Fraunhofer Einrichtung
Experimentelles
Software Engineering

Knowledge Management of Software Engineering Lessons Learned

Technical Report

Authors:
Andreas Birk,
Carsten Tautz

Abridged version accepted for publication by the 10th International Conference of Software Engineering and Knowledge Engineering (SEKE '98)

IESE-Report No. 002.98/E
Version 1.0
July 1998

A publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competetive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach
Sauerwiesen 6
D-67661 Kaiserslautern

Executive Summary

Reuse of lessons learned from past software projects promotes good software development practices and prevents the repetition of mistakes. However, the lessons learned during a software project remain far too often isolated in the minds of the project participants. There is a need to systematically gain, store, disseminate, and apply them.

This report presents a method for integrated knowledge management of lessons learned in software engineering that covers all these aspects. The method bridges the gap between those people and situations that gain lessons learned and those that need them. This requires the transformation of arbitrary experience statements into well-structured and explicitly represented lessons learned.

The report describes how to structure and represent lessons learned. It also presents a technical infrastructure for storing, disseminating, and applying them. Requirements and architecture for a knowledge management system are described, and an organizational infrastructure for knowledge management of lessons learned is suggested.

The method can be tailored to arbitrary organizational environments and usage scenarios. It can be gradually extended towards a comprehensive knowledge management system. Recommendations for managerial aspects are given that should be considered when lessons learned management is put into practice. The presented method has been developed and evaluated through three application cases in academic and industrial environments.

Copyright © Fraunhofer IESE 1998

Copyright © Fraunhofer IESE 1998

Table of Contents

Copyright © Fraunhofer IESE 1998

Copyright © Fraunhofer IESE 1998

1 Introduction

Lessons learned that are gained during software projects remain far too often isolated in the minds of the project participants. Sometimes, post-mortem meetings are performed for exchange of lessons learned between team members. But even then, later reuse of relevant lessons learned throughout the entire software organization can not be achieved systematically because the lessons learned stay invisible for the rest of the organization. When reused systematically, lessons learned promote good practices and prevent the repetition of mistakes.

What is needed is an effective approach to gaining, storing, disseminating, and applying lessons learned from software projects. Lessons learned programs[1] can be set up that improve guidance of present and future software developments throughout an organization. Lessons learned are gained from software projects and processed ("packaged") in order to make their later retrieval and use as easy as possible.

Lessons learned management[2] can very well serve as a catalyst towards establishing comprehensive knowledge management systems in software organizations. It picks up a latent need for more effective use of existing experiences. A knowledge management system can be developed that is integrated in the regular business processes and that can grow gradually from paper-based documents via on-line documents to formal knowledge representation. As the degree of formalization grows and personnel gets used to handling explicitly represented knowledge, it will be possible to gradually introduce higher levels of automated knowledge-based support.

Lessons learned management aims at building a bridge between persons who gain experience and persons who can benefit from these experiences. It also bridges the gap between situations in which lessons learned are gained and those in which they are needed. A set of coherent solutions to close these gaps constitutes a method for lessons learned management: Formalisms for gaining, packaging, disseminating, and using (i.e., applying) lessons learned, complemented by knowledge representation and technical infrastructure (i.e., storage)

1 A lessons learned program is an initiative of a software development organization to capture software engineering lessons learned and to make use of them. It can be seen as part of a more comprehensive process improvement program.

2 In this report *lessons learned management* and *knowledge management of lessons learned* are used as synonyms.

Copyright © Fraunhofer IESE 1998

for lessons learned management. This report presents such a method. In addition, it gives recommendations for implementing effective lessons learned management and provides an automated support infrastructure.

The report is structured through three application cases in which we have developed and evaluated our method. Section 2 introduces the cases while Section 3 elaborates on the specific problems and solution possibilities for gaining and packaging lessons learned. Procedures for gaining and packaging lessons learned are derived from the individual cases. Representation and usage of lessons learned are addressed in Section 4. A model is developed by starting with the application cases. Storage (i.e., the technical infrastructure) and dissemination of lessons learned are subject of Section 5. Section 6 addresses the application of lessons learned management for arbitrary situations in software engineering. It describes how the presented method can be tailored to specific application contexts. This report is concluded by comparing the presented lessons learned management to related approaches.

Copyright © Fraunhofer IESE 1998

2 Application Cases for Lessons Learned

This section presents three application cases in which we have developed specific lessons learned management solutions: The implementation of systematic inspections at a large German company, application of a requirements engineering technique in the software laboratory of the University of Kaiserslautern, and usage of the GQM approach for goal-oriented measurement at European software companies in the context of three ESPRIT and ESSI projects. For each application case, the specific difficulties and implementation possibilities for lessons learned management are demonstrated.

2.1 Application Case 1: Systematic Inspections

In this case we have introduced systematic inspections to a large German company. Inspections are a static analysis technique to find defects in software development documents [GG93]. During the performance of inspections, some unexpected discoveries were made requiring immediate, project-specific solutions. For instance, during the inspection of a requirements document it was discovered that the document contained too many defects to be discussed during a single inspection meeting. Therefore, the defects identified by the readers were grouped into seven categories. For each category (representing one problem type), a solution was devised.

We investigated these problem/solution situations further, generalized them – where possible –, and thus derived (generalized) lessons learned. These help future projects to detect problem situations early or to avoid them entirely. Another benefit is that solutions for similar problems may be derived from past problem/solution situations [AP94].

In order to use lessons learned they have to be organized in a way that they can be found on demand. In this case the lessons learned were collected in a report and grouped according to the topics "inspections" and "requirements". This report can be read by project managers enabling them to recognize and correct problems early or avoiding them entirely. The installation of an on-line retrieval system over the intranet based on WWW technology is planned.

2.2 Application Case 2: Requirements Engineering

With this case we have started our work on lessons learned management. A set of experience statements had been produced by a software development project in the software engineering laboratory at the University of Kaiserslaut-

Copyright © Fraunhofer IESE 1998

ern. This project had used some state-of-the-art software engineering technologies and tried them under project conditions. Particular focus was put on the NRL requirements engineering method [CP93, vS93]. Experiences from using NRL were collected in a post-mortem meeting. It turned out that these experiences were not yet in reusable form. There also were different categories of experiences. In our case study, we categorized the experiences, developed a representation structure for each category, packaged the experiences for reuse, and developed a prototype experience base tool for input, storage, and retrieval of lessons learned via the web.

2.3 Application Case 3: GQM-Based Measurement

Lessons learned with software measurement programs have been gained during several European applied research projects[1]. The measurement programs were conducted according to the Goal/Question/Metric (GQM) approach [BCR94, BDR97]. We have started to systematically process these lessons learned in order to supply a new level of guidance for the GQM process. The GQM process is defined in a handbook as a stepwise procedure that accompanies software projects. Each process step can have one or more refinement levels [GHW95]. Lessons learned are now attached to the process steps. They provide recommended solutions for specific situations and difficulties based on the experiences of past measurement programs.

1 E.g., ESPRIT projects PERFECT (no. 9090) and PROFES (no. 23239), and ESSI project CEMP (no. 10358).

Copyright © Fraunhofer IESE 1998

2.4 Summary of Application Cases

Table 1 summarizes the application cases.

Table 1:Summary of application cases

	Application Case 1: Systematic Inspections	Application Case 2: Requirements Engineering	Application Case 3: GQM-Based Measurement
Goals	Recognize problem situations early Prevent recurrence of problems Solve problems effectively	Process experiences for use in later projects Improve technology Achieve scientific results	Evaluate method Improve method Guide future usage of method
Input	Memos from the project team (narrative text)	Recording from post mortem meetings (narrative text)	Presentation material, reports, interview results, questionnaire results
Output	Report (structured text)	Report (structured text) and object-oriented knowledge representation	Report (handbook-style) and hypertext (process guidelines)

Copyright © Fraunhofer IESE 1998

3 Gaining and Packaging Lessons Learned

Based on the three cases presented in the previous section, an approach for gaining and packaging lessons learned is derived. We subsume all activities regarding the recording of lessons learned which are directly related to normal project work (e.g., writing minutes of project meetings) under "gaining lessons learned". We refer to all activities as "packaging" which are related to making lessons learned reusable for future projects. While the software development teams of an organization are heavily involved in gaining lessons learned, the packaging of lessons learned is typically done by people outside the development team because making lessons learned reusable for other projects is not of interest to an ongoing development project (see Section 6 for details).

3.1 Application Case 1: Systematic Inspections

During the reading of a requirements document many inspection items were found which were to be discussed during the upcoming inspection meeting. As it turned out, there were many similar inspection items, which could be grouped. For each group an associated problem type was postulated. For example, for the group "low comprehensibility" the problem "requirements which are not completely understood may hide technical problems" was postulated. The solution devised was that the requirements which were not completely understood by the development team should be discussed between Mr. X (the project manager of the software development team) and department Y (representing the authors of the requirements which were not part of the software development team). Possible technical problems which would become apparent should then be resolved in individual negotiations.

Even though this solution seems natural, the recognition of the described potential problem alone may save lots of rework in other projects if it is captured as a lesson learned. For this purpose, the project-specific experience statement *gained* must be *packaged* in a way that it is of interest for other projects.

- Input: Informal project-specific decisions.
- Process description:
  1. Abstract from project-specific information ("decontextualize"). For the example given above, the people's and departments' names have to be removed and be replaced by their functions.
  2. Complete lessons (e.g., who negotiates solutions to technical problems?)

Copyright © Fraunhofer IESE 1998

3 Make lessons consistent; use a standard vocabulary (e.g., the same terms should have the same meaning across all lessons learned, synonyms should be avoided)

- Output: set of lessons learned.

The major characteristic of this packaging process is that project-specific information is removed from the informal statements supplied by a project which in turn may require a completion or generalization of the statements. In addition all lessons learned should have a uniform style as to avoid misunderstandings at usage time.

3.2 Application Case 2: Requirements Engineering

The experience statements were collected during a project post-mortem meeting. The statements from the project participants were recorded by a moderator. An example experience statement was:

"Internal functions (e.g., 'next') can not be distinguished by their form from user-defined functions. It is not known to me whether there exists a list of the predefined functions in NRL and how these are defined."

This statement shows several characteristics that limit its reusability: It addresses multiple aspects or facets of information (form of "internal functions" in NRL and whether the "internal functions" have been made known to the development team); it contains uncommon terminology ("internal functions" is not a common term in NRL; "predefined functions" would be appropriate); there is no hint for a later user about when this lesson learned can be useful and when it should be made known to other developers; finally, it is not fully clear at a first glance whether the identified issue is actually a problem or due to a misunderstanding of the developer (i.e., a *validation* of the experience statement is needed). These deficiencies needed to be removed by the packaging process.

The packaging process makes some assumptions about the representation. Basically, these consist of different categories of lessons learned where each category serves a different purpose, i.e., a different usage scenario. The categories differ in the facets (title, topic addressed, context situation in which the lesson learned is relevant, etc.) associated with them. This representation is described in more detail in Section 4.2. Given these assumptions, the packaging process has been as follows:

- Input: Informal experience statement as it was recorded in a project post-mortem meeting

Copyright © Fraunhofer IESE 1998

- Given: Representation that distinguishes several categories of experience statements and lessons learned according to their information contents where several facets are associated with each category of lesson learned
- Process description
  1. Validate and if needed correct or discard the experience statement
  2. Identify types of information contained in the experience statement
  3. Split experience statement such that each resulting part only addresses one type of content information
  4. Assign each resulting partial statement to the category of experience statements that corresponds to its content information type
  5. Rephrase each partial statement such that it is a complete sentence
  6. Standardize the terms used in the experience statements
  7. Supply all information to each statement that is needed to make it self-contained ("decontextualize")
  8. Supply to each statement an explicit description of the context situation[1] in which it is relevant
  9. Distribute the information in each statement over its category's facets and provide still lacking information for those facets that are not yet filled
  10. Rephrase the statements such that they have a uniform style (this enhances readability in the later reuse situation)
- Output: A structured set of reusable lessons learned

Major characteristics of this packaging process are that experience statements are validated; that they are made self-contained, clearly understandable, and uniform; that they are assigned to predefined categories according to certain information types; that they are supplied with a definition of the context situa-

1 An experience statement is only valid in the context it was gained in. The context is typically described by the (a) characteristics of the project (e.g., size of the project team, duration of the project) and (b) the status of the project (e.g., the design is finished). This is different from the project-specific information (e.g., specific names of people on the project team) which is removed as part of "decontextualize".

Copyright © Fraunhofer IESE 1998

tion in which they are relevant; and that they contain various facets of information that should be marked explicitly in order to facilitate their reuse.

3.3 Application Case 3: GQM-Based Measurement

The experiences were reported in various forms: as statements during feedback meetings, through questionnaires, in reports, or through application presentations. They had arbitrary form ranging from vague statements to well-structured problem/solution pairs with context description.

For instance, in the project reports, the experiences[1] were reported as list items. The individual items were a few sentences in plain English with arbitrary structure and content type. They were reported from different application projects by different people.

Using these inputs, the following packaging process was used initially:

- Input: Arbitrary experience statements
- Given: A GQM process model describing how to plan and perform measurement programs.
- Process description:
  1. Review the experience statements from each project
  2. Discard those with low relevance
  3. Assign the remaining ones to the process steps of the GQM process model
  4. Give statements a uniform style (i.e., decontextualize and rephrase)
  5. Merge the lessons learned from the individual projects into one compound experience report
- Output: Compound experience report containing a set of lessons learned ordered according to the steps of the GQM process model.

1 These were experience statements about GQM-based measurement programs from six application projects of ESPRIT project PERFECT. The experience statements were at that time not yet represented as problem/solution pairs. The objective was to identify action points for needed refinements of the GQM process description.

Copyright © Fraunhofer IESE 1998

In a later stage, the selected experience statements were reformulated as problem/solution pairs and integrated with the existing GQM process model in form of a handbook with then integrated, experience-based guidelines.

Major characteristics of this packaging process are that it associates with each lesson learned an object (in this case a GQM process step), and that the result is not a set of individual lessons learned but rather a compound experience report.[1]

3.4 Approach to Gaining and Packaging Lessons Learned

This section provides the summarized findings and results for gaining and packaging lessons learned in software engineering. First, it addresses the underlying issue of the logical structure of lessons learned. Afterwards, procedures for gaining and packaging lessons learned are presented.

3.4.1 Logical Structure of Lessons Learned

In order for the processes explained in the next subsections to be meaningful certain assumptions about the representation of lessons learned have to be made. We call the set of these assumptions the *logical structure* of lessons learned.

The three application cases have shown that the management of lessons learned requires to consider and represent at least the following concepts:

- Object of lesson learned
- Problem
- Solution
- Context

Object of a lesson learned is the software engineering artifact that the lesson learned is about. It can be a process, a product, a technique or method, some policy, etc. Problem and solution are the core parts of a lesson learned. Context

1 From the knowledge engineering point of view, this presents a way of generating detailed general knowledge by taking general knowledge (process model) and case-specific knowledge (lessons learned), structuring and reformulating the case-specific knowledge, and finally merging them.

Copyright © Fraunhofer IESE 1998

describes the situation in which a problem/solution pair is relevant (e.g., small maintenance projects or large projects with stringent time requirements).

Depending on the objectives of the lessons learned program, the concrete manifestation of these concepts may vary. For instance, in the inspections case, lessons learned were represented explicitly as problem/solution pairs in order to provide support for trouble shooting when problems with inspections occurred. Whereas in the requirements engineering case, focus was on avoiding problems by providing improvements of the requirements engineering method and direct usage guidelines. This led to focus on the solution side and not emphasize the problem side in the structure. Of course, these aspects can also be integrated into a comprehensive knowledge representation. We will go into more detail in Section 4.4.

A concrete representation will also associate certain facets to the concepts such as the title of a lesson learned or information about how it has been derived. In the following subsections we abstract from these facets.

3.4.2 How to Gain Lessons Learned

There are three basic strategies to gaining experience statements:

- *Use available technical (i.e., non-human) knowledge sources.* Rely on already existing experience reports that are not yet processed into reusable form. Such experience reports can be project memos, presentation slides, the minutes from project post-mortem meetings or GQM feedback sessions [GHW95], or the results from project or technology evaluations.

- *Use goal-oriented knowledge acquisition (centralized knowledge elicitation).* Conduct investigations specifically for the purpose of identifying lessons learned. Such investigations can utilize interviews, questionnaires, surveys, brainstorming meetings, capturing of ad-hoc statements during project work, etc.

- *Accumulate knowledge during everyday work (decentralized knowledge elicitation).* Establish a framework and tool environment where people can enter already well-structured and readily reusable experience statements whenever they encounter them during their regular project work.

The latter strategy is the ultimate goal of lessons learned management. It is the vision of a knowledge management system that is fully integrated with its organizational context and represents a prototype infrastructure for learning organizations. However, the usual case to start lessons learned management is to acquire already existing experiences that are not yet in reusable form and pack-

Copyright © Fraunhofer IESE 1998

age them as lessons learned. This approach can then be extended gradually towards an integrated knowledge management system.

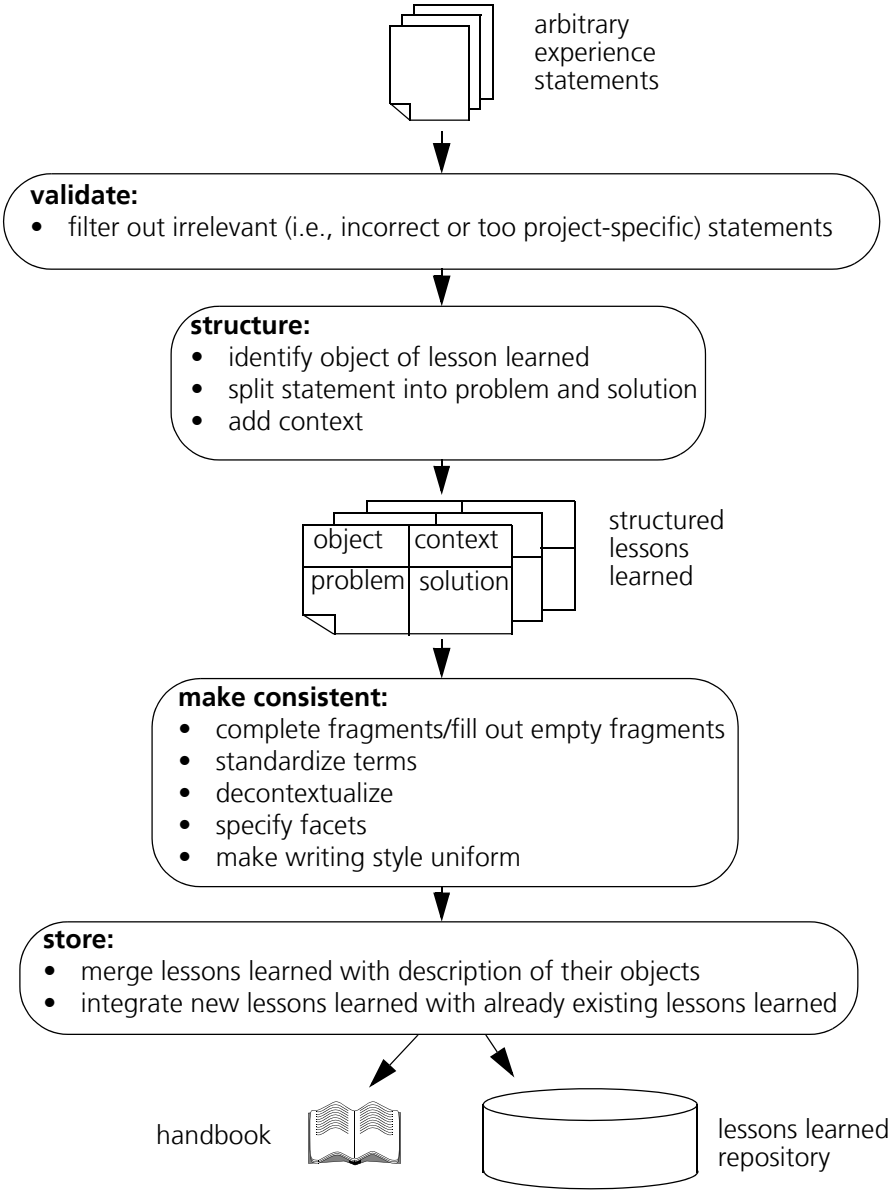
3.4.3 How to Package Lessons Learned

Based on the three application cases a packaging process can be synthesized. The packaging process converts arbitrary experience statements into structured lessons learned. Figure 1 depicts this process. While application case 2 gives a detailed description on how to package informal experience statements, the application case 3 emphasizes that each experience statement is about some kind of object which has to be captured. The object of the lesson learned offers a simple way to filter out irrelevant lessons learned at usage time. For instance, when dealing with requirements, lessons learned about the architectural design may safely be ignored. Also, application case 3 goes one step further than simply deriving lessons learned. There, the lessons learned produced are integrated into a handbook together with a description of the objects the lessons learned are about. This handbook is very effective for training people because it contains both the way a technology (e.g., GQM) should be applied and company-specific pitfalls (lessons learned about certain aspects of the technology). Another option for storage is an on-line repository for lessons learned as a prerequisite for tool support (cf. application case 2).

In summary, the packaging process must convert arbitrary experience statements into structured lessons learned where the structure is defined by the categories of the lessons learned and their facets.

Copyright © Fraunhofer IESE 1998

Figure 1:Process for packaging lessons learned



Copyright © Fraunhofer IESE 1998

4 Representing and Using Lessons Learned

In this section we introduce a representation for lessons learned based on the usage requirements of the application cases because ultimately the way lessons learned are used determines what an adequate representation looks like. This also means that it is not possible to develop a knowledge representation which can be used in all cases. However, there is common ground among all representations, and it is the aim of this section to point it out. The result is a representation which can be easily adapted to specific needs.

4.1 Application Case 1: Systematic Inspections

The lessons learned from the inspections are used as guidelines (for avoiding problems which occurred in past projects) and as suggestions on how to solve problems occurring in the project. In the first case, relevant guidelines are selected apriori (e.g., at the start of an inspection) by checking whether the problem addressed might become relevant for the project. The selected guidelines then become project guidelines. In the latter case, relevant problem/solution pairs are selected on demand by matching the description of the actual problem at hand with the problem descriptions of the lessons learned. In this case, the *problem* description must include a description of the *problem situation* because the way a problem can be solved depends upon the context the problem occurred in. For example, solutions for technical problems concerning the implementation of requirements need not to be negotiated between different parties if the requirements are written by the same team as the system is implemented by.

In our application case we used the form illustrated in Figure 2 for the representation of lessons learned. The issue summarizes the contents of the lesson learned. This allows a coarse-grained search to find out whether the lesson learned is potentially applicable. Context and status describe the problem situation, i.e., they define in detail when the lesson learned is (known to be) applicable. The effects state problems which have occurred in the past as a result of the described situation. The (suggested) correction is a proposed solution on how to handle the problem situation properly. Finally, the guideline suggests how to avoid the problem all together.

Copyright © Fraunhofer IESE 1998

Figure 2:Form used for recording lessons learned

Issue: Poorly understood problems may hide technical problems.

Context: <company/division/department> <project>

Status: The requirements documents (not written by the development team) contain many requirements which are not completely understood by the development team.

Effects: The poorly understood requirements may lead to unrecognized technical problems for the implementation.

(Suggested) correction: The project manager of the development team discusses the poorly understood requirements with the authors of the requirements documents. Technical problems recognized through these discussions should be negotiated between all 'consumers' of the requirements documents.

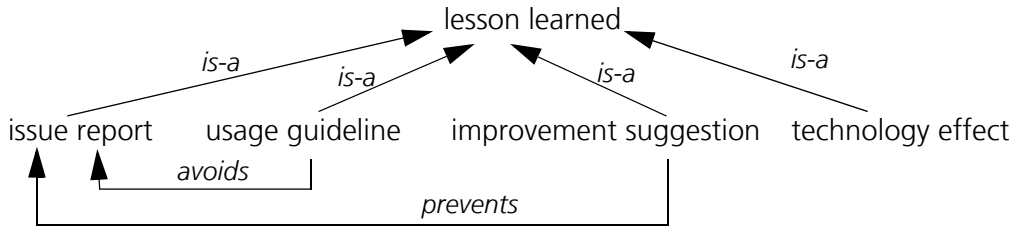
The storage chosen for the lessons learned was – as mentioned in Section 2.1 – in form of a report. For finding relevant guidelines, the list of lessons learned has to be flipped through. For each lesson learned its applicability is decided by making assumptions about the probability that the described problem situation may actually occur. If the probability is rated as high, the stated guideline becomes part of the project guidelines. Similarly, if a problem (situation) occurs (e.g., poorly understood requirements), the lessons learned are flipped through to see whether a lesson learned is applicable. If so, its solution is adapted to the needs of the current project and the modified solution is applied. Note, that a problem description may also be found under "status" because a problem situation may lead to a problem. Applying the lesson learned when the situation described under "status" has been detected will prevent the occurrence of the problem described under "effects". Thus, the occurrence of the problem situation can be interpreted as a (minor) problem itself.

4.2 Application Case 2: Requirements Engineering

The representation that has been gained for the NRL case is depicted in Figure 3.

Copyright © Fraunhofer IESE 1998

Figure 3:Categories of lessons learned as they have been identified in the requirements engineering case



It contains four categories of lessons learned: issue report, usage guideline, improvement suggestion, and technology effect. Issue reports are warnings and descriptions of problems that have occurred during application of the software development practice. Usage guidelines are hints to later users of the software development practice that aim at avoiding possible problems. While usage guidelines and issue reports should be read before using a technology, improvement suggestions address updates and refinements of the definition of the software development practice in order to facilitate later applications. Such improvements remove the root cause of occurred problems and thus prevent the recurrence of these problems. Technology effects refer to impact and interrelations that the software development technology has on other aspects of the software process, e.g., particular contributions to achievement of certain product qualities (e.g., high reliability) or impact on time and effort needed for software development (e.g., a particular software development practice improves the domain knowledge of the engineers and thus reduces further communication needs, development time, and effort).

The categories of the lessons learned have been developed through the following process:

- Input: A set of experience statements as they were recorded in a project postmortem meeting
- Process description:
  1 Identify future usage needs and requirements for the lessons learned
  2 Define conceptual model of the usage context of the lessons learned
  3 Identify categories of lessons learned by aid of usage needs and conceptual model
  4 Categorize the experience statements and evaluate the categorization scheme
  5 If needed, modify and refine the characterization

Copyright © Fraunhofer IESE 1998

- Result: A categorization of lessons learned

In a next step, there was defined an internal facet structure for each category of lessons learned. The facets could easily be derived from the identified usage needs and the conceptual model. Objective was to achieve a uniform and easy to perceive representation. Finally, the actual representation schema was defined using an object-oriented formalism.

A prototype software tool has been developed for storing the lessons learned. It is implemented in Java and uses an object-oriented representation formalism. It offers user interfaces for input and retrieval of lessons learned. Figure 4 depicts a window for displaying a lessons learned of type usage guideline and shows the facets used for structuring the information. The tool is designed such that it can be integrated in web pages and become part of a company-wide intranet.

Copyright © Fraunhofer IESE 1998

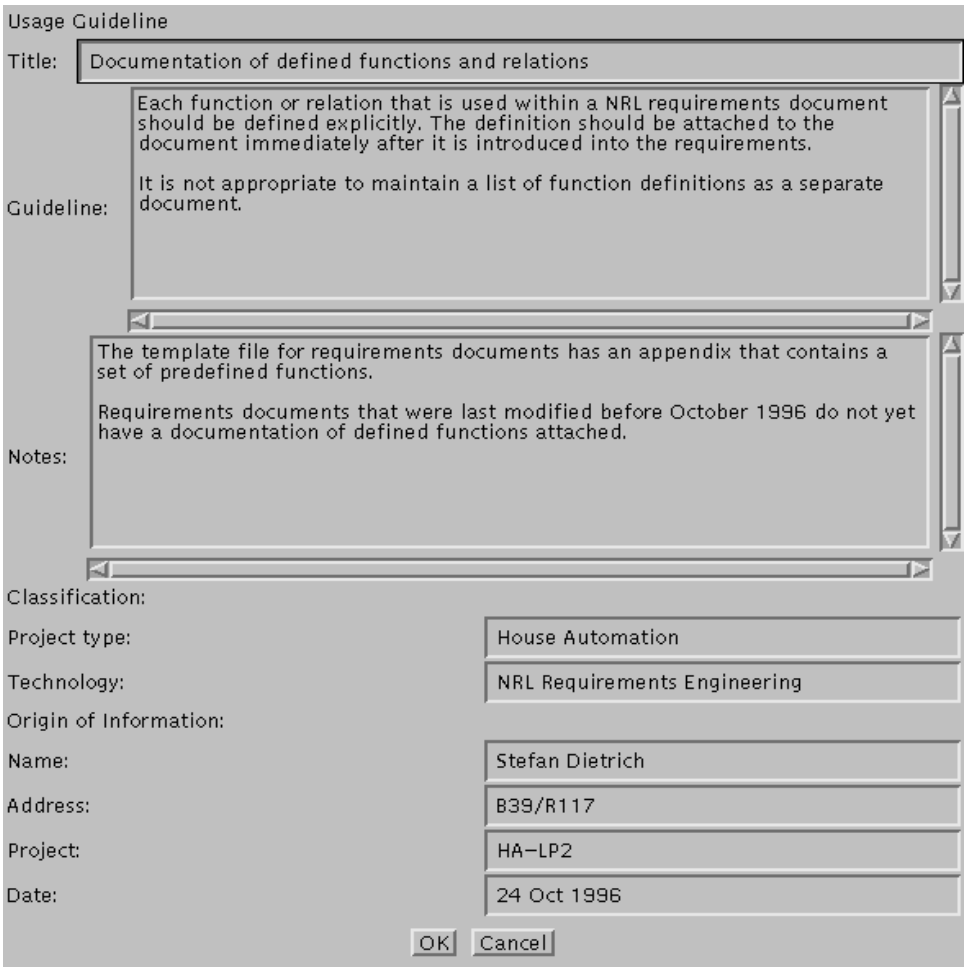


Figure 4:Window for displaying retrieved lessons learned of category usage guideline as provided by the prototype knowledge management system for the requirements engineering case

4.3 Application Case 3: GQM-Based Measurement

The completely processed lessons learned have seven facets (cf. Figure 5): (1) the title of the lesson learned, (2) a reference to a step of the GQM process, (3) the element of the process step that is addressed in the lessons learned (e.g., one of the input documents or a role involved in the process step), (4) the addressed aspect or problem with the process element (e.g., size of the input document exceeds a certain threshold or a needed resource is available only part-time), (5) the context situation in which the lesson learned has occurred and is supposed to be valid (e.g., in a large multi-site project in which a measurement program is run for the first time), (6) the solution suggested to the problem in that situation (e.g., a rule for how to split the input document or a suggested software tool for supporting a part-time resource), and at last (7) notes and comments that do not fit into one of the other facets.

Copyright © Fraunhofer IESE 1998

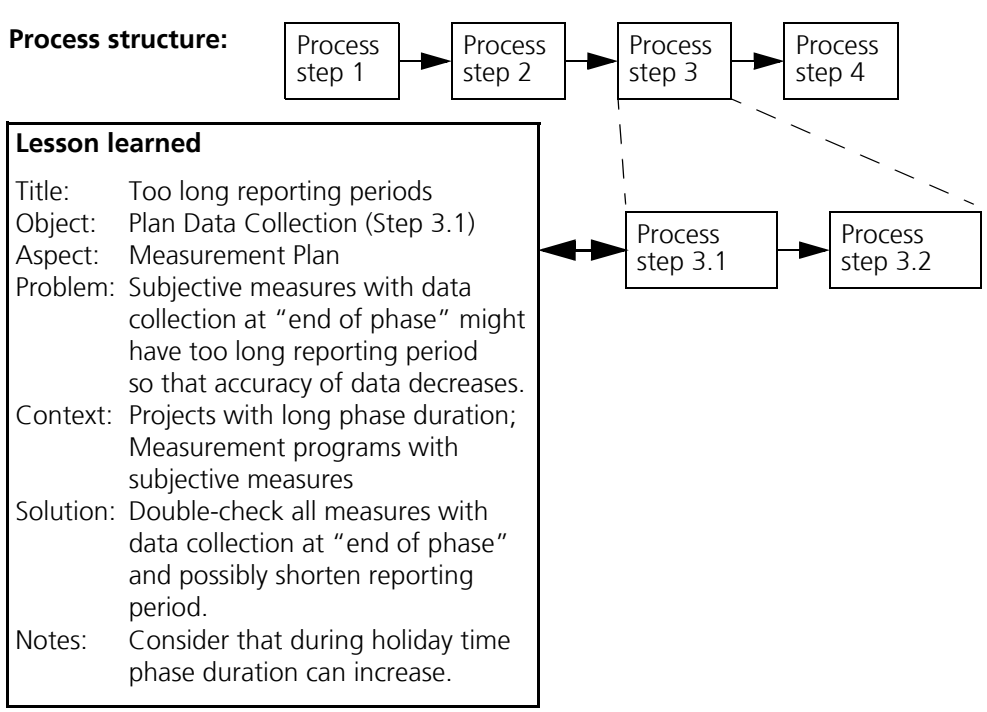


Figure 5:Structure of lessons learned for guiding the GQM process

The lessons learned on GQM-based measurement programs are integrated with a definition of the GQM process. Each lesson learned is attached to the process step to which it refers. Each process step is described in textual and graphical form as a section of a process handbook. It contains parts such as *objective*, *required resources*, *input* and *output products*, *entry* and *exit criteria*, *sub-processes*, and *activity description*. The lessons learned are added as an additional part called *guidelines*.

The handbook is to be used mainly for three purposes: Teaching and training of GQM, priming of personnel before the start of a measurement program, and as reference book.

4.4 Approach to Representing and Using Lessons Learned

In this section we will detail the logical structure introduced in Subsection 3.4.1 and show how the refined structure can be used to support packaging and automate retrieving relevant lessons learned.

The previous sections imply that there are several types of knowledge entities (description of problem situations, problems, solutions, guidelines, etc.). However, these entities are not independent of each other. Rather, the entities form a complex network with semantic relationships. Figure 6 shows the semantic network we suggest for lessons learned. Such a semantic network is important

Copyright © Fraunhofer IESE 1998

for writing (e.g., ensuring a uniform structure) and using lessons learned (e.g., to aid retrieving). From the lesson learned form of application case 1 (cf. Figure 2) we learn that a lesson learned is made up of an issue, project identification, problem situation (= status), effects, (suggested) correction, and a guideline. Project identification and problem situation together constitute the context in which the lesson learned is (known to be) applicable. Therefore, they are merged into the concept "context". The "has-part" relationship shows what types of information have to be given if a new lesson learned is to be written. The other relationships derived from application case 1 (causes, solves, resolves, avoids) are used to reach the goals enumerated in Section 2.1. For instance, for choosing applicable guidelines, all lessons learned with similar projects in their context are retrieved. From the context the associated guidelines (via the "avoids" relationship) are recalled. If during the performance of a project a problem situation is diagnosed, the (suggested) correction can be recalled via the "resolves" relationship.

Application case 2 interprets the different types of knowledge (issue report, usage guideline, improvement suggestion, technology effect) as valuable information on their own (cf. Figure 3). This is represented by the "is-a" relationship.

Finally, application case 3 emphasizes that each lesson learned is about some kind of object. The object of a lesson learned is not restricted to a process step[1], i.e., an activity. This is expressed in Figure 6 by the concept "object of lesson learned". Other possible types of objects are products, resources, etc. In this sense, the semantic network can be further extended.

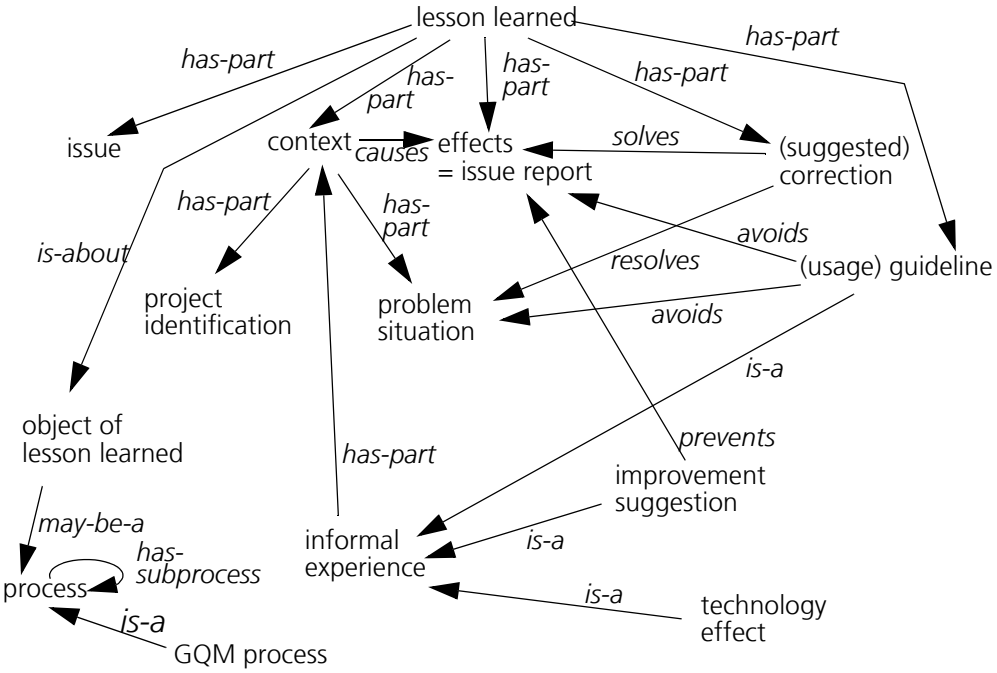
In order to find applicable[2] lessons learned intelligent retrieval is necessary because only seldom an actual problem situation matches exactly a problem situation of the past. For this purpose, characterization schemes may be used which are made up of facets and describe the properties of the knowledge types. For example, a process step can be described by its objective, required resources, input and output products, entry and exit criteria as well as by its activity description (cf. Section 4.3).

1 In application case 3 the object of the lesson learned is actually described by a pair: the corresponding process step and the element of the process step associated with the lesson learned (e.g., a resource or description of the activity).

2 A lesson learned is "applicable" if its contents are useful to solve the problem at hand. Usually this usefulness is modeled by the similarity between the context of the lesson learned and the context of the problem at hand. This means that both the characteristics (e.g., team size) and the status of the projects (e.g., design finished) must approximately be the same.

Copyright © Fraunhofer IESE 1998

Figure 6:Semantic network of types of knowledge entities for lessons learned



Based on such characterization schemes the retrieval of applicable lessons learned can be automated using similarity functions which compute the similarity between the characterizations of the lessons learned in the repository and the characterization of the wanted lesson learned [Prie91, OHPB92, TA97].

Copyright © Fraunhofer IESE 1998

5 Storing and Disseminating Lessons Learned

Tightly coupled with the representation of lessons learned is the storage of lessons learned. The employed technical infrastructure used for storing determines to a large degree the way lessons learned are to be disseminated within an organization:

- If no tool support is available, lessons learned may be collected in binders and flipped through on demand (cf. application case 1). This requires that new lessons learned are copied and distributed to project managers which are expected to use them.

- An alternative is to write a report which describes both some object (e.g., the GQM process) and the lessons learned related to the object (cf. application case 3). An improvement is to have this report on-line because then the update/distribute cycle can be shortened.

- At the end of the spectrum lies specialized tool support which allows access over a net (e.g., WWW-based) and takes advantage of a knowledge representation like the one presented in the previous section. Dissemination is done automatically as new lessons learned become available when they are entered into the system.

The main requirements for automated support of lessons learned usage are as follows:

- Most of the core functionality is related to retrieval; for a given problem and context situation one or more existing solution cases need to be retrieved.

- Data processing is not really needed for the core functionality; it can become relevant for providing additional functionality.

- Problem solving is also not needed for the core functionality; it can become relevant for supporting the adaptation of solutions at a later phase of a lessons learned program.

- Relationships to various other kinds of objects (e.g., process models, project characterizations, narrative experience reports) need to be represented (as links).

- The similarity function expressing the applicability of a lessons learned needs to deal with arbitrary types of data, ranging from formal data to fully informal, narrative text.

Copyright © Fraunhofer IESE 1998

- Over time the amount of data will become very large. Therefore, the tool infrastructure must be scalable.

- A lessons learned management system needs to be integrated into possibly already established on-line information systems (e.g., WWW-based corporate intranets).

Hence, support for lessons learned usage needs to offer strong retrieval functionality, be able to integrate data processing and problem solving functionality, and allow for handling a wide variety of data types and similarity functions. We have evaluated several implementation technologies and concluded that commercially available case-based reasoning (CBR) systems provide the most appropriate platform for lessons learned management [GABT98, TA97]. CBR can be utilized to fulfil the requirement of finding applicable lessons learned using characterization schemes as they have been introduced in Section 4.4 [GRA+98, Sar95]. Therefore we plan to enhance the existing system described in Section 4.2 by a CBR component using CBR-Works by tecInno (Kaiserslautern, Germany) [Alt97, Tec98].

Figure 7:Architecture of a knowledge management system for lessons learned

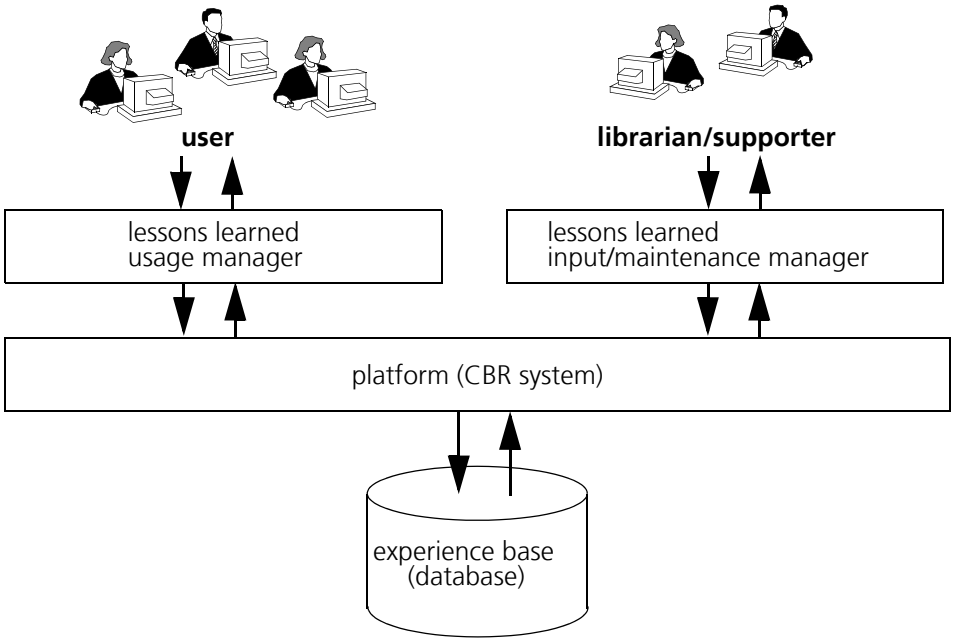


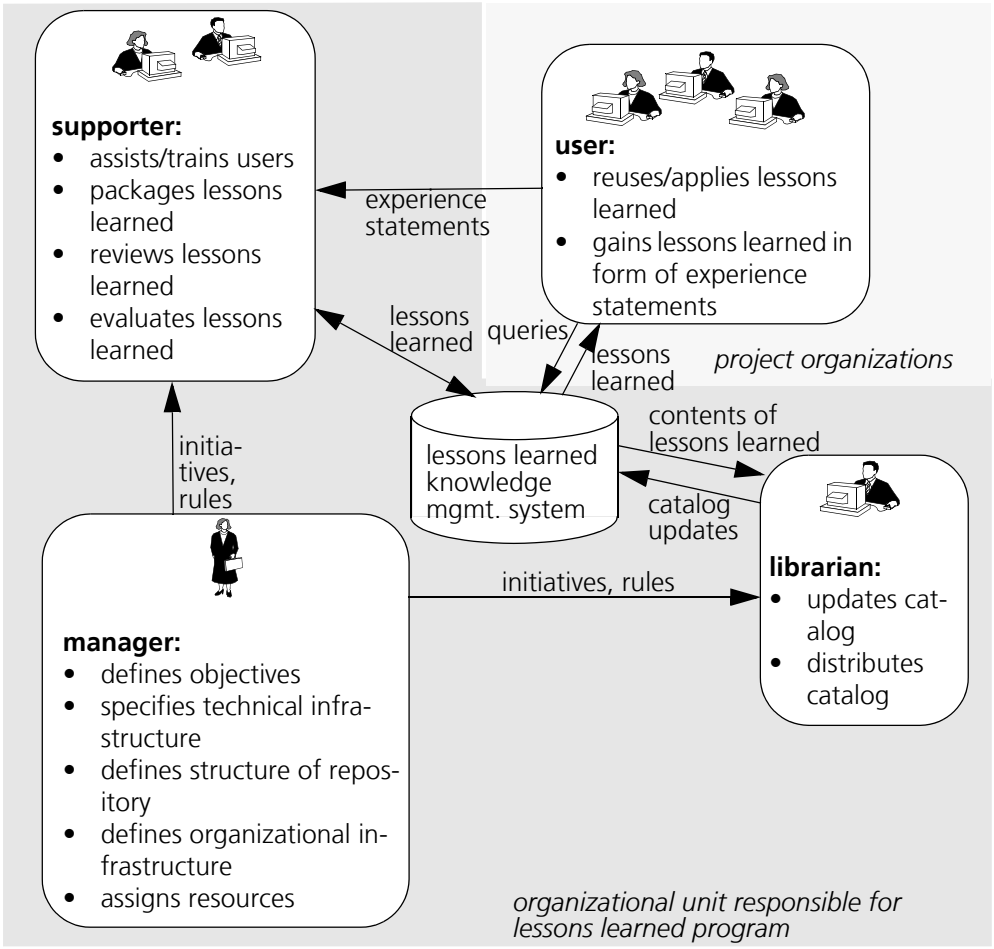
Figure 7 shows a suggested architecture of a knowledge management system for lessons learned. It uses a CBR system as platform technology that provides access to an underlying database of lessons learned and related information ("experience base"). For the two major task areas ("usage of lessons learned" and "input/maintenance of lessons learned") customized software packages are provided. The proposed architecture is capable of handling informal data and guarantees a smooth transition towards formal knowledge representation.

Copyright © Fraunhofer IESE 1998

6 Putting Management of Lessons Learned into Practice

If a lessons learned program is put into practice, it is not enough to know how to gain, package, represent, store, disseminate and use lessons learned. In addition, managerial aspects have to be considered. Management is responsible for setting up and running such a program. Figure 8 shows the general organizational infrastructure.

Figure 8:General organizational infrastructure for a lessons learned program



The initial decisions to be made when setting up a lessons learned program are all made by the manager and include:

- *Define objectives.* The objectives will determine who needs access to the lessons learned management system as well as how the lessons learned management system is to be used.

Copyright © Fraunhofer IESE 1998

- *Specify technical infrastructure*. The technical infrastructure involves the kind of storage system used (e.g., reports or specialized tools) and determines how the lessons learned are disseminated and how the users can access them.

- *Define structure of repository*. Depending on the objectives, certain types of knowledge entities have to be considered for storage. Figure 6 can be used as a starting point. The definition of the structure also includes the definition of characterization schemes as indicated in Figure 1 and pointed out in Section 4.4.

- *Define organizational infrastructure*. The organizational infrastructure requires the establishment of a logical and/or physical organizational unit which is dedicated to the lessons learned program, i.e., a unit which packages lessons learned and supplies them on demand to projects. This includes technical aspects like updating and distributing the catalog of lessons learned as well as consulting and quality assurance functions such as assisting and training users, helping in gaining of lessons learned, packaging, reviewing and evaluating lessons learned. Part of the definition of the organizational infrastructure is also the definition of rules and processes on how to gain, package, evaluate, store, and use lessons learned.

- *Assign resources*. Finally, people have to be assigned to the above mentioned functions. Note that Figure 8 shows only (logical) roles. At the beginning of a lessons learned program a single person may be responsible for all functions (manager, supporter, librarian). As the program grows more and more people may be added to the new organizational unit.

Once a lessons learned program is set up, the manager starts initiatives as well as improves rules and processes on how to gain, package, evaluate, store, and use lessons learned. New initiatives and rules are typically based on lessons learned about the lessons learned program. Here are some of the lessons we have learned (some of which have already been addressed in the previous sections):

- Lessons learned are not a by-product of project work. Rather dedicated effort is necessary to make informal statements reusable for future projects (cf. Section 3).

- The dedicated effort has to be carried out by people who are assigned to the lessons learned program (at least part-time), i.e., an organizational unit responsible for the maintenance of the lessons learned repository has to be established. Otherwise, no one will feel responsible for gaining and packaging lessons learned. Moreover, a small maintenance group ensures that lessons learned are written in a consistent manner.

Copyright © Fraunhofer IESE 1998

- Users need to be willing in supplying their experience in form of informal statements, project decisions, reports, etc. If people are not willing, they will only give irrelevant experience. This means that the organization must provide an environment in which people are happy to share their experience with others. This can be done, e.g., via incentives. In this case the lessons learned program becomes an improvement suggestion program for software development.

- Lessons learned management systems can be used as a communication means. Even if only problem situations and their effects are known (but no solution for them), this information can be entered into the lessons learned management systems, e.g., as issue reports (see Section 4.2) together with the "experience provider". Other people experiencing the same difficulties can now contact the original "experience provider" making up a team which might be able to find satisfactory solutions using synergy effects.

Copyright © Fraunhofer IESE 1998

7 Related Research

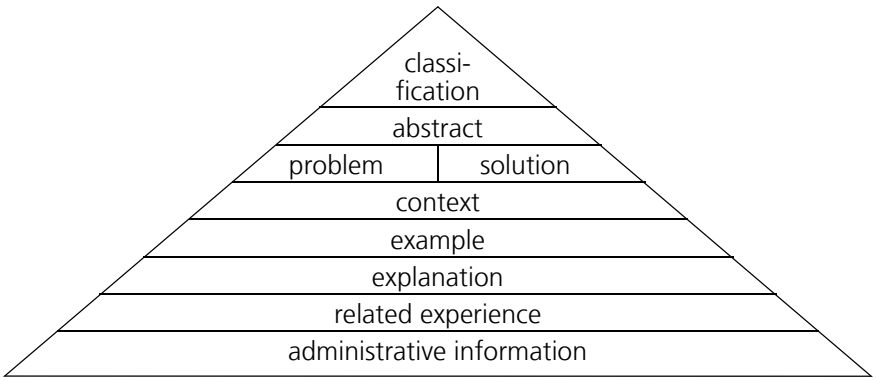
While this report tries to give a comprehensive view of all aspects of lessons learned management, other approaches focus on particular aspects for specific purposes. For example, Gresse von Wangenheim et al. [GRA+98] present a detailed case-based representation for lessons learned about goal-oriented measurement programs together with specific usage scenarios. In their work the solution part of a lesson learned contains a justification which provides an explicit rationale for the selection of the solution. Such a justification is a first step towards supporting the adaptation of a solution if a retrieved lesson learned is about a similar problem. In addition, an outcome assessment states explicitly if the problem was successfully solved by the solution or failed. If the solution did not solve the problem, a failure explanation is given. This is the result of the more formal representation used by Gresse von Wangenheim et al. Using our representation a failure would also be recorded as a solution. However, in this case the solution would not be constructive, but rather prohibitive, e.g., "if you do not understand the requirements completely (problem), then *do not* start design (solution)".

For the lessons learned program at NASA's Goddard Space Flight Center [Sar95] specific guidelines for formulating lessons learned have been developed. Among the criteria to decide whether to store a lesson learned in a central repository are its relevance, comprehensibility, utility, validity, and realizability. A detailed description for filling out lesson learned forms is also given. Using these guidelines new lessons learned are reviewed. A prototype based on case-based reasoning technology allows storage and retrieval of the lessons learned.

Houdek and Kempter present in their work the quality patterns approach [HK97]. They show how the quantitative results of a goal-oriented measurement program can be packaged in so called quality patterns. A quality pattern has a problem-solution pair at its core which is enhanced by further information giving it its pyramidal form (see Figure 9). This way a potential user does not have to read the complete lesson learned to decide whether it is applicable. The explanation gives (similar to the justification of [GRA+98]) an explicit rationale for the stated solution. An example and references to related experiences provide more detailed information if the reader needs more background information to understand the problem-solution description. This example shows that an explicit structure for lessons learned (such as the one presented in Section 4.4) is important.

Copyright © Fraunhofer IESE 1998

Figure 9:Structure used for quality patterns



A more informal approach to lessons learned is presented by Kleiner and Roth [KR97]. Their approach is based on the observation that people in organizations act collectively, but learn individually. Organizational learning, however, requires that individual perspectives are merged and discussed within the organization. Therefore, people are interviewed for their opinions why certain projects failed or were better than expected. These opinions are carefully screened and assembled into a learning history (a special type of report) which can be read like a short story. The learning history is copied and disseminated to managers. The approach has been used for evaluating project management issues. It has not been developed explicitly for software development. Thus, it is unclear whether it can be applied for technical problems in software development. Nevertheless it shows that lessons learned can be handled informally and still be perceived as being valuable. This underlines that it makes sense to start with a small technical infrastructure. However, learning histories have to be read completely to find out whether they are applicable. If lessons learned are to be used as an aid to solve everyday problems, a finer granularity of lessons learned is needed. This can lead to large numbers of lessons learned requiring more technical infrastructure as we have proposed in this report.

In the previous section we gave an example how the organization can benefit from increased communication. While most lessons learned programs aim at preventing the repetition of past failures and promote the recurrence of good practices, only a few state the objective to improve communication explicitly, e.g., "discuss old ways of thinking that led to mistakes" [KR97] and "increase communication and teamwork" [Sar95].

Other approaches extend the scope even further by aiming at the provision of all information relevant to local development practices that cannot be learned in school [Hen97]. Output of Henninger's approach are not only lessons learned, but also custom languages, organization and project-specific programming conventions, policies and guidelines concerning tool usage. All of these are accessible via a system named BORE. In this approach lessons learned are perceived as an important part of an organization-wide repository showing that the lessons

Copyright © Fraunhofer IESE 1998

learned management can actually act as catalyst for establishing comprehensive knowledge management systems in software organizations (as mentioned in the introduction).

Copyright © Fraunhofer IESE 1998

8 Summary and Outlook

An integrated method for lessons learned management has been presented that takes arbitrary experience statements from software projects as input. It makes them reusable and establishes a comprehensive reuse infrastructure for lessons learned. A tool architecture has been suggested for implementing the reuse infrastructure.

The method for lessons learned management involves four major parts: (1) a process model for packaging lessons learned and procedures for gaining, storing, and using them; (2) a knowledge representation for lessons learned and guidelines for tailoring it to specific application contexts; (3) a tool architecture of a knowledge management system for lessons learned management; (4) recommendations for setting up lessons learned programs.

The method has been derived from three application cases. They already provided a first evaluation of the approach. For instance, the packaging process has been evaluated in the inspections and requirements engineering cases (Cases 1 and 2). An integrated storage has been established in the GQM case (Case 3). Case 2 has provided a prototype implementation of the tool architecture's user modules on top of an object-oriented modeling framework. Usage-centered design of an adequate representation has also been elaborated in Case 2.

Future developments will concentrate on providing tool support and implementation of the developed experience bases. Currently, a group at Fraunhofer IESE is about to develop the first version of the knowledge-based system for experience management using and tailoring a commercial case-based reasoning system. The experiences from the lessons learned programs have contributed a significant set of requirements for that work. In ESPRIT project PROFES, additional GQM experiences are currently collected. They will be provided on-line and integrated later into the experience base.

Copyright © Fraunhofer IESE 1998

9 Acknowledgments

The authors would like to thank Stefan Vorwieger who gave us many valuable insights into the special research project for application case 2. Without them the lessons learned program for the requirements engineering application could not have been set up. Also, we would like to express our gratitude to Klaus-Dieter Althoff, Erik Kamsties, and Dietmar Pfahl for reviewing an earlier version of this report. Part of this work has been conducted in context of ESPRIT projects PERFECT (no. 9090) and PROFES (no. 23239) with support of the CEC.

Copyright © Fraunhofer IESE 1998

10 References

[Alt97] Klaus-Dieter Althoff. Evaluating case-based reasoning systems: The Inreca case study. Postdoctoral thesis (Habilitationsschrift), University of Kaiserslautern, July 1997.

[AP94] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AICom - Artificial Intelligence Communications*, 7(1):39–59, March 1994.

[BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. Goal Question Metric Paradigm. In John J. Marciniak, editor, *Encyclopedia of Software Engineering*, volume 1, pages 528–532. John Wiley & Sons, 1994.

[BDR97] Lionel C. Briand, Christiane Differding, and H. Dieter Rombach. Practical guidelines for measurement-based process improvement. *Software Process Improvement and Practice Journal*, 2(3), 1997.

[CP93] P.-J. Courtois and D. L. Parnas. Documentation for safety critical software. In *Proceedings of the Fifteenth International Conference on Software Engineering*, pages 315–323. IEEE Computer Society Press, May 1993.

[GABT98] C. Gresse von Wangenheim, K.-D. Althoff, R. M. Barcia, and C. Tautz. Evaluation of technologies for packaging and reuse of software engineering experiences. To be published, 1998.

[GG93] Tom Gilb and Dorothy Graham. *Software Inspection*. Addison-Wesley Publishing Company, 1993.

[GHW95] Christiane Gresse, Barbara Hoisl, and Jürgen Wüst. A process model for GQM-based measurement. Technical Report STTI-95-04-E, Software Technologie Transfer Initiative Kaiserslautern, Fachbereich Informatik, Universität Kaiserslautern, D-67653 Kaiserslautern, 1995.

[GRA+98] Christiane Gresse von Wangenheim, Alexandre Moraes Ramos, Klaus-Dieter Althoff, Ricardo M. Barcia, Rosina Weber, and Alejandro Martins. Case-based reasoning approach to reuse of experiential knowledge in software measurement programs. In Lothar Gierl, editor, *Proceedings of the 6th German Workshop on Case-Based Reasoning*, Berlin, Germany, 1998.

Copyright © Fraunhofer IESE 1998

[Hen97] Scott Henninger. Capturing and formalizing best practices in a software development organization. In *Proceedings of the 9th International Conference on Software Engineering & Knowledge Engineering*, pages 24–31, Madrid, Spain, June 1997.

[HK97] Frank Houdek and Hubert Kempter. Quality patterns – an approach to packaging software engineering experience. *Software Engineering Notes*, 22(3):81–88, May 1997.

[KR97] Art Kleiner and George Roth. How to make experience your company's best teacher. *Harvard Business Review*, 75(5):172–177, September/October 1997.

[OHPB92] Eduardo Ostertag, James Hendler, Rubén Prieto-Díaz, and Christine Braun. Computing similarity in a reuse library system: An AI-based approach. *ACM Transactions on Software Engineering and Methodology*, 1(3):205–228, July 1992.

[Prie91] Rubén Prieto-Díaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):89–97, May 1991.

[Sar95] Charisse Sary. Recall prototype lessons learned writing guide. Technical Report 504-SET-95/003, NASA Goddard Space Flight Center, Greenbelt, Maryland, USA, December 1995.

[TA97] Carsten Tautz and Klaus-Dieter Althoff. Using case-based reasoning for reusing software knowledge. In D. Leake and E. Plaza, editors, *Proceedings of the Second International Conference on Case-Based Reasoning Research and Development, (ICCBR97)*. Springer Verlag, 1997.

[Tec98] CBR-Works. URL http://www.tecinno.de/tecinno_e/ecbrwork.htm, 1998. tecInno GmbH, Germany.

[vS93] A. John van Schouwen. The a-7 requirements model: Re-examination for real-time systems and an application to monitoring systems. CRL Report No. 242, McMaster University, CRL, Hamilton, Ontario, Canada, February 1993.

Copyright © Fraunhofer IESE 1998

Copyright © Fraunhofer IESE 1998

Document Information

Title:	Knowledge Management of Software Engineering Lessons Learned
Date:	July 1998
Report:	IESE-002.98/E
Status:	Final
Distribution:	Public

Copyright 1998, Fraunhofer IESE.
All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means including, without limitation, photocopying, recording, or otherwise, without the prior written permission of the publisher. Written permission is not needed if this publication is distributed for non-commercial purposes.