Chapter 5
Experiences: Beyond Knowledge

5.1 Objectives of this Chapter

After reading this chapter, you should be able to

- Define experience as opposed to knowledge.
- Draw and explain the experience life-cycle.
- Give examples of activities carried out in each part of the life cycle.
- Analyze your own experience with respect to the concepts presented.
- Decide what needs to be done next when a fellow software engineer turns in an experience report.
- Name several techniques and tools that support activities in the life cycle.
- Map experience to a simple ontology or a case-based reasoning system.
- Discuss advantages and loopholes of explicit experience exploitation.

This chapter builds on the knowledge management issues presented thus far. It goes into more depth about mechanisms for acquiring knowledge and experience and describes how to process them during their life cycles. It might come as a surprise how much effort and sophistication are required to extend knowledge management toward systematic exploitation of experiences.

Recommended Reading for Chap. 5

- Davenport, T.G.P., *Knowledge Management Case Book – Best Practises*. 2000, München, Germany: Publicis MCD, John Wiley & Sons
- Schneider, K. and T. Schwinn, *Maturing experience base concepts at DaimlerChrysler.* Software Process Improvement and Practice, 2001. 6: pp. 85–96

5.2 What is Experience, and Why is it Different?

The previous chapter explained how knowledge can be encoded in different representations and stored in ontologies. The instances of an ontology make up a knowledge base, which can be searched and used for reference. Advanced mechanisms for information retrieval and presentation can make use of the semantic annotations that come with an ontology. Semantic searches on the World Wide Web are at the

K. Schneider, *Experience and Knowledge Management in Software Engineering*,
DOI 10.1007/978-3-540-95880-2_5, © Springer-Verlag Berlin Heidelberg 2009

core of the Semantic Web Initiative [17]. Those mechanisms can also be used for building a new ontology for your company.

According to the introductory chapters of this book, information turns into knowledge when it is related to a person's preexisting knowledge. From that perspective, knowledge is always bound to brains and people. We have suggested interpreting the term *knowledge transfer* as a layered protocol of data, information, and (virtual) knowledge flows. Knowledge management alludes to a similar mechanism: By collecting, refining, and distributing information to those people who can extend their preexisting knowledge, it is fair to call this process (indirect) management of knowledge. Ontologies are among the most sophisticated mechanisms for supporting formal encoding, exploitation, and use of knowledge. There are exchange formats like OWL and tools like Protégé to implement ontologies in practice.

In Chap. 1, essential concepts were defined. Core aspects include:

- Knowledge resides in people only.
- Experience is a subset of knowledge.
- Each experience can be viewed as a three-tuple of observation, sensation, hypothesis, or conclusion. All three components are necessary to gain authentic, impressive, and useful experience.
- Knowledge can be built by learning from experience. Some authors even regard this as the only possible way to acquire knowledge; others see reading and listening as other options.

The informal and semiformal techniques in Chap. 3 were introduced to elicit ideas and experiences. They also show how to formalize them. Ontologies are among the most formal representations of knowledge in software engineering – many companies have not yet reached that level in their knowledge and experience management endeavors. Nevertheless, all projects can build on their own experiences.

Experiences are a particular subset of knowledge in software engineering. Experiences have a number of properties that make them special and that require special treatment. Experience management has never reached the same level of public attention as that of knowledge management. At the same time, there has always been high appreciation for "experienced" software engineers. Experience is held in high esteem in most environments; but the associations related to experience diverge drastically. In this chapter, we will need to narrow the definition of a useful experience in order to handle it more effectively.

Let us look into a few examples of what might be considered "experiences" in a software engineering environment. All examples are taken from real projects. Each of them is briefly discussed to gain a better understanding of relevant properties.

Example 5.1 (Experienced architect)

Alex is an experienced software architect. She has participated in four large projects and a few smaller ones in the role of an architect. When she is confronted with a requirements specification, she always looks at quality requirements first. Those requirements have always had the biggest impact on architecture and design. When Web systems are to be developed, Alex asks for frequency of use and reliability

requirements, whereas many of her junior colleagues never think about those issues until the very end of the project – when it is almost too late.

When people talk about experience, a story like this could be told. In the perception of colleagues and co-workers, experiences are not distinct items of knowledge but an attribute of a person. As such, experience does not describe a very specific ability but refers to a wide and vague spectrum of tasks someone performs very well. In experience management, however, it is often advantageous to look at distinct pieces of experience rather than a vague fluidum of a person. This perspective is dictated by our professional desire to manage, spread, and exploit experience systematically. In private life, a different perspective may be more adequate.

Describing experiences as three-tuples may help when we analyze a situation. In the above-mentioned scenario, one might note one of Alex's *observations*: "Junior colleagues asked for user volume late in projects X, Y, and Z." She obviously remembers that this was "almost too late." In more concrete terms, Alex might have *observed* last-minute changes in architecture, causing errors and long working hours. Alex did not like that (*emotion*), as she prefers a regular family life. Therefore, she has decided for herself to ask for frequency of use right away – hoping to consider those quality requirements in the first place and to avoid overtime work (*hypothesis and conclusions*). Note how some of the aspects in the three-tuple were not explicitly presented in the original story. How do we know Alex observed overtime work and did not like it? In most reports on experiences, the observations are cut short, and emotional aspects are omitted altogether. Unfortunately, only a blend of selected observations, conclusions, and biases is passed on to others. We consider them incomplete experiences.

Alex can act according to her experiences without ever making them explicit. She just does what she has concluded to do. In fact, experiences tend to be tacit. Making them explicit requires a trigger and a motivation. Why should Alex talk about her feelings? And who would care to listen in a professional environment? Transferring experiences effectively is rather difficult. This is the main reason why experience management calls for techniques that go beyond knowledge management.

Example 5.2 (Terrible experience)

Kevin had a terrible experience in his last project: Mary and Dieter may be great software engineers, but they are not team players. Whenever Kevin asked one of them to explain a design concept, they would look at each other and start to giggle. Kevin just joined the company half a year ago, and their reaction drove him crazy. What kind of knowledge management could force them to help Kevin when he needs it?

From our experience management point of view, the experience reported above is neither complete nor useful. There is a full observation and a definite emotional response. However, a conclusion is missing. Should Kevin conclude not to ask co-workers for help? Is it an adequate conclusion to blame the problems on poor knowledge management? None of those are conclusions worth managing. Although a "bad experience" can be a source of rewarding insights, it may also be distorted by anger and overreaction. Only when Kevin's boss learns about the situation may there

be a well-tempered conclusion. This example shows how formally well-structured experience can either be inadequate or only appropriate at a different level.

Example 5.3 (Testing experience)

In the next story, Tom is testing code. He had been a developer for many years and only recently started to focus on testing. During a 1-week seminar, Tom has refreshed his knowledge on testing techniques. That helped him to create test cases in a more systematic way. In particular, he has learned to distinguish black-box testing that is driven by the specification and white-box testing that is driven by code structures and coverage. After the seminar, he is highly motivated to apply those concepts. When he finds a bug in a module ABC, he remembers a similar bug he had to fix as a developer: He had forgotten to initialize a variable. Out of curiosity, Tom checks the initialization in ABC – and there is the error! Tom is thrilled about his finding and eager to tell others about it. He is full of excitement when he talks to four of his co-workers over lunch. He emphasizes that he has invented a new testing strategy: checking for missing initializations. One of the colleagues is responsible for experience and knowledge management. He interviews Tom and captures the experience as long as it is fresh.

In this scenario, fine-grained experience is transferred orally when it is fresh and full of emotion. Tom does not fill in a boring experience report form half a year later; he passes on orally and informally what he has just experienced. Chances are good for his co-workers to pick up Tom's excitement together with his advice. They know the kind of module Tom was working on, and they trust him as a modest and responsible person. In this scenario, all three aspects of an experience are present, and they are transferred over lunch. When Tom is later interviewed by the experience and knowledge management (EKM) personnel, more details come up. However, in an official setting, Tom might be reluctant to show his emotions – too bad for the written account of his experience. The transition from implicit to explicit representation of experience is even more delicate than with knowledge.

Conclusions: The three stories illustrate three different kinds of "experience" that we know from daily life. The term *experience* is used intuitively with very different connotations and different associations. By analyzing all three stories with respect to reusable experiences, we try to identify the three-tuple of experience aspects. Although all stories talk about "experience," some aspects were missing in the first story; overwhelming emotions distorted the experience in the second story; and the status and freshness of experience presentation had a major impact on its usefulness in the third story. In the following sections, we will address several activities in the experience life-cycle. Each of them needs to consider the delicate nature of experiences.

Experience versus knowledge: With knowledge, neither observations nor emotions are required. According to Definition 1.7, knowledge is related to other pieces of knowledge in the brain. In theory, it does not matter where knowledge came from or how people liked it. Everyone can learn facts and even relate them to other facts he or she knew before. However, knowledge is often less authentic, convincing, or

impressive than experience. Smart experience and knowledge engineers will target a balance of simple, factual knowledge with a more powerful and more expensive complement of experiences.

5.3 Valuable Experience in Software Engineering

The stories in the previous section showed different interpretations of experience in a software engineering environment. Obviously, both content and presentation of an "experience" are important for its reusability and merit. The three-tuple of observation, emotion, and conclusion provides a first template for experiences. If any of these content aspects is missing, the experience is not complete. In many cases, knowledge is the conclusion aspect of experiences.

What is good and valuable experience in the realm of software engineering? It is impossible to list all content areas; the SWEBOK [56] mentioned in Chap. 2 is a good reference for an overview. However, we might be able to identify certain criteria that specify intentionally what makes a good experience. This characterization will be used as a reference for all activities in experience management: How do they attract and preserve the right experiences?

Criteria for valuable experiences in software engineering from an experience management perspective follow:

Related to a task at hand. Experiences that are not related to the job or the activities carried out in a software project may be emotionally loaded and very authentic. However, they do not contribute much to the success of the company or the business unit. This criterion may sound trivial, but when experiences are collected, it is often violated. People forget to focus.

Related to a frequently recurring task or situation. Experiences related to a unique event are less valuable than those related to a daily situation. The potential reuse frequency of an experience depends mainly on the frequency of the respective situation. Supporting daily tasks just a little may pay off more than a significant improvement or saving in a rare activity.

Containing all three aspects of an experience: observation, emotion, and conclusion. The above-mentioned stories illustrate this point: It helps to understand better what happened, and it helps in memorizing and applying the experience. The three aspects refer to contents, not to format. It does not matter how an experience report organizes the aspects, nor whether they are neatly separated. The challenge related to this criterion is rather to encourage experience holders to make all aspects explicit – in a professional environment. For the experience manager, the main challenge is to keep experiences "fresh": analysis, storage, recombination, and delivery must not lose any of the three aspects – or a part of the experience is lost.

Experience made in a similar context. Experiences may be highly context-dependent. What might be working in a strictly hierarchical company could be counterproductive in a small team (e.g., quality assurance procedures). The issue of identifying and matching contexts is a major challenge for experience exploitation. There are basically two alternatives for picking only appropriate experiences:

(1) Specifying context attributes and characterizing new situations by those same attributes. Similar attribute values indicate highly related experiences. (2) By limiting the scope of many experiences to a common context, all experiences are more or less related to that *same* context and can be reused. The second approach is easier to implement but has fuzzier borderlines: Very specific experiences might not be transferable based on the common context. It might be best to adopt approach (2), but enrich it with optional attributes to describe specific context constraints. This minimizes effort without limiting the ability to be more specific when needed.

Fresh and authentic. Originally, experiences were exchanged orally. Stories were told over a cup of coffee. When they became boring, the experience would no longer be mentioned. In experience management, the quality criterion of "freshness" is still important. An experience conclusion is more likely to be reused when the audience can relate to the observation. In that case, they will "feel the emotions" and feel as if they were in the shoes of the original experience holder. This emotional link provides authenticity and a better chance for reuse. On the other side, experiences that are outdated, incredible, or from a totally different background are less valuable.

Suggest actions for reusers. The conclusion or hypothesis of an experience may be abstract or concrete. It can explain the reasons for a situation to occur, or it can provide advice on how something can be achieved or avoided. Although all variants of conclusions and hypotheses can make valuable experiences, those that provide concrete advice for action are often more valuable. However, experience engineering is an activity that tries to rephrase and rearrange experiences. Therefore, experiences can be improved later on.

Short and easily comprehensible. Like the previous criterion, size and style can be reworked afterward. In the end, however, a short experience report or presentation is more valuable. When someone needs to find experiences related to his or her own situation, the experience report should allow finding out quickly. It is not only the length of the document that counts: Clear and concise writing style, structure, and overview elements (figures, tables) add to this aspect.

Appropriate granularity; smaller, when in doubt. Some experiences do not even need a full experience report. They can be stated in just a few sentences. This is even more true when unrelated experiences are reported separately. Smaller chunks of experience are easier to read and to understand. However, there are good reasons for not dividing each experience into very small experiences. The lower limit of valuable experiences is defined by their context dependencies: If a small-grained experience is reusable only in a very limited context, it may be more laborsome and less reusable to describe in that context – better leave the experience within its natural context of the longer story. This criterion is more difficult to apply: As small as possible, but as large-grained as dictated by the original experience context.

Reliable. Different people can interpret the same observation very differently. Depending on their own private prior experience and prior knowledge, they may reach different conclusions. Therefore, experience three-tuples may be contradictory or outright wrong. Everyone managing and reusing experiences should be aware of that risk. It comes from relying on all the tacit background of those making the experience. Nevertheless, experience holders should not be encouraged to make up highly speculative hypotheses or draw unlikely conclusions.

Levels of Exchange

Experience exchange can occur on different levels of an organization. Each level has its advantages for certain kinds of experiences.

Personal communication. This can hardly be called experience management, but it is the most common and most effective way of reusing experiences. Experience management should not object to it but add other opportunities that are less confined to a personal relationship.

Exchanging experiences within a team, group, or department. A small environment offers the advantage of familiarity. It is easier to foresee who might eventually receive an experience report and what may be the reactions. As opposed to a very large, anonymous community, a small group can build on trust and exchange more delicate experiences. Critical conclusions and errors are delicate.

Large collections of experiences and knowledge (e.g., in an ontology) are usually established by central units in a company. Factual and procedural knowledge that is independent of context details can easily be shared within a larger group of knowledge workers. This is the level of company knowledge management initiatives. Merging knowledge and experiences from more people has a better chance of exceeding the critical mass. The more a repository can offer, the more likely it will have something to offer for any given requestor. Return on investment for experience and knowledge management will grow when reuse is facilitated within a larger community. However, context-sensitive issues, unreliable or contradictory experiences, or delicate details about individuals must not be spread beyond a small, trusting circle.

Some initiatives have been established among a group of companies (e.g., DaimlerChrysler in the Software Experience Center [95]). They avoid competitive issues but exchange information and highly aggregated experience on software engineering issues, such as the adoption of the Capability Maturity Model CMM [83], the European initiative for software maturity, SPICE (ISO 15 504), or the Capability Maturity Model Integrated CMMI [1]. The CMMI contains concepts inspired by SPICE and CMM. At this level of abstraction, strategic decisions and experiences can be exchanged.

At the highest level of congregation, conferences, user groups, and professional community meetings (e.g., organized by the Association for Computing Machinery, ACM [110], or the German Society for Computing, "Gesellschaft für Informatik" GI [47]) serve a similar purpose: exchanging experiences and insights. On this level, some experiences are conveyed in experience reports. Others are transformed into conference papers and contributions that emphasize an insight. In our terminology, this may be either a conclusion from an experience or a deeper insight gained by relating experiences with previous knowledge. On this level, experiences have mostly lost their authentic quality. When sufficient experience engineering and peer reviewing has been done, some new "knowledge" may emerge from the experiences – and knowledge is treated differently.

This section and the previous sections have set the stage for the experience management activities presented below. You should keep example stories and criteria

in mind when we now walk through the different steps of experience engineering. Knowledge management follows a similar life-cycle, as discussed in Chap. 2. We will mainly look at experience engineering but also point to some differences in the respective activities of knowledge management.

5.4 Experience Life-Cycle

In Chap. 1, an overview of knowledge management was given. Fig. 1.8 summarized the activities in knowledge management. Managing experience is a special case of managing knowledge. The activities of Fig. 1.8 are, therefore, mapped to the activities of the so-called experience life-cycle. The activities of experience and knowledge management are not a one-shot task. Instead, they need to be repeated over and over again. The end of a cycle prepares the organization for the next cycle. The iterative nature is just alluded to by the cycle on top of Fig. 5.1. At the bottom,

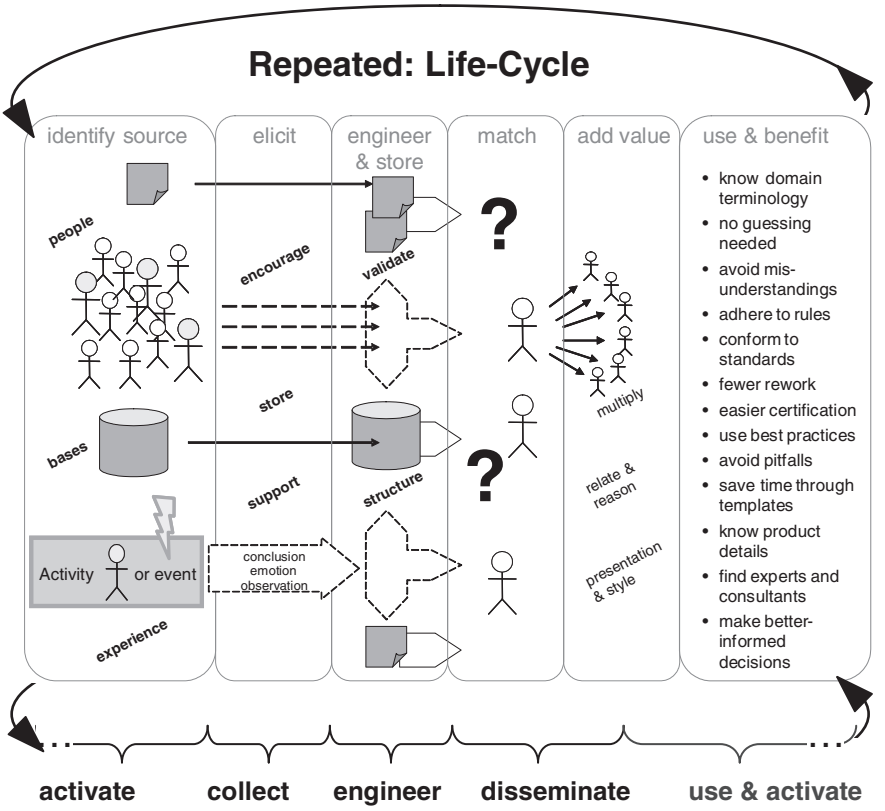

Fig. 5.1 Experience life-cycle as iterative knowledge management tasks

knowledge management steps are mapped to slightly different terms and activities. The latter will be used below to describe the experience life-cycle.

The different wording indicates different challenges. In experience management, a source of experience needs to be identified and activated: We need to take action in order to make experience holders aware of the asset they have. Often, experience is tacit and must be made explicit. This happens during the collection or elicitation activity. Very important is the engineering activity that compares and validates, refines, and turns experiences into best practices. The aspect of storing experiences and intermediate products of experience engineering is a more technical issue that is often overrated. The importance and difficulty of matching existing experience to current needs is often underestimated. Experience can be reused only when the last step of experience management (dissemination) is also carried out effectively.

Actual reuse is not considered part of the experience management life-cycle. In fact, new insights are gained and new experiences are made during the usage of previous experience. Experience management needs to identify new experiences, thus closing the cycle. Actual usage is considered the glue between delivering stored knowledge and experience – and reacting to it, based on new experiences. A slightly more abstract view of the experience life-cycle is presented in Fig. 5.2. As opposed to Fig. 5.1, inner details of the activities have been omitted. In turn, the iterative nature was emphasized, and the experience management terminology was used.

The cycle shows experience management tasks. According to the above argument, it is a special case of the knowledge management life-cycle, as depicted in Fig. 2.5. Application and use of experiences during software projects need to be synchronized with the experience life-cycle. From our perspective of experience and knowledge management, experience-related tasks appear in the foreground. The entire life-cycle is needed to provide software engineering. Application and use are not considered experience management tasks themselves.

Experience Management Tasks and Activities

The following sections describe activities and approaches in experience management that help to instantiate the tasks in Fig. 5.2. The task bubbles are clus-

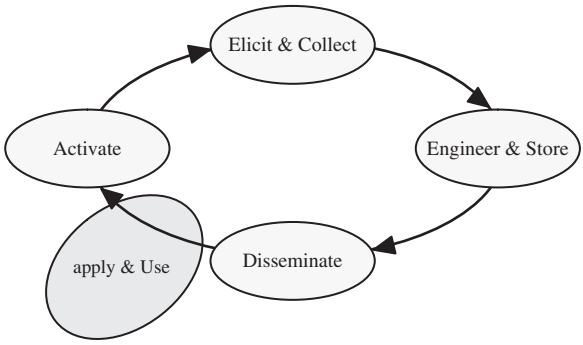

Fig. 5.2 Tasks in the experience life-cycle

tered by logical coherence: Activating experiences is a logical prerequisite for collecting them. Elicitation is a specific subtask of collecting experiences. Instead of simply storing data, experiences must be compared, validated, and reformatted before they are stored. Again, storing is a rather simple subtask of experience engineering. Subsequent sections are grouped by activities: For example, an interview is an experience activity. It performs two life-cycle tasks: activation and elicitation of (raw) experience. Both tasks are iterated within seconds. An interviewer should be aware of those two tasks and know that none of them must be neglected.

5.4.1 Activating and Eliciting Raw Experience

If we start walking through the experience life-cycle at the Activate step, we are confronted with raw experience. Identifying sources of experiences and trying to elicit valuable experiences is mostly carried out by an activity like:

- *Interviewing people*. A member of experience management conducts an interview with one or two people who were identified as having valuable experience. Many people seem to be experienced, but the challenge is to identify those with the right kind of experience. Another challenge is to help them make it explicit. As we have seen in Chaps. 1 and 2, tacit knowledge and experience are hard to capture: Because the interviewer does not know exactly what to ask for, an interview may miss the point.
- There are *structured and open interviews*. Structured interviews are guided by a set of questions to answer. Strictly structured interviews resemble answering a questionnaire. In a semistructured interview, a similar list of questions is prepared, but the interviewer does not strictly follow the list. Questions are rather seen as a checklist that can be addressed in any order and even without asking explicitly. Open interviews leave the initiative completely to interviewer and interviewee.
- *Asking experience holders to write down their experience*. This is a popular approach but a problematic one. Experience holders have little reason to do their best in writing down experiences. What is more, they can often not imagine what needs to be documented.
- *Conduct a workshop with several experience holders*. This is a variant of an interview. The moderator needs to facilitate cross-communication. Someone will take notes and document raised arguments and experiences. In a multiperson situation, the moderator has less control, and notes will need substantial engineering and rework to turn them into useful experience reports.
- *Try an advanced technique for eliciting experience*. There are a number of techniques that are tuned to make experience holders aware of what they know and help them to document it with the lowest possible effort. For example, the lightweight identification and documentation of experiences (LIDs) technique [90], a variant of post-mortem analysis can be used. Others use tools and technology to

facilitate note-taking or recording. A small number of sophisticated techniques apply dedicated software tools in specific situations to capture experience in a light-weight way [112, 113].
- *Studying existing documents*. Initial elicitation does not always imply talking to people. Sometimes, there are traces in documents pointing to an interesting event. For example, two versions of a specification that differ completely might indicate a problem in between. When one document (like a manual) is supposed to build on another one (a design or specification), but there are obvious differences, this can be a good trigger to ask very specific questions.

Mnemonic 5.1 (Raw experience)

No matter what technique is applied during the identification and activation step, the result will be raw experience. Notes, records, and initial experience write-ups will need cleaning and engineering. Raw experiences are rarely fit for reuse. Many raw "experiences" do not even qualify as experiences. According to our definition, they are missing an aspect, or they do not fall into the category of valuable experiences.

Raw experiences usually have deficits:

- Observations and conclusions are too specific for reuse by someone else.
- Conclusions are unclear and either too abstract (what needs to be done?) or too specific (how would that translate to a slightly different situation)?
- Conclusions may be unreliable. A single observation or conclusion will hardly make someone make a drastic change. Confirmation by others is lacking.
- Stories, statements, or experiences are confidential, contain names and secrets, or could be seen as personal attacks. All these cases require careful cleanup and checking. Often, company names, project titles, or individual names should not be disseminated beyond a very restricted circle of team members.

During elicitation, an interviewer or moderator cannot check the quality of the experiences that might or might not be hidden in the raw material collected. Sorting out raw experiences is inevitable. Disseminating unchecked raw material can do severe damage to an experience management initiative.

Many of those phenomena are typical for experience; they would not occur in knowledge acquisition.

5.4.2 Experience Engineering and Storing

Definition 5.1 (Experience engineering)

Experience engineering is an activity that improves experiences or related material by applying systematic procedures, similar to engineering.

Tasks Within Experience Engineering

In particular, experience engineering includes:

- Cleaning, sorting out, and rephrasing raw experiences.
- Technical reformatting and preparing for storage and retrieval (indexing, short description, evaluation, transfer to an exchange format like XML).
- Constructing more valuable experiences by comparing and aggregating several experiences and deriving a common overview or conclusion: Contradictions, inconsistencies, and confirmations are uncovered. There may be problem-specific relationships between experiences, too.
- Reformatting or rephrasing experiences or derived material. Not all experiences are documented in a highly readable and reusable style. Experience engineering should remove typos and mistakes, rephrase overly complex sentences, and facilitate understanding by deriving figures, tables, or visualizations from given material. Material may include other experiences or knowledge.
- Harmonizing experience and derived material in style and granularity.
- Turning textual experience reports into presentation material like slides, diagrams, or short reading material.
- Turning experiences into procedures to follow. They are often called "best practices." In a project situation, software engineers usually do not care to learn about experiences made by others. They need to know what to do in their own project. Therefore, experience engineering should turn one or more experiences and related material into hints for action. In the ideal case, a complete process or procedure will be described that reflects the experiences made.

Storing the results of experience engineering is a logically separate task. Of course, it will be carried out every time a result is created. At this point, storing can be considered a trivial "save" operation that does not require any in-depth discussion. Preparing for appropriate storage, however, is less trivial: Examples in Chap. 6 show how storage structures (databases, Web sites, files) need to follow intended dissemination strategies. Once they are in place and the experience cycle is turning, storing has become an easy operation (Fig. 5.3).

The term "**best practice**" is often used in a euphemistic way. Strictly speaking, a best practice would need to be derived from practice or practical experience – with no speculation or theory involved. This degree of experience coverage is rarely achieved. Mostly, conclusions from experiences will be bundled with common knowledge or some of the experience engineers' convictions. Calling a documented procedure a "best" practice logically seems to imply it has been compared with some other, less suitable practices. Again, there will rarely be a set of competing practices that failed to win the title of the "best of all practices".

In most environments, best practice indicates a procedure that is based on experiences from practice. Some mistakes were identified through experiences and are avoided by that procedure. Further conclusions from experiences have been taken

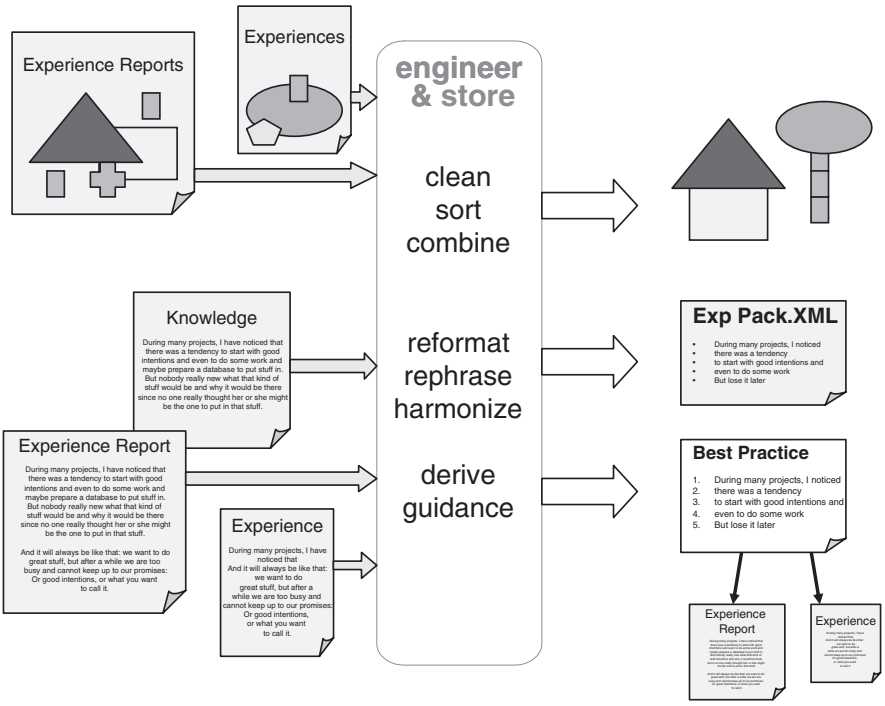

Fig. 5.3 Overview of experience engineering tasks

into account, and gaps have been filled with existing prior knowledge or other experiences.

Although it is not customary to apply very strict criteria to what is called "best practice," this term should be more than a marketing label. We should not accept calling a prescribed procedure a "best practice" that has never been tried and proved its value in practice. This is a reasonable criterion for using the term "best practice" for practical purposes.

Experience engineering can be carried out by dedicated experts in the field of experience and knowledge engineering. However, even lay persons will add value to their own raw experiences when they review them in a separate cycle. Peer reviews of experiences will produce even better results. An essential point is: No matter how the raw experience was collected and no matter who does the job – **experience engineering is crucial** for the success of an experience management initiative! If you doubt the necessity of this step, think about the above-mentioned list of potential deficits in raw experiences. Collecting and cleaning are two contrary modes of operation. From a cognitive point of view, it is impossible to carry out both in parallel. Therefore, they cannot be carried out on the side while raw experiences are reported. A separate step is required

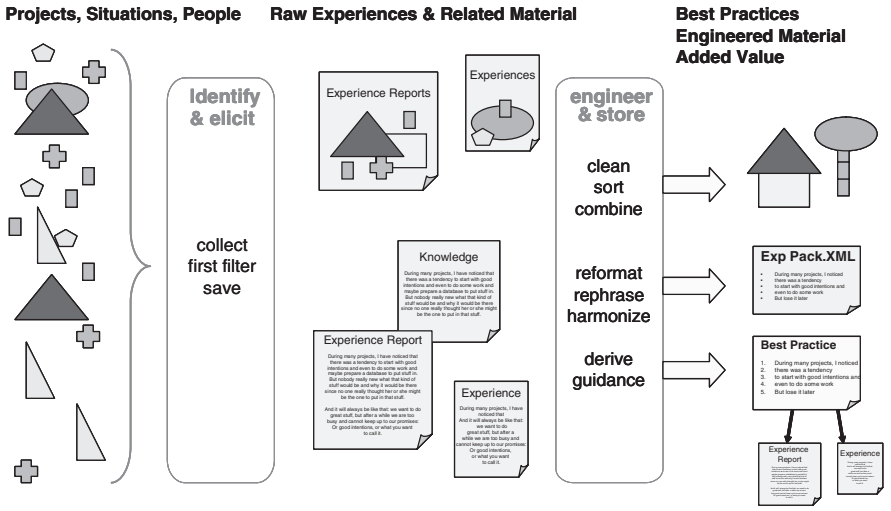

Fig. 5.4 Eliciting raw experience; engineering to add value

Best practices and experiences: Note how experiences are linked to best practices in the lower left corner of Fig. 5.4. Although it is important to present the combined conclusions in the form of advice for action, the authenticity and power of experiences should be preserved. For that reason, a best practice should cite the experiences it was derived from. This allows readers to (1) quickly find out what to do and to (2) learn more about backgrounds and related experiences. Even better: it should provide hyperlinks to allow direct access from the best practice to related experiences. The second case will be especially important if a best practice does not seem to fit their current situation. A project manager or software engineer needs to find out the rationale behind the best practice advice and derive his or her own conclusions.

Experience engineering itself is a highly knowledge-intensive task that requires expertise and experience in the field. Because many of the experience engineering operations listed above require filling gaps or interpreting material, experience engineers cannot be replaced by mechanisms or administrative clerks who are foreign to the software discipline. Therefore, an appropriate representation of experience, knowledge, and related or derived material is crucial for experience engineers to elicit ("activate") ongoing feedback from the projects. It is in experience management's best interest to encourage critical reflection and feedback (see Sect. 5.4.4). A format like Experience Reports invites detailed comments on rationale. This is a good opportunity for the refinement of experience and best practices. As we will see, it may be a necessary but not sufficient precondition for closing the experience learning cycle.

5.4.3 Dissemination

Novices in the field of systematic learning from experience often forget or underestimate this task. They will often neglect the matchmaking task and consider their

job completed when experiences or best practices have been stored and "made available" at some Web site. This is a fundamental mistake.

It has been shown in many practical knowledge management initiatives [9, 12, 39, 92] that an active and explicit effort is needed for matching the derived material to those who might need it and to help them use it. We call this step *dissemination*.

Distribution is a related term, but dissemination emphasizes the challenge of actively distributing material *to the right person at the right time in the right format*. This task is far from trivial. Without effective dissemination, all previous activities for identifying sources, eliciting and activating experiences or knowledge, and all effort spent on knowledge and experience engineering and storing will be wasted. This claim holds for the entire field of knowledge and experience management: Failing to deliver results to the software engineering practitioners in an appropriate way turns the entire organizational learning initiative into a failure.

Acknowledging this challenge is probably the most important step. Experience management needs to carefully consider how to address the following issues with each given piece of experience or related material:

1. Who is the potential addressee of an experience or a derived best practice?
2. What is the situation in which this kind of material would be most helpful, and are any other reuse scenarios imaginable?
3. What material and advice would a software engineer or addressee need most in that situation? Are there any documents, templates, data, or experiences that would assist them efficiently? Can they be derived or engineered from the material?
4. How do we know who is doing what in a current project situation, and how do we know this across multiple projects that may even be distributed in a large organization?
5. How do we know what situation people are in and when they will reach a status that matches the reuse situation anticipated above?
6. How can they articulate what they need – which might or might not differ from the anticipated needs?
7. How can someone find and reuse material such as experiences, best practices, or parts thereof. In most cases, neither the person nor the situation will match the exact profile of a given experience. The only communality may be a similar challenge in a different context. How can different contexts be considered?

Dissemination must take into account the individual situation and needs of those software engineers who are supposed to benefit. Chapter 6 discusses several examples and options for concrete implementation. Some of them can be reused. However, each new situation calls for careful consideration of the above-mentioned questions.

Example 5.4 (Order is important)

The above list is numbered because we need to refer to some of the above-mentioned issues. The numbers do not necessarily imply a *strict* order, but it may be useful to use it as a checklist for each experience or derived result. Moreover, the order

of issues can guide dissemination. For example, there is no use in contemplating exceptional reuse (7) as long as regular users (1) and scenarios (2) have not been identified. Each issue will have consequences for dissemination: Where, how, and when should what material be presented or even sent directly to someone?

As software engineers, you are used to planning. Planning is based on information and estimations. Issues 1, 2, and 3 have to do with anticipating users and situations where that reuse may occur. We need to guess what the reuse situation might look like and optimize the engineering and dissemination efforts for that purpose. If we have good and valuable information on reuse situations, chances are better to meet the needs. Therefore, experience management needs information on how the material might be reused. A good approximation is to look at how it has been reused in previous cases. For that reason, dissemination needs to be a two-way road: While reusers receive material, experience management needs to get back information on usage. As we will see, both directions need to be made as easy and painless as possible.

Issues 4, 5, and 6 address a different problem: In order to match profiles, how do we know about all current activities? If experience management takes these issues seriously, keeping track of projects and status information is obviously important. As a direct consequence, experience and knowledge management should be linked to planning and tracking information held in software management tools. If automated matchmaking is desired, profile information needs to be formal enough for automated comparison and evaluation.

Issue 7 goes beyond making anticipated matches. Many artificial intelligence researchers and even some knowledge engineers dream of the "magic match" when a piece of previously engineered experience or knowledge is combined and delivered in a nonanticipated way, making a big improvement. Software engineers will be delighted if they encounter this rare phenomenon – but they should not rely on "surprise matches" (Fig. 5.5). We should plan for planned matches and welcome all others as an extra bonus.

Some knowledge engineers argue: "We can never know all the situations in which a piece of knowledge can be reused – so it is counterproductive to plan and think about it for long." Although they may be right in some situations, software engineering is a domain too sophisticated to expect emerging matches. Obviously, there are different opinions on this point, and you should make up your own mind when you implement experience and knowledge management in software engineering practice.

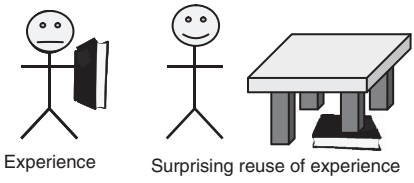

Fig. 5.5 Magic match: unanticipated use of an experience report

Example 5.5 (Expecting an emerging match)

A software company sells computer games. Sometimes customers called in and wanted to buy gaming hardware with the games. A knowledge engineer was just about finishing an ontology on the company games. Instead of providing a simple Web interface with a list of links to the games, the engineer provided a natural language interface for customer requests that could search for all potential future devices and services and features imaginable.

There is usually a huge gap between the potential power and the actual usefulness of a system that waits for emerging matches. It is often better to optimize for a set of matches that are required.

5.4.4 From Dissemination to Activation (Next Cycle)

Dissemination is closely tied to experience engineering. The goal of engineering a certain type of experience should be specified by looking at the material needed for dissemination. As a rule of thumb, engineering refers to adding value by working on contents, whereas dissemination refers to presentation, advertising, and distribution.

As issues 4, 5, and 6 have indicated, dissemination also extends into the use of experience-based material: The more experience management knows about the actual use of the delivered material, the better it can focus future engineering and dissemination. The experience life-cycle is a two-layered learning cycle: Software projects learn by reusing experience-based material, and experience management learns to handle experience-based material better. In the terminology of Chap. 2, these two layers resemble single-loop learning (project level) and double-loop learning (both project and experience management levels).

How can we **establish feedback** from using material for activation of new experiences? Fig. 5.6 illustrates the idea.

- There are several options for eliciting feedback, and one or more of them need to be implemented.
- *Do nothing and wait*. Why should a project manager or a software engineer spend her time on providing feedback? As we will see below, appealing to altruism is

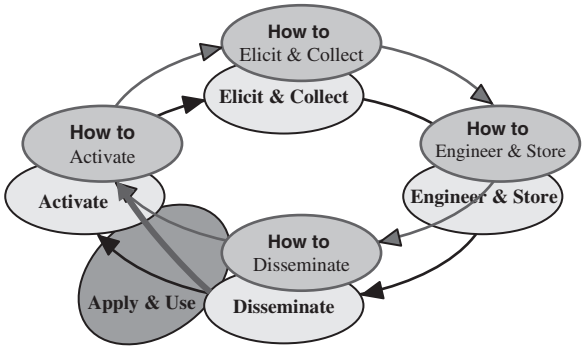

Fig. 5.6 Double-loop learning in the experience life-cycle

not a good option. Often, software engineers do not even know who to contact. And they do not know whether their feedback will be welcome. Only when people are really angry and dissatisfied will they complain on their own initiative. This is only a small fraction of the feedback you should be interested in.

- *Distribute questionnaires with the material*. This is a simple and straightforward but not necessarily the most effective approach. The entire experience (or knowledge) management life-cycle is brittle and always at risk of being interrupted. If you rely on volunteer activity like completing a questionnaire and sending it in, you are running the risk of losing potential feedback. At least, software engineers know you are looking for feedback, and where they could send it. However, you might go one step further in assisting them.
- *Web-based feedback*. At first glance, this approach looks similar to questionnaires. However, electronic feedback often poses a lower threshold to software professionals. They are used to working on the computer anyway, so a few additional keystrokes are not as bad as completing a paper form and putting it in the mail. There are several variants of Web-based feedback.
- *Contextualized feedback opportunities*. When you provide feedback over the Internet, you might have encountered the need to describe your problem or situation. This is tedious and error-prone for both sender and receiver. There are several approaches to sending e-mails or messages directly from a work environment that you might want feedback about. In that situation, the environment can add tags to indicate the position you were in when you wrote the message. Different levels of sophistication are possible. This is a research issue. Ontologies and semiautomated techniques can help to lower the threshold and improve feedback quality and quantity.
- *Calling in*. Using a phone number instead of a Web address will be appealing to some people who prefer talking to typing. Also, the receptionist can ask back and clarify things during the first contact. However, calling somebody requires time and initiative, too. If you reach a helpdesk that knows nothing about your context, this will discourage you from calling again.
- *Talking to somebody*. Experience is usually transferred by talking to people. Informal lunchtime conversation is an important channel. Direct communication is time-intensive, but for some environments and kinds of experiences, it may be worthwhile having experience and knowledge brokers. In the final chapter, an approach by Ericsson will be described that builds upon this concept [36].

Expected Contents of Feedback

The above-mentioned options describe channels for feedback. We can also provide a rough classification for the kinds of contents we might expect.

Arguing against. The most likely kind of feedback will be reasons why something did not or will not work exactly as described in a best practice. Ideally, this kind of feedback can contain constructive elements that help experience management to refine preconditions and details of use. All channels listed above can be used for

arguing against a recommendation. Experience management should not defend its material but accept this feedback as a serious and valuable contribution.

Positive feedback. In rare cases, projects will thank experience management for material like templates, hints, or warnings that helped them to work better. Of course, this kind of feedback raises confidence in experience-based material. Besides that, it usually does not contain any new details or information.

Additional experiences. When the culture in a software company is mature enough, projects will capture post-mortems or LIDs – experiences of different kinds. They send results to experience management. In this scenario, experience is not really activated by previously disseminated material; project participants carry out their duties independently.

Additional support material. Highly committed software engineers will volunteer to "improve" a best practice by sending their own templates, procedures, and support material. Experience engineering should again take those contributions seriously and make sure to thank the senders. However, sent-in material needs to be checked and evaluated before it is disseminated. Was this material ever used in practice? Is there an experience related to it or does the sender simply assume the template or process "should be better for obvious reasons"? If an experience repository contains many elements that lack an experience background, authenticity and reliability will suffer. Experience engineering will at least compare volunteered material with other experience-based results.

Cases. Sometimes, a project or person has collected rich data on a project or a certain activity. That data may be in a different format than that of experience three-tuples, hence, it may not be directly usable. However, a skilled experience engineer will gladly accept those send-ins and visit the sender. Often, there is more useful material in an environment that has realized the value of collecting case studies or data.

In general, experience management should take each contribution as a trigger to reflect on the validity of material. Those who showed commitment and contributed once should also be remembered as potential contributors in the future. Experience engineering needs to match people as well as contents.

5.4.5 Software Engineering Specifics

Note that the discussions above referred to software engineering contents. Experience and knowledge management might look very different in different domains such as insurance marketing or repair of Xerox machines.

Example 5.6 (Xerox machine: A different domain)

Xerox maintenance personnel can benefit from knowledge and experience management: Each problem encountered with a certain copier model can be stored together with the symptoms and the solution used. In all future incidents, similar symptoms can be mapped to suggest trying that solution again. Because of the large number of

incidents, statistical effects may help to see repeating success or failure. Because of the simple nature of symptoms and solutions, there are good chances to map experiences and proposed solutions to an ontology. This ontology will probably describe the structure of a Xerox machine and provide experience-based patterns matching symptoms with solutions. In this case, many of the challenges of activation, elicitation, and dissemination do not occur: A technician must provide information on each problem he worked on – regardless of motivation or activation. Dissemination can be facilitated by ontologies or other knowledge bases. This is difficult and requires good skills in the knowledge management area. Experience engineering in the Xerox machine example is still important: All formalisms used to match and deliver patterns of symptoms and solutions rely on a well-formed knowledge base. Feedback from the field of the technicians must be engineered into that format. However, Xerox-type experience management faces none of the social problems that come with software engineering experience.

Example 5.7 (Helpdesk: A different domain)

A similar situation occurs with the helpdesk of an insurance company. Agents need to search and access huge amounts of information fast. Like the Xerox technicians, however, they face numerous cases of a similar nature. The problems reported by calling customers can be indexed via their insurance products. Therefore, rather small, recurring, and similar-looking pieces of knowledge and experience need to be managed.

In software engineering, we are usually not dealing with so many, so small, or so similar incidents. A big problem in a software project may evolve over weeks or months and then lead to a long and complex experience report. Despite our definition of the experience three-tuples, software engineering observations and emotions can hardly be formalized to the same degree as in the case of insurance products or Xerox machines (Fig. 5.7).

There are examples of small-scale knowledge management in software engineering. Jäntti [58], for example, reports on knowledge management that resembles the Xerox example: There is a repository of problem reports and suggested diagnoses.

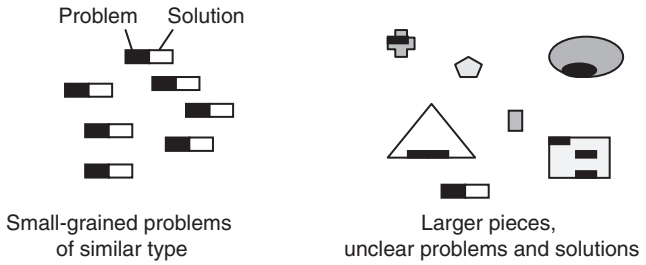

Fig. 5.7 Granularity and size of experiences: similar in routine tasks (*left*), different in software engineering (*right*)

When a program does not print, for example, an experience-based hypothesis may be: The printer module has a software bug.

The extra-fine granularity in this example might encourage a large number of small entries. However, because there is no mechanism or task for combining the microscopic "knowledge entities," this example is not very likely to fully exploit the potential of knowledge management.

5.5 Challenges to Software Engineering Experience Reuse

Before we discuss a few more misconceptions about reusing experiences and knowledge in software engineering, you should keep in mind why we tried to reuse it in the first place: If we can avoid a mistake based on previous experiences or best practices, we will gain a lot in software engineering.

Example 5.8 (Avoiding mistakes)

There are many cases in which mistakes are avoided:

- A project manager who considers a risk he might have neglected otherwise. When she is prepared for the risk, the project may take precautions. For example, an unreliable subcontractor might not be hired despite a cheaper price if he caused trouble in another project.
- Reusing a good test strategy may help a software engineer to find three errors in the code that might have gone undetected otherwise. Each error popping up at the customer costs 100 to 1000 times as much as an error fixed in advance.
- A team that finds a detailed prescription (best practice) of how to carry out reviews in a certain situation will save time and avoid failure. If the best practice does not only offer an experience report, but also all templates, procedures, and hints for an efficient review, the savings are enormous.

Benefits are multiplied if more than one project can reuse experiences. Software engineering often deals with bigger chunks of material. Matching is more difficult, and psychological barriers exist. However, even a few successful attempts pay for quite some effort invested.

At face value, experience exploitation is supposed to provide benefit for new projects based on the insights gained in past projects. However, there are several other expectations and assumptions, some of them hidden or implicit. During the **Software Experience Center (SEC)** project at DaimlerChrysler, we could often identify those expectations only when we faced them the *second* time. The most important ones of those findings were reported in [92]. The following list of expectations is taken from that publication.

The assumptions and misconceptions are clustered. They are briefly described, discussed, and a piece of advice is given on how to handle the challenge that arises from that assumption.

5.5.1 Can Experience Management Replace Experts?

When you carry out an experience management initiative, you may meet people who believe you will extract and dump all relevant experience into a repository.

This expectation is unrealistic and should not be nurtured.

It implies valuable experience is around everywhere and just needs to be put into a repository. However, eliciting, collecting, and engineering *useful* experience is not trivial. Stories about projects are often mistaken for reusable experience.

In most environments, there is a lot of experience residing in people. At first glance, the problem just seems to be picking up all that experience so it does not dissolve into vague rumors or oblivion. However, it is a popular misconception that there is much value in *raw and superfluous* experience. Not everything that comes to anybody's mind is a valuable and reusable piece of experience. A lot of experience elicitation effort is wasted when only this thin and unsettled spread of "experiences" is captured. We learned that the assumed and the perceived values of those "experiences" often deviate drastically: The bearer of the experience considered a story highly relevant experience, whereas experience managers could not translate it into anything reusable. Much more effort must be spent on analyzing raw experiences than on getting them in the first place. In the 5 years of SEC, thin-spread observations were almost never reused, whereas deep experiences going beyond rumors could effectively be turned into best practices and were actually reused.

The misconception also implies that anybody could tell all of her or his experiences. Again, this is not true. As we have seen in Chaps. 1 and 2, a lot of experience and knowledge is tacit. Experts are unaware of it. Experts often have several layers of knowledge and experience, each building on many others. There are also psychological barriers: Who wants to give all his or her knowledge away? As a consequence, a highly ambitious experience management initiative will not be able to "extract it all." A more modest attitude is more likely to succeed.

Identifying a misconception is an opportunity: We know what mistake not to make. Concrete consequences for an experience manager in the field of software engineering should be:

- Do not claim or nurture the vision of "extracting all knowledge or experience" from anybody.
- Treat experts with respect and be aware that you will still need them after an experience repository is built.
- Plan more time for engineering raw experiences into reusable material than for the elicitation of raw experiences.
- Interviews and workshops are important, but do not expect them to uncover huge amounts of valuable experiences during the first cycle. Allow for several experience life-cycles to build up, validate, and consolidate.
- Elicitation requires sophisticated approaches. A simple interview will not reach the depth of experience you might be interested in. By exposing experts to previously collected and engineered material, they can argue on a more profound level. Act as a facilitator.

5.5.2 Do Not Rely on Altruism

In the field of experience and knowledge management, many people assume others will act altruistically: For the benefit of the company or the team, they are supposed to invest time and effort but do not receive any immediate reward. You will find a number of unconscious assumptions at the root of many techniques or procedures:

- "People will fill an empty experience base."
- "Experts will document their knowledge or experience."
- "Busy experience holders will take time to talk to interviewers."
- "Experts will be glad to answer all questions, repeatedly."
- "Software engineers will encode their insights in OWL."
- "Contributors will check whether experience engineering has correctly presented their (previously donated) raw material."

Some of these assumptions sound appealing, but they are often wrong. Researchers and managers both tend to assume that business unit employees see the need for putting experiences into a repository or telling an experience management support group before they can get something back from there – later.

This is sometimes true, but often it is not. We met a few employees who had embraced the concept of experience exploitation. They accepted to talk to experience management several times just to provide information – and they did not expect to get something back in return. At the same time we had drastically overestimated the patience of many employees. Even those willing to contribute limit the amount of effort spent.

There are two obvious reasons against sustained altruism:

- Sheer amount of work. Software engineering is a busy field, and there are always deadlines and time pressure. Squeezing in even a few extra minutes is a luxury. When we reach overtime – and we always do – the question boils down to: Why should an expert spend 20 minutes on unpaid, volunteer experience validation rather than on playing with his children?
- Loss of status. Knowledge is power, and so is experience. Giving away knowledge or power means lowering one's own value. Who knows who will be laid off when business is slower? This is a classic dilemma hampering knowledge management. It has many facets, and it cannot be honestly resolved. There is some truth to the concerns, but in most cases, experts do not suffer disadvantages when they give some of their knowledge away – they are recognized as experienced and knowledgeable experts which often raises their status.

Both reasons need to be considered. A successful initiative needs to find an individual, tailored answer to these concerns. The workload of experts must be minimized, at least for the experience management work they do: Use assistants and tools and sophisticated techniques to save hours and even minutes. The second problem will always require management commitment. Volunteer contributors must (must!) not have disadvantages. Details cannot be solved in general.

Incentives are often proposed to provide rewards. If a contribution to a knowledge base or an experience repository is paid in money, there is an obvious reward. The balance of effort and reward can be restored. This is a good idea and should be considered. However, incentive systems can become counterproductive quite easily, and in a surprising way, as Davenport [26] describes: At Siemens, there were different incentive systems.

- It turned out that low incentives might not convince people to do extra work if they would not do it anyway and without incentive.
- However, generous financial incentives evoked incentive-optimization strategies by some employees. When packages were counted, they tended to contribute two small experience packages rather than a big one. They wrote sloppier reports when contents were not evaluated. In short, substantial incentives corrupt the system as many software metrics do: People are intrigued to optimize their income, not the value they add.
- One attempt was to reward both experience contributions and the usage of experience – those packages that were reused more often received higher incentives. But reusers had advantages, too. You can imagine the consequence: Downloads soar and circles of friends cross-optimize their benefits. None of those phenomena needs to be criminal or even mean in nature. Optimizing advantage is a part of human nature. However, experience managers should exploit that trend in a responsible way, not deny it or complain about it.
- Not all incentives need to be financial. It is a breakthrough to include social recognition like awards and honors into the reward system. More than many managers might think, social recognition can be gained when experiences are not disseminated anonymously. Ambitious experts like to be recognized as the author of many contributions. They would never turn in bad quality under their own name.

The open source or Wikipedia community seems to operate on a similar basis. Altruism occurs, and it is an important ingredient of open platforms. However, altruism is associated with volunteer activity. By definition, no one can be forced to volunteer information. It cannot be part of a job description to work without reward. In fact, open contribution systems provide **social recognition as a valuable reward**, together with complete flexibility of work. A good experience and knowledge management initiative will learn from those communities and build a similar reward system.

Mnemonic 5.2 (Altruism)

We should be glad to receive altruistic contributions. But an experience and knowledge management initiative cannot rely on it. Plans and operations should be designed to run without altruism.

Again, there are concrete consequences for experience management:

- Consider efforts and rewards carefully. When you plan procedures for elicitation and dissemination, make sure each role and person has a fair balance of investments and benefit.

- When you design an elicitation technique, make sure to optimize it for individual effort! Those providing input must be saved every single unnecessary task and step. Administrative tasks should be shifted to the experience personnel instead of "asking the experts to classify. . .," and so forth.
- Rethink each and every procedure that will not work without altruism. Essential procedures that build on altruism are bound to fail [50].
- Experience managers are paid for running the initiative. Their reward for eliciting, engineering, and fine-tuning activities lies in a successful operation of the initiative. For this group, effort and benefit appear in different forms than for participating software engineers.
- Install a moderate incentive system. Incentives should appeal psychologically, but high financial benefits may be counterproductive. Reward contributions by making them public.
- Consider carefully what material you want to identify by author: It may provide social rewards, but some experiences need to remain anonymous. Others will be absorbed into best practices – maybe main contributors should be listed on the derived document.

The problem of motivation and incentives is delicate. After looking into concerns and problems, we should mention one more element that is crucial for experience reuse: **intrinsic motivation** of talented and committed software engineers. No extrinsic (coming from the outside) incentive or reward system can compensate for missing intrinsic (internal) motivation. Intrinsically motivated employees are among the most valuable assets a company can have. Implications cannot be discussed in detail in this book. Nevertheless, intrinsic motivation is a good starting point for effective experience work. The above-mentioned considerations and pieces of advice will help to sustain it.

5.5.3 Barriers to Experience Flow

Those interested in experience management assume: "Some people document their experiences, others read and reuse them." Surprisingly, even this basic assumption is often not fulfilled.

Example 5.9 (Fears and barriers)

As we have seen above, effort may be a barrier to documenting experience. There needs to be some incentive or motivation. One of the barriers mentioned above was a reluctance to give away experience and knowledge. Software engineers may fear losing some of their value for the company when they give away their experience or knowledge. There are also concerns on a psychological level concerning power and influence. Rare knowledge or experience can be a source of power, and experts can receive advantages and benefits in exchange for sharing their knowledge. What

happens if that knowledge is now stored in a repository? Those concerns are easy to understand.

There are also barriers to reusing experience. This is more surprising, as the intention of reuse is to save time and effort. Who would not want that?

Information overflow. Many researchers and authors of experience-based material consider their task completed when they have inserted their contribution into a database, experience base, or intranet page. This assumption is widespread, but it is tragically wrong. As others have pointed out in their domains [80], there are so many sources of information on a wide variety of media that most people suffer from an information overflow. As a consequence, most "available" information must be ignored.

Not aware of matching information. Making information "available" is just not good enough. Software quality agents really need a piece of experience at the time and in the way they can best integrate it into their respective work assignment. The tragic element lies in the fact that excellent material can sleep on a wonderful experience base system – without ever being found and used by those who desperately need it. In SEC, we reacted by not only advertising our experience bases but also by tightly integrating them into a system of training courses, newsletter notes, and information briefings for the subject conveyed (e.g., risk management) [65].

Doubt about reliability/not invented here syndrome. Researchers and managers assume that the need for experiences is obvious. Reuse seems to be the obvious solution, and experience exploitation seems self-supportive in nature. However, some people do not believe in the quality of material they do not know.

Example 5.10 (No reuse)

It came as a surprise to us in SEC when people were often very open and giving in interviews but reluctant to search, read, or apply any material we had retrieved from others. Our conclusion was that they did not trust the quality of the offered experiences. This phenomenon is well-known as the "not invented here syndrome."

Reuse needs to be encouraged actively. There are different approaches to implementing this attitude: Incentives for reusers is one. Using existing personal contacts for recommending reusable material is probably more effective. In this perspective, workshops, quality circles, and, in particular, coaching and consulting situations can be used not only to *capture* experience but also to discuss and to *spread* it.

5.5.4 Essential: Management Support

When there is not sufficient support from management, some software engineers consider starting knowledge management initiatives bottom-up. They hope that small and local successes will grow into a big initiative and incur increasing management support.

Management support and commitment are essential in many activities. Experience and knowledge management is impossible without strong and visible management backing. Financial support and resources are important. If there is not sufficient money for hiring experience managers or experience engineers, respective activities fade away. While writing an experience report could be done on the side, experience engineering is a serious and demanding job that requires time, dedication, and qualification as an experience and knowledge manager.

Management *backing* is equally important: There will be times when experience management seems to delay a project or an activity. Benefits will not arise instantaneously, as experience engineering needs to collect some experiences before it can compare and disseminate them. Initial effort to start an initiative is high, and it requires some patience. Impatient management that withdraws resources and trust will kill each initiative.

It is sometimes difficult to convince higher management to spend money and free valuable resources for the sake of "cross-functional" activities like experience exploitation. Business units tend to start with bottom-up activities (e.g., quality circles, volunteer documentation, etc.), hoping for more management commitment later. Beginning successes are supposed to open the door for larger-scale activities. In SEC, many of those basic support activities were rather successful (i.e., produced benefit for the business unit).

However, bottom-up activities seldom exceeded local impact. Management commitment rarely grew gradually. Local activities were welcome but not perceived as a first step toward a bigger initiative. They differed in scope, goals, participants, and structures.

A pure bottom-up approach will hardly exceed its initial level. There seems to be a critical mass of investment in experience exploitation. With a small number of potential experience reusers, success is much less likely. Investment, effort invested into discussions, and strategic attention are also needed. Below a certain threshold, most activities will vanish. Above the threshold, there is a tendency for mutual reinforcement.

The conclusions from this challenge sound straight-forware, but are difficult to follow:

- An experience and knowledge management (EKM) initiative will not succeed if there is no real and active management commitment.
- When starting bottom-up, activities will most probably remain on the initial level and may be hard to sustain, due to a lack of critical mass.
- Management commitment can be measured by resources assigned to the initiative and by explicit public statements made to the employees supporting the initiative.
- If you do not have management commitment and do not want to shift to bottom-up software process improvement, you should not attempt to start an EKM initiative. It is not worth the effort.

Personal commitment will work on a small scale: If you are happy with a Web site or a Wiki for exchanging news and opinions, you may generate substantial benefit

for yourself and some co-workers. However, this hardly qualifies as experience or knowledge management.

When you join a working EKM initiative, you are much better off. A working reuse cycle does not need as much energy to sustain. Also, individuals may decide to run a very modest and small-scale initiative just for their own benefit. Countless examples show that the (moderate) effort invested is often rewarded by reasonable benefit. The concerns about building an EKM initiative bottom-up are mostly associated with a large scale and with critical mass effects.

5.6 Problems for Chapter 5

Problem 5.6.1: Life cycle

Draw the experience life-cycle and briefly explain each activity.

Problem 5.6.2: Experience engineering

Assume your team is using a new framework for software development. Programmers report problems they have had. If possible, they also report how they solved them in the end. What should experience engineering do with those reports and what roles can patterns play?

Problem 5.6.3: Experience three-tuple

Describe one of your own experiences in software engineering as an experience three-tuple. Make sure you do not neglect the emotion!

Problem 5.6.4: Best practice

All projects in a business unit need to follow a new process. The first project using it reports some experiences. Why can experience engineering usually not be shortcut by asking the projects to provide best practices right away?

Problem 5.6.5: Contradictory experiences

Two testers report experiences on a test tool. Tester A is happy, because the tool helped him to create far more test cases than usual. A larger number of errors were found early in the testing process, and late testing phases were far more relaxed. Tester B, however, was disappointed because the tool provided only "trivial test cases," as he puts it. Assume you are an experience engineer and need to make sense of those contradictory experiences. Give at least two potential explanations for the different experiences! If both testers A and B are equally competent, what could experience engineering ideally try to derive from their experiences?

Problem 5.6.6: Delicate experience

Why is experience an even more delicate matter than factual knowledge? Describe two different kinds of experience: (1) one highly critical kind of concrete experience that would be very helpful to reuse, but that will be very hard to elicit; (2) one kind of knowledge that should be easy to get (and useful to reuse). As an experience engineer, what kind of experience would you focus on first?

Problem 5.6.7: Argue well

How do you react if your boss asks you to start developing a knowledge-building strategy for your team of eight people – with the option of spreading it across the entire business unit if it really provides substantial benefits. Once you reach that level, he promises to provide additional resources and promote you to "knowledge expert." Sketch your argumentation!

http://www.springer.com/978-3-540-95879-6

Experience and Knowledge Management in Software Engineering

Schneider, K.

2009, XVI, 235 p., Hardcover
ISBN: 978-3-540-95879-6