Knowledge Services for Experience Factories

Eric Ras, Jörg Rech, Sebastian Weber
Division Competence Management
Fraunhofer IESE
Fraunhofer-Platz 1
67663 Kaiserslautern, Germany
{Forename.Surname}@iese.fraunhofer.de

Abstract: The Experience Factory (EF) is an infrastructure designed to support experience management (e.g., the reuse of experience) in software organizations. It supports the collection, processing, management, analysis, and dissemination of experiences made on the job. One key issue of experience management is the aggregation of these documented experiences into more valuable patterns and laws. In this article, we describe how the different activities in an experience factory can be supported by knowledge services. Knowledge services are especially valuable for maturing activities, such as formalization and generalization. We describe how to leverage Web 2.0 concepts as features of an experience management system in order to implement the knowledge services.

1 Introduction

Experience is knowledge that can let us act in a practiced and automatic manner, or that helps us to assess, select, and apply an appropriate problem solving strategy, method, technique, or tool. It has been gained by acting/doing and may either result from unprocessed and unreflected events in specific situations or from conscious reflection and interpretation about ongoing issues. Several approaches have shown how parts of this knowledge and the related observations could be externalized and hence become easier to share with others. Without the reuse of well-proven knowledge, e.g., in the form of software patterns, software engineers would have to rebuild and relearn the knowledge again and again.

The research fields concerned with the management and maturing of experiences are software reuse and *experience management* (EM). They have increasingly gained importance during the past thirty years. EM is based on concepts from the Experience Factory (EF) [3], Case-Based Reasoning [1], and Knowledge Management [9].

In this article, we propose leveraging common features of Web 2.0 applications (e.g., syndication, recommendation, etc.) within experience management systems by supporting the EF activities (e.g., retrieving, searching, packaging, formalizing, generalizing, etc.). Section 2 describes the EF paradigm, explains what an experience package is and what kind of context characteristics need to be considered, and lists common Web 2.0 application features. Section 3 assigns Web 2.0 features to EF activities and explains how the features support the user or the experience engineer, respectively. Finally, we summarize the paper and give an outlook on further work in Section 4.

2 Background

This section will highlight important aspects of the experience factory and experience packages, in order to make the mapping of Web 2.0 features to the activities of an experience factory easier and more comprehensible.

2.1 Experience Factory

The *Experience Factory* (EF) is an infrastructure designed to support experience management (e.g., the reuse of experiences regarding software products or IT projects) in software organizations. Further, it supports the collection, processing, analysis, and dissemination of experiences and represents the physical or at least logical separation of the project and the experience organization as shown in Figure 1. This separation is meant to relieve the project teams from the burden of finding and preserving valuable new experiences that might be reused in later projects.

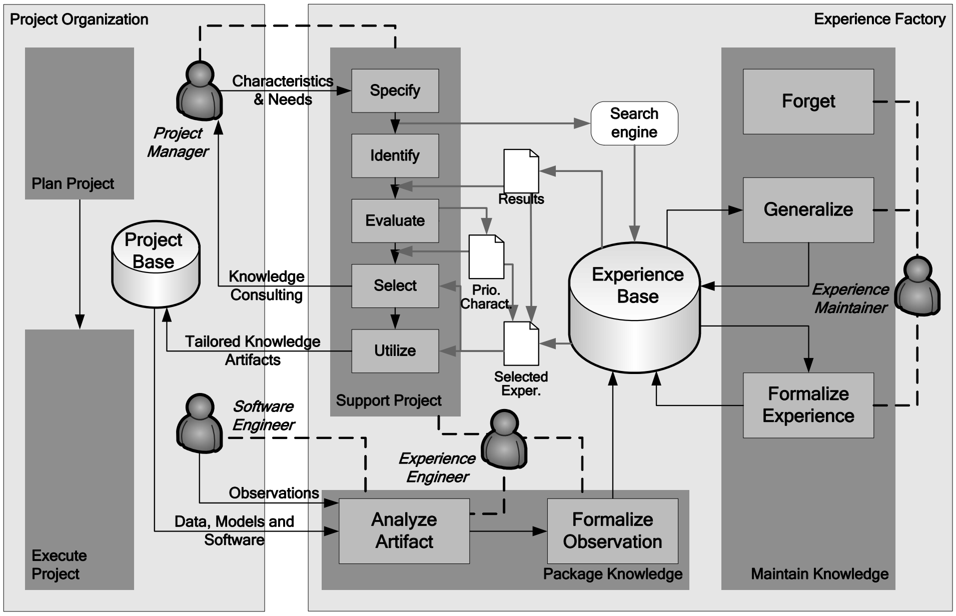

Figure 1: The Experience Factory ([3], [10])

In the “knowledge dust to pearls” approach [4], the observations made in day-to-day work (i.e., the knowledge dust) are analyzed, synthesized, and transformed into knowledge pearls (i.e., patterns or antipatterns).

For example, if we begin a project (Plan Project), the project manager uses the EF to search for reusable knowledge *in prior* to the project. This knowledge comes in the form of effort models, reference architectures, design patterns, or process models based upon our project context (Support Project). As described in [10], this phase is split into the process steps “Specify”, “Identify”, “Evaluate”, “Select”, and “Utilize”. In the execution phase (Execute Project), the EF is used by the project manager and the software engineer to retrieve knowledge *on demand*. During the project and at the project’s end, it is analyzed (e.g., using a post-mortem analysis) in order to extract reusable knowledge (i.e., observations or artifacts) that might be useful in other projects (Package Knowledge). Later during the lifecycle of experiences and observations, this knowledge is then formalized, generalized, or forgotten (i.e., removed) in order to be more usable in future projects (Maintain Knowledge).

2.2 The Experience Package and its Context Characteristics

An experience package is defined as an explicit representation of an experience that can be stored, categorized, and disseminated in an organization. It stems from formalizing and generalizing either experience knowledge or experience gained through systematic measurement and improvement. An experience package description contains a problem statement, optionally a proposed solution including the expected benefit/effect when applying it in a new situation, a context description, and additional administrative information. The so-called *A2E structure* encompasses all information that is required in the experience factory to describe an experience package. The elements used in this structure are described in Table 1.

Table 1: The A2E structure for experiences

Action	Description of an activity that was applied to cause an outstanding effect.
Benefit	Description of the positive or negative effect that was caused by the action.
Context	Characterization of the environment the action was performed in.
Description	Detailed explanation and depiction of the problem, solution, intent, applicability, etc. of the software pattern – based on a pattern template.
Evidence	Report and list of evidence that back up the claim of the software pattern (e.g., used experiences) as well as other relevant references.

The context of a reusable artifact can be seen as all information that does not describe the artifact itself. When the core of the artifact/experience represents the “how”, the context characterizes the “where”, “when”, “why”, “what”, “by whom”, etc. the artifact was created/used in. In general, the artifact represents the data, and all environmental metadata (i.e., data about data) is context. Context explains the environment the knowledge was created in and is a way of giving knowledge meaning and focus, and the more understandable and focused it is, the most effectively it can be captured and reused in a given situation [2].

We derived the following context categories, which are essential for describing the context of experiences:

- individual context (e.g., role, skill and competence profiles, learning preference, activity history)
- group context (e.g., team size, team members, team competencies and experience)
- process context (e.g., activity, lifecycle model)
- product context (e.g., type of product, complexity, quality, application context)
- project context (e.g., size, effort, resources, costs)
- organization context (e.g., competence development strategies, corporate quality strategy, business targets)
- customer context (e.g., business domain, market strategies, etc.)

The refinement of each context category depends on the domain where reuse is applied. In addition to the abovementioned context characterization, each artifact needs a description of the observation itself as well as additional environmental and non-environmental information (i.e., metadata), such as classification, abstract, keywords, definition, explanation, relationship to other artifacts, etc.

Describing the context of experiences has several advantages: Context characteristics help the software engineer to search and find appropriate experiences for reuse. Context categories can help the reusing person in understanding, evaluating, selecting, utilizing, and eventually adapting the experience to a new context.

In addition, context categories support the generalization (i.e., aggregation) of experience. This so-called de-contextualization is necessary to increase the range of applicability of a specific experience. De-contextualization does not mean the removal of the context information, but its stepwise abstraction by combining and generalizing parts of the context description.

2.3 Common Features of Web 2.0 Applications

This section describes widespread features that can be found in many Web 2.0 applications. In Section 3, we assign the described features to concrete EF activities. Weber et al. [11] described common Web 2.0 features based on the survey of Wong and Hong [12]. These features can be part of so-called knowledge services that support the previously mentioned activities in the context of the EF. A knowledge service is provided to the software engineer through applications – in our case Web 2.0 applications.

The following list shows an extract of the relevant features together with examples (the examples constitute mashups [8] as a special type of Web 2.0 applications):

- **Syndication**: The mashup summarizes multiple websites/services or data sets (e.g., Vidmeter.com aggregates videos charts from multiple websites, Yahoo Pipes (pipes.yahoo.com) enables users to create combined feeds).
- **Search**: The mashup provides a search over multiple external data sources (e.g., Kayak.com aggregates multiple search results from different engines).

- **Visualization**: The mashup provides the user with some kind of visualization of external data sets (e.g., liveplasma.com visually suggests to users those artists who might be of interest by leveraging the Amazon API).
- **Personalization**: The mashup makes use of the user's personal information exposed by other websites/services (e.g., leveraging personalized feeds of Del.icio.us as input source) or enables the construction of a personalized data set from the original service (e.g., the mobile shopping service Wishpot.com).
- **Folksonomy**: The mashup leverages a tagging mechanism for organizing its content. Tag clouds help to navigate through the content (e.g., TagBrowsr.com as an alternative way for browsing Flickr content).
- **Alternate UI & In-situ Use**: A mashup provides an alternative UI to external services (e.g., oSkope.com provides a visual interface to Amazon, Ebay, etc.). In-situ use refers to mashups that support specialized usage of a service outside the typical use case (e.g., HeyWhatsThat.com utilizes GoogleMaps to describe mountains and terrain areas to users).
- **Recommendation:** The mashup utilizes recommended content originating from social communities or where users suggest related content with regard to the content of the Web page (e.g., recommended Del.icio.us bookmarks).
- **Rating**: The mashup shows content that was either assessed by users or that has been referred to often (e.g., Amazon's stars, bookmarks in Del.icio.us).
- **Commenting/Annotating**: The mashup integrates all kinds of comments attached to some content (e.g., sticky notes in Diigo.com).

3 Web 2.0 Features for Experience Management Systems to Support Experience Factory Processes

This section deals with assigning common Web 2.0 features to EF activities. We elaborate how these features could support the activities of retrieving, analyzing, formalizing, generalizing, adapting, and discarding experiences. We provide suggestions, for how to leverage these Web 2.0 concepts as features of an experience management system.

The following table shows in the first column the activities performed during a project to reuse experience and within the EF itself. The other columns represent the Web 2.0 features and their support for specific activities (marked with an "x").

Table 2: Activities within the Experience Factory

Experience Factory Activities		Syndication	Search	Visualization	Personalization	Folksonomy	Alternate UI & In-situ Use	Recommendation	Rating	Commenting / Annotating
Support	Specify	x				x	x	x		
	Identify		x	x	x					x
	Evaluate							x	x	
	Select									x
	Utilize								x	x
Package	Analyze Artifact	x								
	Formalize Observation	x				x		x		
Maintain	Formalize	x				x		x		
	Generalize	x						x	x	x
	Forget							x	x	

The first activity *Specify* intends to formulate the query to be processed by the search engine. The query reflects the specification of the required reuse candidate [5]. *Specify* can be supported by recommending search keywords that originate from all specified queries of users, groups, roles, etc. In doing this, the current user context (provided, e.g., through the user profile, user profiling, or a separate form associated with the query) is considered. A syndication feature allows collecting the previously used queries of different persons, groups, roles, etc. in order to process the data (e.g., providing an RSS feed). The folksonomy feature visualizes the relevance of recommended keywords via a tag cloud in order to assist the user in selecting adequate keywords. The alternate UI feature provides a role-specific search view, for example, according to a manager or developer role.

The activity *Identify* uses the formulated query to retrieve experience packages from the database. Finding queried experience packages can be supported by the Web 2.0 feature *Search*. Here, the experience management system searches over diverse data sources, e.g., experience base, corporate blog, enterprise wiki, etc. Recommendation enables ranked results (e.g., based on user ratings (“stars”) or links (“page rank”)). Visualization of clusters within the search results enables fine-grained ranking. Personalization enables the filtering of search results based on a personal profile (i.e., interests, preferences, knowledge, etc.), work context (see Section 2.2), and access rights.

The search results, i.e., the reuse candidates, are now *evaluated*. The evaluation is supported by a list of evaluation criteria defined by the reusing person. Recommendation helps the user to find his own personal and prioritized list of criteria, which enables him to filter relevant artifacts from the search result. Examples of criteria, i.e., experience characteristics, include language, learnability, modification and application costs, comments by other users, etc. A characteristic rating feature is the basis for recommending candidate characteristics.

The activity *Select* is done based on the characteristics of the reuse candidates. After the user has selected the relevant experience, he has to document why he has made this selection. This can be achieved by using a commenting/annotating feature, where the comments are associated with the current context (user context and query).

The application of the selected experience may require a modification. After using the selected experience (activity *Utilize*), the user has to document how the experience (i.e., artifacts) was applied, which can again be supported by a commenting/annotating feature. In addition, he has the possibility to rate how good the application of the experience was.

An experience engineer [7] has to *analyze* project artifacts as a prerequisite for formalizing and storing experience within the experience base (activity *Analyze*). This step implies complex cognitive activities, which are difficult to support by tools. However, the engineer can be assisted by being informed about changes in the project, which may lead to a new experience to be analyzed. As an example, the engineer registers to be notified of activities in particular folders of the enterprise net drive. This information can be provided unobtrusively, e.g., as an RSS feed.

During the activity *Formalize Observation* and *Formalize*, the experience engineer formalizes observations or semi-formal experiences and stores new experiences within the experience base (see Figure 2). Recommendations suggest artifacts similar to a specific topic or task as examples in order to assist the formalization process. Syndication help to collect information related to a topic or task from different sources. Furthermore, the experience engineer registers for particular topics (e.g., by specifying relevant keywords, tags, categories, etc.) in order to be notified about interesting artifacts, e.g., via RSS feeds. A tag cloud supports him in finding adequate keywords for characterizing the artifacts.

The activity *Generalize* is used to summarize multiple project-specific Experiences (E) into a software pattern, for example, a design pattern such as “Abstract Factory”, and finally (together with other patterns) into a Law (L). A law is a generally applicable statement, principle, or heuristic that is valid for all software systems – e.g., Constantine’s Law, “A structure is stable if cohesion is strong and coupling low” [6]. The core goal of this step is the partial de-contextualization (i.e., abstraction) of the experience from its project, domain, language, or technology context. Combining artifacts or existing experience packages into new experiences can be supported by recommending similar or related artifacts or experience packages. These artifacts or experiences can be aggregated via syndication from diverse sources. User ratings help the experience engineer to decide on which artifacts should be integrated. In addition, user comments may influence the decision.

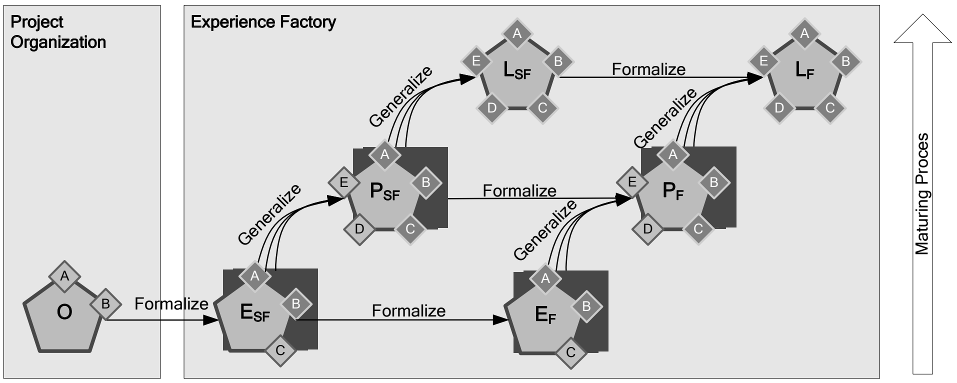

Figure 2: Experience maturity levels

The activity *Forget* helps to remove obsolete artifacts from the experience base. This activity can be supported by recommendations and ratings. Potential artifacts may be suggested based on the number of artifact accesses or ratings, for example. Additionally, manual classification of artifacts, e.g., a user tagging a document as obsolete, can be seen as a selection of an identification of deletion candidates.

Summary and Outlook

The Experience Factory (EF) is an infrastructure intended to support the collection, processing, management, analysis, and dissemination of experiences made on the job. We described how the different activities in an EF can be supported by knowledge services and proposed leveraging Web 2.0 concepts as features of an experience management system in order to implement these knowledge services.

Up to now, only a few experience engineers [7] have analyzed data from projects with the intent of packaging valuable experiences. Nevertheless, the EF will profit from a shift towards collaborative authoring and sharing of experiences. Knowledge services will support not only the experience engineer but also the project staff in conducting EF-related activities. The challenge of an experience engineer is to support several concurrent projects with up-to-date information just in time. Many software projects follow an evolutionary approach, which means that software is developed in teams working under tight deadlines and with changing stakeholder requirements. Hence, the experience engineer is often not able to perform activities such as formalizing, generalizing, or forgetting.

In this paper, we made suggestions on how common Web 2.0 features may support EF activities. It is hardly possible for the knowledge of the experience engineer to cover the knowledge of the collective intelligence in the enterprise all the time. Everybody should have the possibility to contribute to those activities by using Web 2.0 technologies, such as Wikis, blogs, podcasts, or social networking platforms, which are used for knowledge capturing and transfer, collaborative work, and workplace learning.

The idea for the future is to integrate different Web 2.0 technologies in order to build an EF mashup environment. In short, the term mashup refers to an ad-hoc composition of information and services from different sources into new services [8]. In the context of EF, one approach might be to combine the experience base with diverse enterprise data sources, such as content in learning content management systems, corporate globs, project wikis, social networks, discussion forums, and maybe data originating from sources outside the enterprise firewall. Web 2.0 concepts, such as user-generated content or recommendations, as shown in this paper, would help to increase the quality and up-to-dateness of the experiences.

References

1. Althoff, K.-D.: Case-based reasoning, In (S.-K. Chang (eds.)): Case-based reasoningvol. 1, pp. 549-587. World Scientific, Singapore, 2001

2. Araujo, R.; Santoro, F. M.; Brézillon, P.; Borges, M. R.; Rosa, M. G. P.: Context Models for Managing Collaborative Software Development Knowledge, In First International Workshop on Modeling and Retrieval of Context (KI 2004), MRC2004, Ulm, Germany, 2004.

3. Basili, V. R.; Caldiera, G.; Rombach, H. D.: Experience Factory, In (J. J. Marciniak (eds.)): Experience Factoryvol. 1, pp. 469-476. John Wiley & Sons, New York, 1994

4. Basili, V. R.; Costa, P.; Lindvall, M.; Mendonca, M.; Seaman, C.; Tesoriero, R.; Zelkowitz, M.: An experience management system for a software engineering research organization, In 26th Annual NASA Goddard Software Engineering Workshop, 2001. IEEE Computer; pp. 29-35

5. Basili, V. R.; Rombach, H. D.: Support for Comprehensive Reuse, Journal of Software Engineering, vol. 6, no. 5, pp. 303 - 316, 1991.

6. Endres, A.; Rombach, H. D.: A handbook of software and systems engineering: empirical observations, laws, and theories (1st Edition), Pearson Education Limited, Harlow, England, 2003.

7. Feldmann, R. L.; Frey, M.; Mendonca, M.: Applying roles in reuse repositories, In International Conference on Software Engineering and Knowledge Engineering (SEKE'2000), Chicago, USA, 2000. Knowledge Systems Institute; pp. 1-15

8. Merrill, D.: "Mashups: The New Breed of Web App," http://www.ibm.com/developerworks/library/x-mashups.html, last accessed on 24 November 2008.

9. Nonaka, I.; Takeuchi, H.: The Knowledge-Creating Company Oxford University Press, New York, 1995.

10. Tautz, C.: Customizing Software Engineering Experience Management Systems and Related Processes for Sharing Software Engineering Experience, Ph.D Thesis. Kaiserslautern: University of Kaiserslautern, Germany, Department of Computer Science, 2000.

11. Weber, S.; Thomas, L.; Ras, E.: "Investigating the Suitability of Mashups for Informal Learning and Personal Knowledge Management," in Workshop on Mash-Up Personal Learning Environments (MUPPLE08). Maastricht, The Netherlands (2008)

12. Wong, J.; Hong, J.: "What Do We "Mashup" When We Make Mashups?," in 4th international Workshop on End-User Software Engineering (WEUSE 2008). Leipzig, Germany: ACM, New York (2008)