Software Experience Container (SEC) is the core element of the experience repository. They store experience about the impacts and critical context dependencies of the software experience. The concept of Software Experience Container has been drawn from the 'Experience Package' (EP) concept of [6] and [7], 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 container has been defined as the unit of experience, within an experience repository which stores the contents related to a software experience.

The overall structure of a SEC is shown in Figure 1. It has following four components:

SEC Header

Software Experience Definition

Software Experience Usage Impact

Software Experience Critical Context

4.1 SEC Header

The Header part of the SEC stores the information required for the maintenance and administration of the SECs. This part of the SEC contains:

SEC_Id

SEC_Name

Version

SEC_State

Repository_Name

Created by/ Owner

Source

Last Update/ History

SEC_Id is the unique identification tag assigned to an experience container. Every SEC in a repository is assigned a unique identity number or Id. A logical name is assigned to each SEC which gives a clear and unambiguous description about the contents of the SEC in SEC_Name. Version is required for updation and configuration management. SEC_State slot indicates the status of the SEC. A SEC could be placed in one of the many possible states – initial, confirmed, validated etc. The initial state indicates that the SEC is in the preliminary state in the repository and still has not been confirmed or validated through use. A SEC in confirmed state indicates that the said SEC has been used sufficient number of times by software engineers and it has been found useful. The validated state means that the SEC has been tested empirically and is fully mature. In fact these states reveal the reliability factor of a SEC which is relevant for the selection decision of the SEC by software engineers. The appropriate values depicting different SEC states along with their respective interpretations must be defined during the installation of the experience repository. Repository_Name stores the name of the parent experience repository. Created By/ Owner lists author(s) who created the contents of the SEC. Source displays the source(s) of information, and Last Update/ History reveals the currency of the SEC and it is important for possible revisions of the SEC.

SEC Header	
SEC_Id	
SEC_Name	
Version	
State	
Repository_Name	
Created by / Owner	
Source	
Last Update / History	
Software Experience Usage Impact	
Software Experience	
SE Process	
SE Product	
Role	
Domain	
Environment	
Software Experience Usage Critical Context	
Critical attribute$_1$	Value
Critical attribute$_2$	Value
:	:
Critical attribute$_n$	Value

Fig 1: Structure and Schema of SEC

4.2 Software Experience Usage Impact

The Software Experience Usage Impact has six parts (cf. Figure 1). 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. 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 SEC during the design and installation of the experience repository in the organisation. These concepts and definitions are stored in the Meta KB part of the repository.

4.3 Software Experience Critical Context

The Software Experience Critical Context is specified by a collection of <critical_attribute, value> tuples that are called critical context variables. The conceptual model of software experience critical context is presented in UML notation in Figure 2.

5. CRITICAL CONTEXT IN EXPERIENCE REPOSITORY

Critical context can be represented as a tuple <critical_attribute, value> for all the relevant success factors. For instance, programming language used, experience of software engineers and project size are the critical attributes.

5.1 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. The essential factors associated with a software experience can be a minimum amount of staff skills and competencies including training and technical knowledge; provision of required resources like personnel, tools and techniques, and time; and availability of required inputs like documents etc.

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 proposed schema because they are always directly linked to a software experience and do not vary with change in the process or desired product attributes.

5.2 Abstract layer and the Operative layer

The Software Experience Critical Context is specified by a collection of <critical_attribute, value> tuples that are called critical context variables. The conceptual model of software experience critical context is presented in UML notation in Figure 2. Critical context is defined in two layers – the Abstract layer and the Operative layer. The Abstract 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 of the critical context defines these abstract contexts in precise and standard defined terms.

5.3 Critical Context Variables

A context variable is a <critical_attribute, 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.

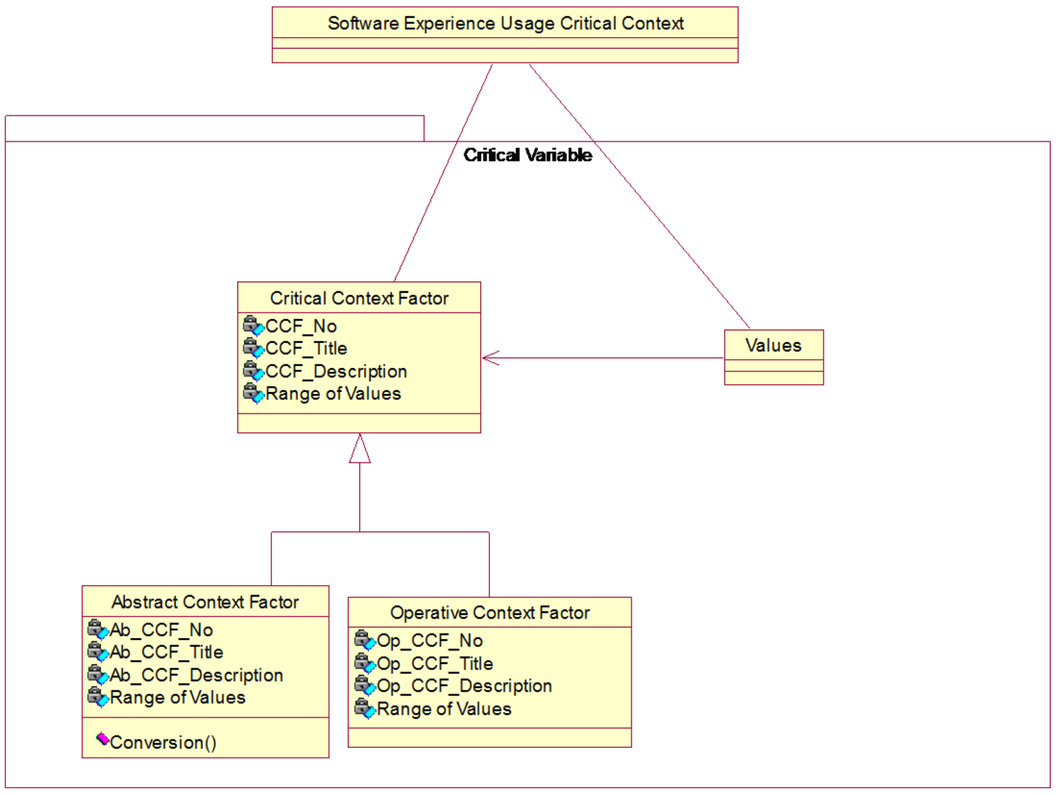

Fig 2: Conceptual Model of Software Experience Critical Context

5.4 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 SEC.

5.4.1 Abstract Context Factors

An abstract context factor has all the components of a context factor along with one more element, called 'Operator.' 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.

5.4.2 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 'Operator.' 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.

5.4.3 Operator

The '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 an Operator.

6. REFERENCES

[1] Althoff, K.-D., Birk, A., Wangenheim, C.G.V., and Tautz, C. 1998. CBR for experimental software engineering. In Proceedings of the Case-Based Reasoning Technology - From Foundations to Application, Chapter 9, 235-254. Springer, Heidelberg.

[2] Althoff, K.-D., Bomarius, F., Mller, W., and Nick, M. 1999. Using case based reasoning for supporting continuous improvement processes. In Proceedings of the 12th German Workshop on Machine Learning.

[3] Althoff, K.-D., Bomarius, F., and Tautz, C. 2000. Knowledge Management for Building Learning Software Organizations. Information Systems Frontiers, vol. 2, 349-367, Kluwer Academic Publishers.

[4] Althoff, K.-D., Nick, M., and Tautz, C. 1999. CBR-PER: A tool for implementing reuse concepts of the experience factory for CBR systems. In Proceedings of the 7th German Conference on Knowledge Based Systems (XPS99).

[5] Bartlmae, K. 1999. An experience factory approach for data mining. In Proceedings of the 2nd Workshop in Data Mining and Data Warehousing as Basis of Modern Decision Support Systems.

[6] Basili, V.R., Caldiera, G., and Rombach, H. 1994. The Experience Factory. In Marciniak, J. (ed.) Encyclopedia of Software Engineering, vol. 1, Chapter X, 468–476, John Wiley & Sons, NJ, USA.

[7] Basili, V.R., Lindvall, M., and Costa, P. 2001. Implementing the Experience Factory Concepts as a Set of Experience Bases. In Proceedings of the 13th International Conference on Software Engineering & Knowledge Engineering, 102-109. Knowledge Systems Institute.

[8] Basili, V.R., and Rombach, H.D. 1991. Support for Comprehensive Reuse. IEEE Software Engineering Journal, 22 (4), 303-316.

[9] Bergmann, R., Breen, S., Goker, M.; Manago, M., and Wess, S. 1999. Developing Industrial Case Based Reasoning Applications - The INRECA Methodology. LNAI, 1612, Springer-Verlag.

[10] Finnie, G., Wittig, G., and Desharnais, J.-M. 1997. Estimating software development effort with case based reasoning. In Proceedings of the 2nd International Conference on Case Base Reasoning, 13–22, Springer-Verlag.

[11] Henniger, S. 1997. Capturing and formalizing best practices in a software development organization. In Proceedings of the 9th International Conference on Software Engineering and Knowledge Engineering.

[12] Houdek, F., Schneider, K., and Wieser, E. 1998. Establishing Experience Factories at Daimler-Benz: An Experience Report. In Proceedings of the 20th International Conference on Software Engineering, 443-447.

[13] Kalfoglou, Y., and Robertson, D. 2000. Applying experienceware to support ontology deployment. In Proceedings of the International Conference on Software Engineering and Knowledge Engineering.

[14] Koennecker, A., Jeffery, R., and Low, G. 2000. Implementing an experience factory based on existing organizational knowledge. In Proceedings of the Australian Software Engineering Conferenc.