# The Quality of Knowledge: Knowledge Patterns and Knowledge Refactorings

**Jörg Rech\*, Björn Decker, Eric Ras, Andreas Jedlitschka**
Fraunhofer Institute for Experimental Software Engineering,
Kaiserslautern, Rhineland-Palatine, Germany
E-mail: (joerg.rech, bjoern.decker, eric.ras, jedl)@iese.fraunhofer.de
\*Corresponding author

**Raimund L. Feldmann**
Fraunhofer USA, Center for Experimental Software Engineering, Maryland
4321 Hartwick Road, Suite 500, College Park, MD 20742-3290, USA
E-mail: Rfeldmann@fc-md.umd.edu

**Abstract:** Knowledge management is a relatively young discipline. Nevertheless, it has accumulated a valuable body-of-knowledge in the structuring of knowledge and in the design of socio-technical knowledge management systems. However, concepts to describe common, recurring patterns of how to describe, structure, interrelate, group, or manage knowledge elements are still missing. In this paper, we introduce the concepts "knowledge pattern" and "knowledge anti-pattern" to describe best and worst practices in knowledge management, "knowledge refactoring" to improve or change knowledge anti-patterns, and "quality of knowledge" to describe desirable characteristics of knowledge in knowledge management systems. The concepts are transferred from software engineering to the field of knowledge management based on our experience from several knowledge management projects.

**Keywords:** knowledge management best practices, knowledge patterns, knowledge management patterns, knowledge refactoring, knowledge quality, quality of knowledge

## 1 INTRODUCTION

Today, knowledge management (KM) as well as learning management (LM) consist of a multitude of models, theories, and systems comprised of valuable and recurring knowledge that waits to be reused in new KM systems. However, the quality of the knowledge gained, the technical KM system used, or the social KM method applied is neither easy to be evaluated, nor is it easy to be improved. Many best practices in form of success factors (Mathi, 2004) (Thomas, 2006) (Morisio, Ezran, & Tully, 2002), success models (Jennex & Olfman, 2004, , 2006), success measures (Jen & Yu, 2006), reference architectures for KM systems (Davenport & Probst, 2000; Mertins, 2003), or worst practices (Fahey & Prusak, 1998) are known in KM and typically preserve knowledge about the whole KM systems or initiative. But commonly accepted best practices on how to structure knowledge, how to design an interface for a KM system, or how to start a storytelling session can hardly be found and concepts to describe common, recurring patterns of how to describe, structure, interrelate, group, or manage knowledge are still missing.

During the mid-1990s the concept "design pattern" was developed in software engineering to describe best practices regarding the design of software systems in a structured way. Design patterns are used to represent knowledge that is based on experiences captured in several real-world projects and which is widely accepted. This semi-formal representation is often used for describing and presenting the gained knowledge.

In this paper, we transfer the concepts quality, patterns, and refactoring from software engineering to the field of KM and introduce the concepts of *knowledge patterns* and *knowledge refactorings* in the context of *knowledge*

*quality*. We describe an approach to structure knowledge in knowledge management systems in the form of so called *knowledge patterns*. These patterns and anti-patterns can be used to develop KM systems and improve the quality of the systems themselves as well as that of the knowledge within (i.e., the *quality of knowledge*). Furthermore, we transfer the concepts of software refactoring and software quality to describe the effect of knowledge patterns as well as countermeasures (i.e., *knowledge refactorings*) to remove knowledge anti-patterns. To illustrate the concept of knowledge patterns, we provide examples that are based on our observations from developing and operating several knowledge management (KM) systems (i.e., they do not represent empirically validated findings). Knowledge patterns state lessons learned and best practices for the structuring of knowledge, the design of KM systems, and the development of underlying ontologies. They should be kept in mind when building high-quality knowledge and KM systems. Furthermore, patterns in KM represent a way of structuring knowledge as well as a form of language that helps knowledge engineers to communicate about knowledge and KM systems. With this paper we also want to stimulate the discussion about the meaning of quality in the context of KM, how knowledge should or should not be described in a KM system, and what is needed to generate a fruitful socio-technical KM system.

Relevant background information concerning KM, best practices in KM, software engineering, and software patterns are presented in the next section. Section 3 describes several desirable quality aspects of knowledge in KM systems that are affected by patterns. The core of this paper – the knowledge patterns and anti-patterns – are described in section 4, followed by a section about how these patterns might be implemented (c.f. section 5). Finally, we conclude and give an outlook of future work in section 6.


## 2   BACKGROUND

The relevant background for knowledge patterns is comprised of knowledge and learning management, software engineering and reuse, KM in software engineering, as well as patterns in software engineering. The following sections will focus on these fields and their relations to patterns in general.

### 2.1   Knowledge and Learning Management
KM and learning management (LM) both serve the same purpose: facilitating learning and competence development of individuals, in projects, and in organizations – but, they follow two different perspectives. KM is related to an organizational perspective, because it addresses the lack of sharing knowledge among members of the organizations by encouraging individuals to make their knowledge explicit by creating knowledge elements, which can be stored in knowledge bases for later reuse or for participating in communities of practice. Learning management emphasizes an individual perspective, as it focuses on the individual acquisition of new knowledge and the socio-technical means to support this internalization process. The high potential for synergies between Knowledge- and Learning Management seems obvious given the many interrelations and dependencies of these two fields. An interview-based study demonstrated that perceived connections between both are not operationalized (Efimova & Swaak, 2002). Even a few years later, many barriers regarding their integration still exist (Ras, Avram, Waterson, & Weibelzahl, 2005).

Knowledge as the fourth factor of production (Senge, 1995) is one of the most important assets for any kind of organization, and for all areas of science. While *experiences* describe events in one specific context that can only be reused carefully, *knowledge* is usually applicable in previously unknown contexts with a fair amount of certainty. Unfortunately, a small number of experts who have acquired knowledge through their experiences in day-to-day work hold major parts of the knowledge in an organization. Surprisingly, this is equally true for researchers in KM. Experience gained about knowledge itself and KM systems, either technical, social, or socio-technical ones, are typically recorded in the form of models or process models only. Fine-grained knowledge about the structuring, interconnection, or classification of knowledge is rarely documented, and common and recurring patterns are hardly available – while best practices about the whole KM systems and initiatives are often shared (Davenport & Probst, 2000; Mertins, 2003).

While the concept of knowledge patterns is still new in KM, there are similar concepts such as success factors (Mathi, 2004) (Thomas, 2006) (Morisio, Ezran, & Tully, 2002), success models (Jennex & Olfman, 2004, , 2006), success measures (Jen & Yu, 2006), reference architectures for KM systems (Davenport & Probst, 2000; Mertins, 2003), worst practices (Fahey & Prusak, 1998), barriers (Eberle, 2003), facilitators (Damodaran & Olphert, 2000), and incentives (Feurstein, Natter, Mild, & Taudes, 2001) that are often described in an unstructured and informal way. Barriers, facilitators, or incentives represent types of patterns that describe common and recurring incidents,

practices, or behavioural structures in KM. There are many different types of barriers such as knowledge barriers in general (Riege, 2005), barriers in knowledge transfer (Sun & Scott, 2005) and distribution (Bick, Hanke, & Adelsberger, 2003), barriers based on culture (Lippert et al., 2003), as well as barriers based on roles and activities (Awazu, 2004). Nevertheless, a concept for documenting commonly recurring patterns on how to describe, structure, interrelate, group, or manage knowledge components is still missing.

The expectations on Learning Management and e-Learning content in particular are high (cf. SCORM 2004 2nd Edition Overview page 1-22, http://www.adlnet.org/scorm/index.cfm): Systems should provide access to instructional components from diverse locations, instruction should be adaptable to individuals and organizational needs, delivering instruction must be affordable, and the system should address the criteria durability, interoperability, and reusability. Only best practices exist related to the development of learning management systems and learning content since they strongly depend on the learning context of the individual, group, or organization who want to learn – no general patterns or antipatterns are available. Nevertheless, numerous initiatives like AICC (the Aviation Industry CBT Committee), ADL (Advanced Distributed Learning), IEEE LTSC (the Learning Technology Standards Committee of the IEEE) and IMS Global Learning Consortium have made efforts to establish standards as a first step to guide the development of patterns. For several years, a number of initiatives have agreed to cooperate in the field of standards and specifications. Several of these specifications have been incorporated and in some cases been adapted by ADL to define the SCORM reference model. SCORM describes that technical framework by providing a harmonized set of guidelines, specifications, and standards based on the work of several distinct e-Learning specifications and standards bodies. These specifications have one aspect in common: by separating the content from the structure and layout, they enable the author to develop different variants of learning material very efficiently, while relying on the same set of learning objects.

Many commercial as well as open source-based learning management systems have implemented the concepts provided by these specifications and standards in order to fulfil the requirements listed previously. The concepts are still not available as a comprehensive set of patterns. Nevertheless, the development of patterns and antipatterns for knowledge management should also refer to learning management concepts, since they provide relevant aspects about content structuring, linking, and navigation. Knowledge transfer – as a phase of KM – is related to learning and competence development, and therefore its success depends on knowledge structures that stimulate learning processes.

## 2.2 Software Engineering

The discipline of *Software Engineering (SE)* was born in 1968 at the NATO conference in Garmisch-Partenkirchen, Germany (Simons, Parmee, & Coward, 2003), where the term "software crisis" was coined to describe the increasing lack of quality in software systems that were continuously growing in size. Today, quality is still of utmost importance in the development of software products.

At the same conference, the systematic reuse of software components was motivated by Dough McIllroy (McIllroy, 1968) to improve the quality of large software systems by reusing small, high-quality components. The reuse of existing knowledge and experience is one of the fundamental parts in many sciences. Engineers often use existing components and apply established processes to construct complex systems. Without the reuse of well proven components, methods, or tools, we would have to rebuild and relearn them again and again.

In software reuse, several named barriers were described by Judicibus and classified into the two classes, "individual factors" and "collective factors" (Judicibus, 1996). Individual factors are:

- *Artist's Syndrome*: Developers consider themselves more like artists than like engineers; they want to build something "beautiful" and avoid the reuse of external and "ugly" software. Typically, developers would more likely develop a function from scratch than reuse an existing component that does not fulfill the given requirements 100%.
- *Standards' Phobia*: Standards are needed in software reuse to build upon standardized components. But developers have to build components based on the project requirements and the reuse standard and often neglect the additional effort.
- *Egghead's Syndrome*: Developers and esp. experts do not want to share their expertise and abandon their power. Building reusable components would enable other, less experienced developers to reuse this knowledge.
- *Feudal Lord's Syndrome*: Typically, managers think and are judged by the numbers – the more personnel or budget the more important a manager has to be. Building reusable components will mostly generate a benefit for other departments and relatively decrease one's own status. Furthermore, reuse would lead to smaller teams, cheaper projects, and therefore fewer personnel and a lower budget.

In contrast to these individual factors, the collective factors group together cultural and social barriers:

- *Not Invented Here Syndrome*: Companies or departments often see the products of others as inferior to what they themselves have or could create. The motivation to (re-)use them is non-existent to negative.
- *The Technology Syndrome*: The first impression of a new technology often decides about how it will be seen in the company. New technologies typically need time to be tailored and understood in order to be efficient and effective.
- *The Revenue Mania*: Departments are often judged solely by their income. The more revenue a department makes right now, the better. Building reusable components for future usage is often not recognized as a long-term investment and only decreases the short-term income.

Another social barrier was described in (Favaro, 1991): New approaches and technologies like software reuse are often introduced with high expectations that lead to an initial euphoria followed by disillusion. This barrier could be named *manic depression*.

### 2.2.1 Knowledge Management in Software Engineering

Since the NATO conference on software engineering, the two fields software reuse and Experience Management (EM) have increasingly been gaining importance. The roots of EM lie in Experimental Software Engineering ("Experience Factory"), in Artificial Intelligence ("Case-Based Reasoning"), and in KM. EM is comprised of the dimensions methodology, technical realization, organization, and management. It includes technologies, methods, and tools for identifying, collecting, documenting, packaging, storing, generalizing, reusing, adapting, and evaluating experience as well as for the development, improvement, and execution of all knowledge-related processes.

KM Systems in the area of Software Engineering (SE) deal with the processes and products as described in the Software Engineering Body of Knowledge (SWEBOK, 2004). They should be able to handle work products, reusable parts thereof, and organization-specific experience achieved with or through applying those processes.

KM systems for EM (esp. in SE) are usually instantiations of the Experience Factory (EF) concept. The EF is an infrastructure designed to support EM (i.e., the reuse of products, processes, and experiences from projects) in software organizations (Basili, Caldiera, & Rombach, 1994). It supports the collection, pre-processing, and dissemination of experiences and represents the physical or at least logical separation of the project and experience organization as shown in Figure 1. This separation is meant to relieve the project teams of the burden to find, adapt, and reuse knowledge from previous projects as well as to support them in collecting, analyzing, and packaging valuable new experiences that might be reused in later projects.

Typically, such experience and/or (external) knowledge is stored and documented in the form of FAQs, lessons learned, war stories, or identified best practices (e.g., (Harrison, 2004), (Nicholls, 2004)). Examples for possible products to be stored in a KM system for SE are requirements, designs, patterns, source code, test cases, or other documentation about products and processes. Additionally, knowledge about the development and measurement of products and processes is stored in these systems. In (Briand, Basili, Yong, & Squier, 1994), three kinds of documentations were considered for software maintenance projects:

- *product related*, describing the system itself (i.e., software requirement specification, software design specification, and software product specification);
- *process related*, used to conduct software development and maintenance (i.e., software development plan, quality assurance plan, test plan and configuration management plan);
- *support related*, helping to operate the system (i.e., user manual, operator manual, software maintenance manual, firmware support manual).

In the "knowledge dust to pearls" approach (Basili et al., 2001), the experiences gained in day-to-day work (i.e., the knowledge dust) is "analyzed, synthesized, and transformed into knowledge pearls, which represent more sophisticated, refined, and valuable knowledge items that take a longer time to produce". Similar to this approach, in the context of quality improvement, the raw defect data (i.e., experiences) found by testing, usage, or maintenance is collected, and if enough data is available and a pattern is apparent, it is generalized into a general pattern or anti-pattern description.

Almost all KM system implementations in SE make use of metadata (i.e., data about data) for describing and capturing the content (e.g., (Prieto-Díaz & Freeman, 1987), (Prieto-Díaz, 1993), (Lindvall, Frey, Costa, & Tesoriero, 2001)). Probably the most widespread and best known example for commonly used metadata are the file system attributes. For each file, these attributes provide additional information such as creation data, date of

last modification, or owner information. Such metadata can simply be described as attribute-value pairs, for example, (creation_date, 9/25/2005), or (owner, system admin).
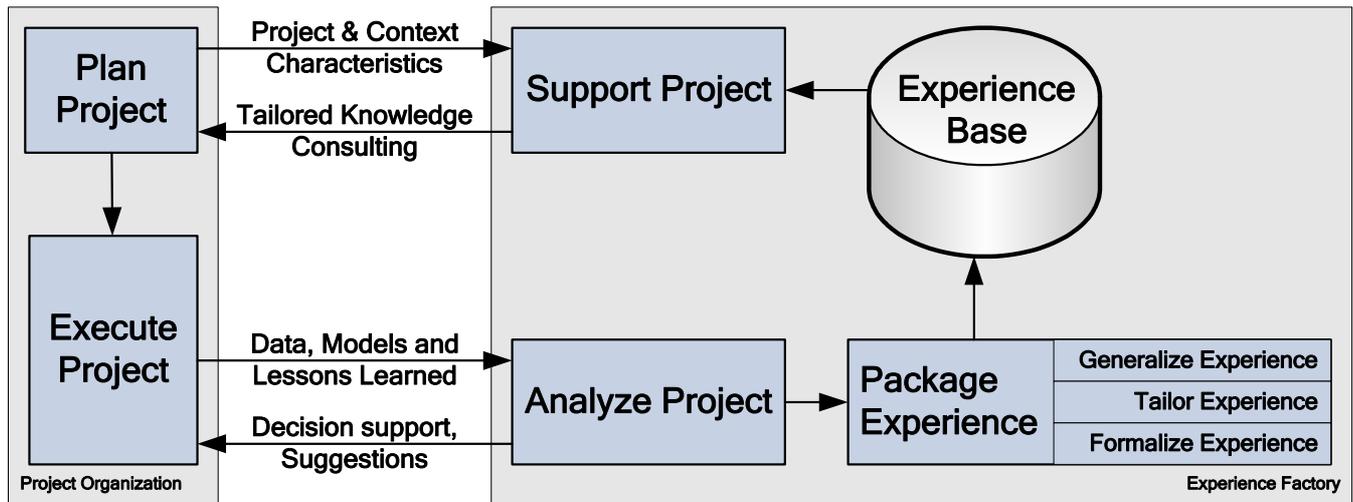


**Fig. 1** The Experience Factory (EF)

One of the first and probably best documented KM system in SE is the EF at NASA Software Engineering Laboratory (SEL) (Basili & Rombach, 1988), (Rombach & Ulery, 1989), (McGarry & Pajerski, 1990), (Basili, Caldiera, & Cantone, 1992), (Basili et al., 1995). Since then, many different implementations have been reported in North America (Basili, Lindvall, & Costa, 2001), (Kamel, M., & Sorenson, 2001), (Mendonca, Seaman, Basili, & Kim, 2001), as well as in Europe (Markkula, 1999), (Halvorsen & Nguyen, 1999), (Schneider & Schwinn, 2001), or Australia (Koennecker, Jeffery, & Low, 2000), (Scott & Jeffrey, 2003). One of the most recent initiatives in the US is the so–called OSD Acquisition Best Practices Clearinghouse (Dangle, Dwinnell, Hickok, & Turner, 2005), (BPC, 2005) which can be compared to the German VSEK (Feldmann & Pizka, 2003) or European ESERNET (Jedlitschka & Ciolkowski, 2004) systems.

### 2.2.2    KM Systems in Software Engineering

To illustrate the concept of knowledge patterns and refactorings in the following sections, we provide examples that are based on our observations from developing and operating several knowledge management (KM) systems. The EF was the underlying model for developing several experience and knowledge bases in projects such as RISE (Decker, Park, Quan, & Sauermann, 2005), ESERNET (Jedlitschka & Ciolkowski, 2004), SFB-EB (Feldmann, 1999), or VSEK (Feldmann & Pizka, 2003).

  The RISE (Reuse in Software Engineering) project was conducted during 2004 and 2005. With heavy industrial cooperation (about 50%), research focused on supporting reuse of SE knowledge by SMEs of the software industry (Decker, Park, Quan, & Sauermann, 2005). RISE aimed at integrating lightweight experience management with agile software development. The objectives targeted by RISE were to improve the communication between employees, to strengthen and accelerate the transfer of knowledge via a socio-technical system, to improve the retrieval of knowledge and orientation in a body of knowledge, and to optimize the amount and accelerate the time to access relevant knowledge. A further objective was to improve the quality of knowledge by assisting software engineers in creating optimized artifacts (i.e., with optimized content and structure) based on didactical principles.

  ESERNET (Jedlitschka & Ciolkowski, 2004) was a thematic network project conducted between 2001 and 2003 as part of the European Union's 5th Framework Programme under contract number IST-2000-28754 (cf. http://www.esernet.org). It had the objective to gradually change the mentality of software engineers and their organizations towards systematic empirical studies, for the purpose of long-term learning. The overall goal was to collect, systematize, and disseminate relevant and valid insights by building a SE knowledge base for several European countries with different cultural backgrounds. Knowledge collected in the project serves as a empirically validated base for assessing, understanding, changing, innovating, and using software technologies. The task of collecting this knowledge required a joint effort between academia, technology providers, software developers, and possible endusers.

In the context of the SFB 501 project, a long-term strategic research activity of the DFG (German Research Foundation), we created an experience base for software artefacts (Feldmann, 1999) to support the reliable and low-cost customization of complex domain-specific software systems. The PLEASERS (Product Line Approach for SE Repository Schemata) library with building blocks for the development of EM schemata (i.e., patterns of knowledge base structures) is based on this project (Feldmann & Carbon, 2003).

Germany's VSEK portal (previously known as ViSEK, see http://www.vsek.de) is a portal implemented to offer up-to-date SE knowledge in order to support SMEs in their daily work (Feldmann & Pizka, 2003). The German Federal Ministry of Education and Research (BMBF) funded the ViSEK project based on the idea that experience gained from research and practice should be packaged and easily made available to all companies. Therefore, an on-line SE repository was developed and installed during the project, which offers access to up-to-date software engineering technologies of selected application domains.

### 2.3 Patterns in Software Engineering

In the 1990s a new concept was transferred from architecture to computer science that helped to represent typical and reoccurring patterns of good and bad software architectures. These design patterns (Gamma, Richard, Johnson, & Vlissides, 1994) and anti-patterns (Brown, Malveau, McCormick, & Mowbray, 1998) were the start of the description of many patterns in diverse software phases and products. Today, we have thousands of patterns (Rising, 2000) for topics such as software reuse (Long, 2001), agile software projects (Roock & Havenstein, 2002) or pedagogies (http://www.pedagogicalpatterns.org/) (Fincher & Utting, 2002; Monteiro, Almeida, Goulao, Abreu, & Sousa, 1999). Many other patterns are stored in pattern repositories such as the Portland pattern repository (PPR, 2005) or the Hillside pattern library (HPL, 2005) and are continuously expanded by conferences such as PLOP (Pattern Languages of Programming; see http://hillside.net/conferences/).

While there are similar concepts in KM and software reuse such as barriers (Riege, 2005; Sun & Scott, 2005) and incentives (Ravindran & Sarkar, 2000), (Judicibus, 1996), the idea of patterns seems to be underdeveloped in KM.

However, we found the concept of patterns and anti-patterns helpful for documenting our knowledge and the experience we gained with the projects mentioned in Section 2.2.2. Our KM patterns are based on the following definitions as used for SE design patterns:

**Def: Design pattern**: A design pattern is a general, proven, and beneficial solution to a common, reoccurring problem in software design. Built upon similar experiences, design patterns represent "best-practices" about how to structure or build a software architecture. An example is the façade pattern, which recommends encapsulating a complex subsystem and only allows the connection via a single interface (or "façade") class. This enables the easy exchange and modification of the subsystem.

While patterns typically state and emphasize a single solution to multiple problems, anti-patterns typically state and emphasize a single problem to multiple solutions. According to (Brown, Malveau, McCormick, & Mowbray, 1998), anti-patterns are defined as:

**Def: Anti-pattern**: An anti-pattern is a general, proven, and non-beneficial problem (i.e., bad solution) in a software product or process. It strongly classifies the problem that exhibits negative consequences and provides a solution. Built upon similar experiences, anti-patterns represent "worst-practices" about how to structure or build a software architecture. An example is the "lava flow" anti-pattern that warns about developing a software system without stopping sometimes and reengineering the system. The larger and older such a software system gets, the more dead code and solidified (bad) decisions it carries along.

In the following sections, we will set the stage for our KM patterns by exploring quality aspects of knowledge, and we will use these definitions to describe best practices in KM.

## 3  QUALITY OF KNOWLEDGE

One fundamental goal of software engineering is the development of high quality, reliable and safe software at acceptable costs. Attention to software quality is important in software development not only because of its influence on long-term corporate goals, but also because of the increasing pervasiveness of software in everyday life. Software has become an enabling technology that is being used more and more as a product enhancement rather than as a standalone product. Hence, software quality is recognized within the industry as a key factor to guarantee market success.

There are various definitions of *software quality* and various ways of how to achieve it. The standard ISO-9126 (ISO/IEC-9126-1, 2003) as depicted in Fig. 2 defines a quality model for software that encompasses:

- *Internal quality factors,* concerned with static aspects that are visible to the developers but not to the user of the software system, such as maintainability or reusability.

- *External quality factors,* concerned with dynamic aspects that are visible only to the developers but not to the user of the software system (e.g., memory requirements when running the system).

- *Quality-in-use factors,* concerned with usability aspects (i.e., about the extent to which the software meets the needs of the users and not the ones of the developers).

In order to quantify a software system according to such a model, metrics are used to measure current quality factors and to develop strategies for improving the software system.

In KM the *quality of knowledge* is likewise important (Marwick, 2001) because what one person documents in a knowledge component (e.g., a Wiki page) might be read by several others and should therefore, have some good "quality-in-use" factors such as understandability (Kari, 1996) and preciseness – similar to requirements and other software artifacts in software engineering . The knowledge even has "external quality" factors that do not represent aspects when the knowledge is "executed" (i.e., dynamic aspects), but rather show how well it helps or solves a problem. Furthermore, the knowledge documented will (hopefully) have a lifetime of several years and will be analyzed, improved, and adapted. Therefore, the knowledge needs high "internal quality" factors to ensure its maintainability or portability (e.g., to another KM system).

However, as no quality model for knowledge (components) exist right now we present some quality characteristics that were transferred from software engineering and database technology. We present this to stimulate further discussions about the meaning of quality in the context of knowledge management. In the following, we describe some characteristics of high-quality knowledge based upon our experience and software quality models. We use the following terms to describe knowledge that was transferred to a knowledge base (i.e., a technical KM system such as a Wiki):

**Def:** **Knowledge Elements** are the most basic components that knowledge is stored in and cannot be further divided without destroying the ability to understand them using other knowledge fragments.

**Def:** **Knowledge Components** are complete and self-sufficient (i.e., independent of other knowledge elements) descriptions of knowledge (e.g. a SE artefact). A knowledge component consists of at least one or more knowledge elements.

From our point of view, knowledge components and elements should possess at least the following (non-complete) group of characteristics in order to foster high quality content:

- ***The AID Properties***: A knowledge element has to be *atomic*, i.e., the element cannot be divided further without destroying the ability to understand it without other knowledge elements. A knowledge component should be *independent* in its use, i.e., the component should be applicable (or reusable) without using other knowledge components. A knowledge component or element should be *durable*, that is, if it is still in the active knowledge base, the content should be valid. While knowledge might change over time as the environment shifts, the knowledge should be valid for the time it is applied or a warning should be attached. The AID properties are based on the ACID characteristics from database design (Haerder & Reuter, 1983).

- ***The four C's***: The content of knowledge components in a knowledge base has to be correct, complete, consistent, and concise. *Correct* means that it should not be wrong or ambiguous, *complete* means that all available and relevant information was recorded and no information is missing, *consistent* means that it was recorded in a style similar to other instances of this type of knowledge (e.g., by using templates), and *concise* means that the content is simple, adequate, and precise. The four C's are based on widespread used characteristics of software requirements (Sommerville & Sawyer, 1997).

These characteristics aim at supporting the authors and maintainers of knowledge bases in creating high-quality knowledge components.

In order to characterize the quality of a knowledge component, we transfer the concept of software quality models to KM. This is not to be confused with a *KM system quality model,* which would state quality factors for the technical software system (or social system) used for KM. A ***knowledge quality model*** groups several quality aspects that should be kept in mind when recoding knowledge as well as maintaining the knowledge base. Similar to software quality models, there are diverse quality models, and each knowledge domain might require a specific quality model emphasizing specific quality aspects.
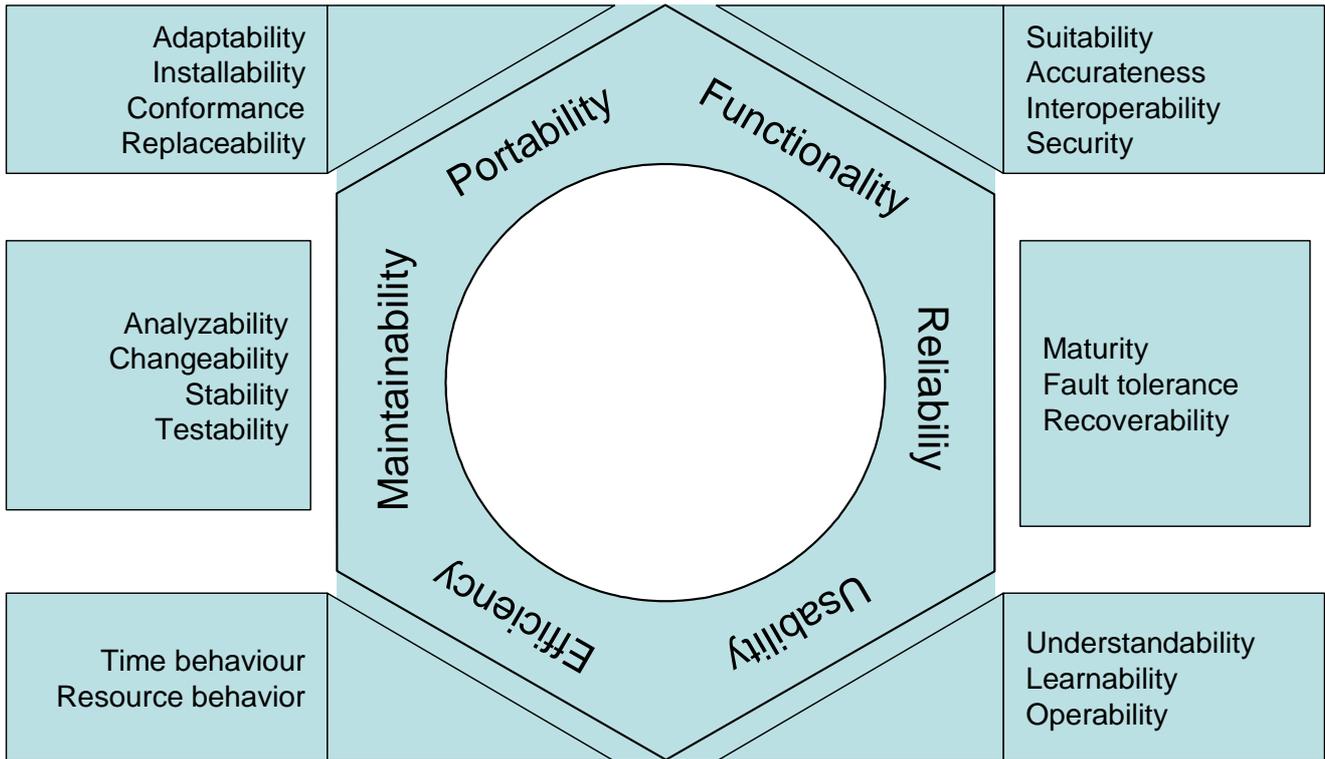
**Fig. 2** Quality Factors (including sub-characteristics based on ISO 9126)

The core quality factors of software quality models (ISO/IEC-9126-1, 2003) applied to knowledge are:

- *Functionality,* meaning that knowledge components should work in a suitable and accurate way as indicated,
- *Reliability,* meaning that the knowledge components should be mature and valid enough to not cause great damage and be easily undoable,
- *Usability,* meaning that the knowledge component should be easily understandable, learnable, and applicable,
- *Efficiency,* meaning that the knowledge component should state the quickest solution with the least resource requirements,
- *Maintainability,* meaning that the knowledge component should be easily changeable (e.g., low coupling or distribution), very stable, and testable (e.g., in thought experiments or fast case studies),
- *Portability,* meaning that the knowledge component should be adaptable to new contexts, conforming to internal (e.g., templates) or external (e.g., SCORM for course materials) standards, and be replaceable in larger knowledge arrangements.

## 4   PATTERNS AND ANTI-PATTERNS IN KNOWLEDGE MANAGEMENT

By transferring the concept of patterns to knowledge management, we therefore define knowledge and knowledge management patterns as follows:

**Def:   Knowledge Pattern**: A knowledge pattern is a general, proven, and beneficial solution to a common, reoccurring problem in knowledge design, i.e., the structuring and composition of the knowledge (e.g., on or via Wiki pages) or the ontology defining metadata and potential relationships between knowledge components.

In general, anti-patterns are the opposite of patterns and represent worst-practices that should not be applied. We define knowledge anti-patterns as follows:

**Def:   Knowledge Anti-Pattern**: A knowledge anti-pattern is a general, proven, and non-beneficial problem (i.e., bad solution) in a knowledge product, system, or process, i.e., the structuring and composition of the knowledge (e.g., on or via Wiki pages) or the ontology defining metadata and potential relationships between knowledge components.

In order to group and delimitate the patterns, we describe them in seven groups ranging from KM System patterns, via content patterns to KM maintenance patterns. For describing our (anti-)patterns, we use the following short template that was derived from more elaborate templates. It consists of the following sections:

- *Name*: What is the (anti-)pattern called?
- *Issue*: What is the issue (e.g., problem) addressed by this (anti-)pattern?
- *Q-Effect*: What "knowledge quality aspects", as described in section 3, are affected the most by this (anti-)pattern? In this section, we state if there is a positive (+), negative (-), or neutral (0) effect.
- *Solution*: What are the principal solutions underlying this pattern? Multiple alternative solutions might be given to remove an anti-pattern or build a pattern. In this section, we cite "knowledge refactorings", which are described in more detail in section 5.
- *Causes*: What are the basic causes of this (anti-) pattern?

The full template format to describe these patterns consists of additional information entities such as structure, dynamics, anecdotal evidence, example, or exceptions. The usage of the full template would be outside the scope of this paper.

### 4.1 Knowledge Content Patterns & Anti-Patterns

These patterns and anti-patterns apply to the content of a knowledge component as well as semantic relations between components. Typically, they are perceived from the viewpoint of the reader or writer.

| Name | Knowledge Blob Anti-Pattern | | |
|------|-----------------------------|---|---|
| Issue | The description of an experience or knowledge component gets larger and larger over time and subsumes more and more information.<br>The search for an arbitrary knowledge component will often include the knowledge blob. The knowledge blob can be used for different problems, has multiple solutions, or contact data. | | |
| Q-Effect | Functionality | - | |
| | Reliability | 0 | |
| | Usability | - | |
| | Efficiency | - | |
| | Maintainability | - | |
| | Portability | - | |
| Solution | *Compact Knowledge*: Summarize and rewrite the knowledge in a shorter form (e.g., on one page).<br>*Extract Elements*: Apply divide & conquer to create several mutually exclusive pages with parts of the original page.<br>*Extract Commonalities*: Find elements in other pages with overlapping knowledge and extract this overlapping element from both (or all) pages into a new page. | | |
| Causes | The KM system makes it easy to find and change (e.g., extend) a knowledge component, the users are not sensitized to create individual experiences, or there is no maintenance or change process established for the knowledge in the KM system. | | |

| Name | Superfluous Information Anti-Pattern |
|------|--------------------------------------|
| Issue | The description of an experience or knowledge component is too long and has information that is either not relevant to the topic, already stored elsewhere, or outdated.<br>The reader has to read more to get little relevant information, which might lead to an abandoned system. Furthermore, the description is longer than one page in the KM system and requires that the user scrolls and has to interrupt his learning mode. |

| Q-Effect | Functionality | 0 | |
|---|---|---|---|
| | Reliability | 0 | |
| | Usability | - | |
| | Efficiency | -/0 | |
| | Maintainability | -/0 | |
| | Portability | 0 | |

| | |
|---|---|
| Soluti on | *Compact Knowledge*: Summarize and rewrite the knowledge in a shorter form on one page <br> *Offer Templates*: Find all knowledge components of a specific type and offer a distinct template for every type. |
| Cau ses | The writer does not really know what to describe in order to produce a simple, short and comprehensive knowledge component. Additionally, this might be caused by missing guidelines how to structure a knowledge component and how to write for one or multiple target groups. |

| Name | *Unnecessary Refinement Anti-Pattern* | | |
|---|---|---|---|
| Issue | Multiple pages that are used to describe one topic are not reusable for other knowledge descriptions and all have to be read to understand the knowledge component. <br><br> The reader has to read several pages in order to understand the knowledge; he interrupts his learning mode, and might interrupt or stop the learning activity completely. Furthermore, a search on the knowledge base might return only a page within this knowledge chain. | | |
| Q-Effect | Functionality | 0 | |
| | Reliability | 0 | |
| | Usability | - | |
| | Efficiency | 0 | |
| | Maintainability | - | |
| | Portability | 0 | |
| Soluti on | *Compact Knowledge*: Summarize and rewrite the knowledge in a shorter form on one page. | | |
| Caus es | The writer either tries to over-generalize the knowledge (e.g., he thinks that every small piece of knowledge might be used in other knowledge components) or does not really know what to do in order to produce a simple, short and coherent knowledge component. | | |

| Name | *Duplicated Knowledge Anti-Pattern* | | |
|---|---|---|---|
| Issue | Multiple versions of the same information reside in different locations in the knowledge base. <br><br> The change of one piece of information causes changes to be made on several pages of different knowledge components. If not all replications are changed as well, multiple, slightly different versions might exist in the knowledge base. | | |
| Q-Effect | Functionality | 0 | |
| | Reliability | 0 | |
| | Usability | 0 | |
| | Efficiency | 0 | |
| | Maintainability | - | |
| | Portability | 0 | |
| Soluti on | *Compact Knowledge*: Summarize and rewrite the knowledge in a shorter form on one page. <br> *Extract Commonalities*: Find elements on other pages with overlapping knowledge and extract this overlapping element from both (or all) pages to a new page. | | |
| Ca us es | Writers are not aware of or do not care about similar knowledge. Furthermore, either the knowledge base is not cleaned up from time to time or similar knowledge components are not aggregated. | | |

### 4.2  Knowledge Usage Patterns & Anti-Patterns

These patterns and anti-patterns apply to the maintenance of knowledge components or elements and are typically perceived from the view of the knowledge maintainer or gardener.

| Name | Dead Knowledge Anti-Pattern | | |
|---|---|---|---|
| Issue | A knowledge component is considered useless, is not (re-)used anymore by the users, and wastes either space in the knowledge base or computational power from the server of the KM system (e.g., in search algorithms). | | |
| Q-Effect | Functionality | -/0 | |
| | Reliability | -/0 | |
| | Usability | - | |
| | Efficiency | 0 | |
| | Maintainability | - | |
| | Portability | - | |
| Solution | *Fuse Knowledge*: Find a similar and "non-dead" knowledge component and integrate the remaining useful information (i.e., combine, compact, or rewrite their descriptions).<br>*Forget Knowledge*: Remove the knowledge from the knowledge base (maybe after an inspection by possibly interested parties). | | |
| Causes | The knowledge is outdated right from the start, too specific, or too general. This can be caused by incentive systems that "pay" for contributions but do not monitor the quality, or by authors who do not really know what to do in order to produce a simple, short and comprehensive knowledge component. | | |

| Name | Invisible Knowledge Anti-Pattern | | |
|---|---|---|---|
| Issue | Knowledge is not used anymore by the system and is undiscoverable by the users. While it might be useful to the users, it cannot be reused anymore and wastes space or computational power (e.g., in search algorithms). | | |
| Q-Effect | Functionality | 0 | |
| | Reliability | 0 | |
| | Usability | - | |
| | Efficiency | 0 | |
| | Maintainability | - | |
| | Portability | 0 | |
| Solution | *Reintegrate Knowledge*: Reintegrate the component in the search index or an applicable navigational structure.<br>*Fuse Knowledge*: Find a similar and "non-dead" knowledge component and fuse them together (i.e., combine, compact, or rewrite their descriptions).<br>*Forget Knowledge*: Remove the knowledge from the knowledge base (maybe after an inspection by possibly interested parties). | | |
| Causes | The knowledge is not linked anymore and does not show up in any navigational structures or search results. This might be caused by knowledge refactorings, knowledge base gardening activities, or the KM system itself. | | |

### 4.3  Knowledge Ontologies Patterns & Anti-Patterns

These patterns and anti-patterns apply to the ontologies used to structure knowledge components or elements and are typically perceived from the viewpoint of the ontology developer.

| Name | Template Pattern |
|---|---|
| Issue | Knowledge components of a specific type (e.g., patterns) that have different structures are harder to understand because, on the one hand, the reader first has to understand how the knowledge is structured (e.g., where is the problem statement?) and, on the other hand, the writer must remember how to describe a complete component (e.g., he should not forget the problem statement). |

| Q-Effect | Functionality | + |
| | Reliability | + |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | 0 |

| Solution | *Offer Templates:* Every type of knowledge should have a uniform representation. Find all knowledge components of a specific type and offer a distinct template for every type. |

| Causes | The writers are free to describe their knowledge and typically use their own structure or write as it seems fit. This might also be caused by different standards used in separate projects (e.g., to describe requirements). |

| Name | *Landmark Pattern* |
|------|-----------------|

| Issue | Knowledge components that have no linked start page typically confuse the reader. The reader might miss some crucial information from the previous pages if the found page is directly linked in a search result. |

| Q-Effect | Functionality | 0 |
| | Reliability | + |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | + |

| Solution | The knowledge described, especially if it is distributed over multiple pages, should always have at least one starting point that is linked from all subsequent pages.<br>*Link to start page*: Either link to a specific start page where one should start reading the knowledge, or link to an overview listing all starting points for this knowledge. |

| Causes | The search technique uses all pages and returns a hit list including knowledge elements that are meaningless or at least hard to understand by themselves. Alternatively, entry points to understand knowledge elements are not integrated by the knowledge authors. |

| Name | *Overview Pattern* |
|------|-----------------|

| Issue | Knowledge components that semantically belong together and have no overview page that lists them are harder to find and several of them might get lost in a simple search. |

| Q-Effect | Functionality | 0 |
| | Reliability | 0 |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | 0 |

| Solution | *List knowledge*: Find all knowledge components related to a specific topic and generate a page for this topic listing all these knowledge components. |

| Causes | The writer is not informed about other, similar knowledge components and does not integrate the new component into existing structures. |

| Name | *Ambiguous Relations Anti-Pattern* |
|------|--------------------------------|

| Issue | Links between knowledge elements that belong to the same knowledge component are not clearly described and defined. The same holds for the relations between different knowledge components. Authors are confused if there are too many or too similar relations that are to be assigned manually. |
|---|---|

| Q-Effect | Functionality | 0 |
|---|---|---|
| | Reliability | + |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | + |

| Solution | Named relations help users to navigate thru the KM system content and can also be used to check for a complete documentation (e.g., are refinements and links to process descriptions included?) |
|---|---|
| | Define clearly named relations between the knowledge elements (e.g., is_refined_by, part_of_process) and different knowledge components (e.g., use_with, applied_in_project, measured_with). Use predefined relations to define learning-cycles and support systematic learning. |
| | *Minimize Set of Relations*: Minimize the set of relations available. Use same naming for relations with (similar) objectives (e.g., used in/used_by). Fuse relations that are very similar or identical – possibly by introducing a more general relation. |
| | *Provide Authoring Guidelines:* Clearly describe how classes of knowledge have to be structured, which relations and metadata have to be used, and how existing templates may be used. |
| | *Offer Templates*: Wherever possible, offer templates for inserting or editing knowledge in the KM system. |

| Causes | No authoring guidelines, templates, or tool supported authoring environments are offered. Concrete rules for adding and editing the KM system content are missing. As a consequence, each author/editor uses a different naming and refinement structure. The content grows in an uncontrolled and often confusing way. This case can often be found in Wiki-based systems. |
|---|---|

| Name | *Common Metadata Pattern* |
|---|---|

| Issue | KM systems frequently store different kinds of knowledge elements. The metadata describing each of these knowledge classes typically varies (e.g., "programming_language" makes sense with code components but not with tool descriptions or lessons learned). However, to generate indexes and catalogs, or to easily address all KM system entries in a search algorithm (e.g., retrieve the ten newest entries), a minimal set of common metadata, shared by all entries of the KM system, is needed. |
|---|---|

| Q-Effect | Functionality | 0 |
|---|---|---|
| | Reliability | 0 |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | + |

| Solution | *Create common metadata class*: Common metadata (e.g., entry name, creation date, author) can be applied to all KM system entries (cf., file system attributes). This common set can be enriched by specific attributes for maintenance (e.g., "last_reviewed", "version_number") and access control (e.g., "allowed_user_roles"). This common metadata class is inherited by all other knowledge classes of the KM system and extended for each specific type of knowledge element. |
|---|---|
| | *Provide Authoring Guidelines:* Clearly describe how classes of knowledge have to be structured, which relations and metadata have to be used, and how existing templates may be used. |
| | *Offer Templates*: Wherever possible, offer templates for inserting or editing knowledge in the KM system. |

| Causes | No rules for editing and adding content exist. Common describing metadata (e.g., such as the file system attributes) are often forgotten by authors or used with different names (e.g., name, ID, entry_name, etc.). |
|---|---|

| Name | *Useless Metadata Anti-Pattern* |
|---|---|

| Issue | The ontology consists of (too) many characteristics that describe useless (or all imaginable) aspects of the knowledge. Furthermore, the metadata does not provide any direct benefit to the users. |
|---|---|

| Q-Effect | Functionality | 0 |
|---|---|---|
| | Reliability | 0 |
| | Usability | -/0 |
| | Efficiency | 0 |
| | Maintainability | - |
| | Portability | + |

| Solution | Use only metadata that is required for specific tasks in the usage or maintenance of the knowledge base. *Remove Metadata*: Remove metadata that is not used by a function or by the typical users (readers and maintainers). *Fuse Metadata*: Find similar metadata and either use only one of them or create a new metadata that represents the essential facts from all of them. |
|---|---|

| Causes | The ontology designers tried to "develop for the future" and for every possible use of the system. The essential goals of the system as stated by the stakeholders are diluted by many other "fancy" ideas. |
|---|---|

## 4.4 Knowledge Presentation Patterns & Anti-Patterns

These patterns and anti-patterns apply to the presentation of knowledge components or elements in a KM system and are typically perceived from the viewpoint of the reader or writer.

| Name | *Sequential Reading Pattern* |
|---|---|
| Issue | Knowledge components that are not arranged in a sequential order often confuse or distract the reader. When knowledge is not presented in a logical order, the user might not be able to understand and apply it correctly. |

| Q-Effect | Functionality | 0 |
|---|---|---|
| | Reliability | + |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | 0 |
| | Portability | 0 |

| Solution | Knowledge should be presented in a way that allows sequential reading. The actual serialization (i.e., at "runtime" or "read-time") depends on the user's needs. *Serialize Knowledge*: Serialize every coherent knowledge block (e.g., component). |
|---|---|

| Causes | Multiple rewrites of the knowledge component were made, but with different activity flows. This caused a non-linear description of possible ways, for example, to solve a problem. |
|---|---|

| Name | *Large Template Anti-Pattern* |
|---|---|
| Issue | Authors have to fill out many metadata fields manually to describe their knowledge and consequently lose interest in recording their knowledge in the KM system. |

| Q-Effect | Functionality | 0 |
|---|---|---|
| | Reliability | - |
| | Usability | - |
| | Efficiency | 0 |
| | Maintainability | - |
| | Portability | - |

| Solution | Elicit as many metadata as possible using automated techniques and reduce the amount of manually requested ones. For example, administrative information (author name, creation / modification date) can be derived from the system or underlying workflow. Furthermore, classification terms, e.g., from SWEBOK (SWEBOK, 2004) can be automatically inferred and suggested by comparing the content or context of the new knowledge component to already existing components. *Create Template Defaults*: Templates used should have meaningful default values. If defaults are not used, this indicates the need for an edited or additional template. *Shorten Template*: Reduce the size of the template elements. |
|---|---|

| Causes | The KM system is either based on a very large ontology or it offers (too) many features and aims at supporting many application scenarios that require this amount of structured information. |
|---|---|

| Name | *Unique Presentation Pattern* | | |
|---|---|---|---|
| Issue | Different representations of knowledge components confuse and distract the reader. While every user groups might want a unique way of representation, it should be fixed before the launch of the KM system. | | |
| Q-Effect | Functionality | 0 | |
| | Reliability | 0 | |
| | Usability | + | |
| | Efficiency | 0 | |
| | Maintainability | 0 | |
| | Portability | 0 | |
| Solution | Every type of knowledge should have a uniform, distinguishable form of presentation (e.g., by using a specific template, color, or a screen design). Nevertheless, it should be identical for all knowledge components of this type. *Offer Template*: Design and use a template for every type of knowledge (e.g., based on standards). | | |
| Causes | The writers are free to describe their knowledge and typically use their own structure or write as it seems fit. This might also be caused by different standards used in separate project (e.g., to describe requirements). | | |

| Name | *Information Flood Anti-Pattern* | | |
|---|---|---|---|
| Issue | Too many knowledge components (which might even be very similar) are presented to the user, for example, in the form of all knowledge components found by a search. | | |
| Q-Effect | Functionality | 0 | |
| | Reliability | 0 | |
| | Usability | - | |
| | Efficiency | 0 | |
| | Maintainability | - | |
| | Portability | 0 | |
| Solution | *Aggregate Knowledge*: Find and integrate similar knowledge components into a single, more general knowledge component. *Chunked Presentation*: A search should partition the results and present only a small amount (chunks) of knowledge components (e.g., 10). *Cluster Results*: Results from a search should be clustered to identify groups of similar pages. *Refine Classification*: Find subgroups of classes in a classification that has too many components and reclassify the concerned knowledge components. | | |
| Causes | As a knowledge base grows a large amount of knowledge components are amassed with potentially similar contents. This is typically caused by missing knowledge gardening activities or a large number of separated, but similar projects. | | |

### 4.5 Knowledge Transfer Patterns & Anti-Patterns

These patterns and anti-patterns apply to the transfer (i.e., reading, understanding, and application) of knowledge components or elements and are typically perceived from the viewpoint of the reader or writer.

16

| Name | Context-based Enrichment Pattern | |
|---|---|---|
| Issue | Information stored in a knowledge element is often described solely from the point of view of the author. An expert might take contextual knowledge for granted that a novice does not possess. | |
| Q-Effect | Functionality | + |
| | Reliability | + |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | +/- |
| | Portability | +/- |
| Solution | *Describe Context*: Clearly describe what the circumstances of the knowledge are and who (with what level of expertise) created it.<br>*Link Knowledge*: Link every concept that is used in the description with a similar concept (or training course) in the knowledge base. | |
| Causes | The writer does not really know how to describe knowledge in order to produce a reusable, simple, short and comprehensive knowledge component. Additionally, this might be caused by time constraints, lack of management support, or missing feedback (i.e., ignorance of the problem). | |

| Name | Notification Pattern | |
|---|---|---|
| Issue | In a static knowledge base, the user is not informed about changes to knowledge components he has written, used before, or needs on a daily basis. The user is not informed about updates to his contributions or to knowledge components he is currently using (e.g., in a project). | |
| Q-Effect | Functionality | + |
| | Reliability | + |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | 0 |
| Solution | *Monitor Knowledge*: Users should be able to monitor pages and be notified if changes are made to a knowledge component (especially if he is the author).<br>*Monitor Ontology*: A user should be able to monitor part of the ontology and be notified if changes are made to it (e.g., to reclassify their own pages). | |
| Causes | The observation by automated notification helps to keep up to date with a knowledge component especially if the knowledge is currently still used in a project. | |

### 4.6 KM Systems Organization Patterns & Anti-Patterns

These patterns and anti-patterns apply to organizational aspects of the technical infrastructure of knowledge management systems (KMS) such as Wikis and are typically perceived from the viewpoint of a KM system (e.g., Wiki) developer or administrator.

| Name | One Group One Area Pattern | |
|---|---|---|
| Issue | Often, a KM system or a delimited area therein (e.g., a project area) is used by specific groups with no or small overlaps in interests. This causes the decrease of the percentage of relevant content for each individual group as the system grows. | |
| Q-Effect | Functionality | 0 |
| | Reliability | 0 |
| | Usability | + |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | -/0 |

| Solution | For each group in an organization such as a project, there should be a specific area or KMS. Further integration of these areas is done via a centralized search function or inter-system linking. *Separate Concerns*: Delimitate areas of groups with different interests (e.g., using group-specific pages, access rights or individual KMS). |
|---|---|
| Causes | The authors of different groups describe knowledge in a very similar way and cause collisions between the knowledge spaces. By structuring the KM area or system into smaller and non-related parts, the overlap of knowledge decreases. |

| Name | *VOIC Pattern* | |
|---|---|---|
| Issue | If a KM system does not separate the subsystems for presentation (View), structure (Ontology), rules (Inference), and knowledge (Content), its maintenance will become hard. | |
| Q-Effect | Functionality | 0 |
| | Reliability | + |
| | Usability | 0 |
| | Efficiency | 0 |
| | Maintainability | + |
| | Portability | + |
| | Comment: These quality aspects are meant to describe the quality of the system (e.g., the maintainability of the system) and not the knowledge within! | |
| Solution | The VOIC (View, Ontology, Inference, Content) pattern states that the different architectural subsystems should be separated. In general, this is an extension of the MVC pattern including the ontology layer. The separation improves the extension and modification of the individual subsystem (e.g., it is easier to exchange or improve the ontology without looking for changes to be made in the rest of the system). Separate VOIC: Identify and separate view, ontology, inference, and content parts in the KM system. | |
| Causes | Typically, a KM system is either a proprietary or open source system that is tailored for a single purpose. Due to time or design constraints, information is not explicitly separated, and information (e.g., about the ontology) is hard-wired into the system, which, for example, decreases the adaptation to new ontologies. | |

## 4.7 Social KM Patterns & Anti-Patterns

These patterns and anti-patterns apply to the social system or purely human-based part of KM systems and are typically perceived from the viewpoint of a KM system (e.g., a Wiki) developer or administrator. For example, they can be implemented via social rules and supported by architectural styles and patterns.

| Name | *Coffee Kitchen Pattern* | |
|---|---|---|
| Issue | Exchange between colleagues does not take place, as there is no place to meet such as a coffee kitchen, smoker's corner, or reading room. The less the employees know each other, the less information is shared between them. | |
| Q-Effect | Functionality | 0 |
| | Reliability | 0 |
| | Usability | + |
| | Efficiency | + |
| | Maintainability | 0 |
| | Portability | + |
| Solution | *Arrange Meeting Space*: Provide one or more comfortable places to meet such as a coffee kitchen with chairs or a sofa (i.e., a lounge area). | |
| Causes | Either the management does not want people to talk and share information (i.e., waste time), or they gave multiple coffee machines to their employees (e.g., one coffee machine per department or group). | |

| Name | *Knowledge Meetings Pattern* |
|---|---|
| Issue | The content of the (technical and human) knowledge base is not used due to time pressure or doubts about whether the content is helpful. Furthermore, the content affects operational and managerial decisions. |

18

| | Functionality | 0 |
|---|---|---|
| **Q-Effect** | Reliability | + |
| | Usability | + |
| | Efficiency | + |
| | Maintainability | 0 |
| | Portability | + |

| | |
|---|---|
| **Solution** | Knowledge should be a part of every meeting and the communication between employees should be fostered.<br>*Talk About Knowledge*: Present new knowledge and discuss knowledge demands as part of regular meetings (e.g., departmental meetings).<br>*Explicit Knowledge Meetings*: Hold explicit meetings about specific topics (e.g., a new technology) with the goal of distributing knowledge and eliciting knowledge gaps. |
| **Causes** | The people are either not willing to use the KM system and communicate with their colleagues or fear virtual barriers (e.g., feudal lords syndrome). Additionally, this is often caused by stress and time constraints. |

## 5   KNOWLEDGE REFACTORING

During the few last years, refactoring has become an important part in agile processes for improving the structure of software systems between development cycles. Especially in agile development, under-engineering usually happens when the focus lies on adding more functionality to a system without improving its design along the way. When code works, it is often simpler to engage the next task than clean up the previous work. Additionally, as systems are getting larger, refactoring gets more and more complex and time consuming to do manually. Even if one knows how to refactor software, it is not clear where and under what conditions which refactoring should be used (Fowler & Beck, 1999).

In practice, refactoring is a great challenge, as most software systems are badly implemented and therefore hard to evolve, maintain, and reengineer (e.g., Y2K). This becomes worse if the software has to be optimized in order to meet new requirements, remove defects, or improve qualities like maintainability or reusability.

In order to improve the quality of a knowledge component representing a knowledge anti-pattern, we transferred the concept of software refactoring (Fowler & Beck, 1999) to *knowledge refactoring*. A software refactoring is a formal transformation process that describes how a software component (e.g., a class in an object-oriented system) can be changed in order to improve its quality. It is defined as follows:

**Def:  Software refactoring**: A (software) refactoring is an explicit, replicable, and beneficial activity that transforms the structure or representation of a software component without changing its meaning. The goal of software refactoring is the improvement of the quality (e.g., maintainability; see section 3) of the software system.

We use the term "knowledge refactoring" defined as follows:

**Def:  Knowledge refactoring**: A (knowledge) refactoring is an explicit, replicable, and beneficial activity that transforms the structure or representation of a knowledge element or component without changing its meaning. The goal of knowledge refactoring is the improvement of the quality (e.g., understandability; see section 3) of the documented knowledge.

For improving the above mentioned anti-patterns, we used the following activities, which we learned and applied in several of our projects, for the refactoring of knowledge components or KM systems. While these could be classified by their executive role (e.g., ontology developer or knowledge author) and the affected product (e.g., templates or knowledge elements), we merely list them in this paper:

- **Compact Knowledge**: Summarize and rewrite the knowledge in a shorter form such that it fits the requirements (e.g., that it fits on one page). Find and remove filler words or superfluous sentences that do not change the meaning of the knowledge.
- **Extract Elements**: Find and extract parts of the description that semantically belong together and can be extracted into a self-sufficient knowledge element. Recursively apply this "divide & conquer" strategy to create several mutually exclusive knowledge elements with parts of the original knowledge component until all resulting knowledge elements fit the requirements.

- **Extract Commonalities**: Find parts of two different knowledge components that semantically belong together and extract them into a self-sufficient knowledge element.
- **Fuse Knowledge:** Find a similar knowledge component and extract useful knowledge not existent in the other components. Fuse this useful knowledge with the other knowledge components and remove the remaining original knowledge component (i.e., apply "Forget Knowledge").
- **Forget Knowledge:** Remove the knowledge component from the knowledge base. This might also be realized by marking the knowledge component as dead, removing it from any navigational structures and the search indices, and having it inspected by interested parties or experts.
- **Describe Context:** Clearly describe what the circumstances of the knowledge are and who (with what level of expertise) created it.
- **Reintegrate Knowledge:** Integrate the knowledge element into existing navigational structures (e.g., in lists of knowledge classes or higher-level knowledge components) or search indices.
- **Link To Start Page:** Identify one or more starting points for every knowledge component and mark them respectively. Statically or dynamically link every knowledge element (or every page) to one or all starting points.
- **Aggregate Knowledge:** Similar to "Fuse Knowledge", but with the difference that one has to find several similar knowledge components, extract the common or general knowledge within, and then create a knowledge component with this generalized knowledge. The original knowledge components are not dismissed or forgotten but remain in the knowledge base as concrete or specific knowledge. For example, the aggregation of multiple similar experiences would lead to a new pattern (e.g., about software testing) or a "rule of thumb".
- **Chunk Presentation:** Find and subdivide a search result list into several chunks (e.g., ten links) and present these chunks on a single page.
- **Cluster Results:** Find similar knowledge elements and present them by their commonality. These clusters might be based on the text (i.e., Wiki page content), knowledge type (e.g., requirement documents), or other similarities.
- **List Knowledge**: Find all knowledge components related to a specific topic and generate a page for this topic listing all these knowledge components.
- **Link Knowledge:** Find and link every concept (i.e., in the form of a word or phrase) in the description of a knowledge element with the respective knowledge component in the knowledge base.
- **Serialize Knowledge:** Serialize every coherent knowledge block (e.g. component).
- **Refine Classification:** Find classes with (too) many elements in a classification (e.g., ontology) and refine this concept (i.e., class) by identifying and introducing new meaningful subclasses.
- **Create Common Metadata Class**: Common metadata (e.g., entry name, creation date, author) can be applied to all KM system entries (cf., file system attributes). This common set can be enriched by specific attributes for maintenance (e.g., "last_reviewed", "version_number") and access control (e.g., "allowed_user_roles"). This common metadata class is inherited by all other knowledge classes of the KM system and extended for each specific type of knowledge element.
- **Remove Metadata:** Remove metadata that is not used by a function or by the typical users (readers and maintainers).
- **Fuse Metadata:** Find similar metadata and either use only one of them or create a new metadata that represents the essential facts from all of them.
- **Minimize Set of Relations**: Minimize the set of relations available. Use same naming for relations with (similar) objectives (e.g., used in/used_by). Fuse relations that are very similar or identical – possibly by introducing a more general relation.
- **Provide Authoring Guidelines:** Clearly describe how classes of knowledge have to be structured, which relations and metadata have to be used, and how existing templates may be used.
- **Offer Templates:** Find a class of knowledge components that have a common topic, extract common headlines or semantical blocks, and identify the information offer and need of the knowledge users. From this information, synthesize a comprehensive template for this knowledge class.
- **Create Template Defaults:** Templates used should have meaningful default values. If defaults are not used, this indicates the need for a modified or additional template.
- **Shorten Template:** Reduce the size of the template elements.

- **Separate Concerns:** Delimitate areas of groups with different interests (e.g., using group-specific pages, access rights or individual KMS).
- **Arrange Meeting Space:** Provide one or more comfortable places to meet such as a coffee kitchen with chairs or a sofa (i.e., a lounge area).
- **Talk About Knowledge:** Present new knowledge and discuss knowledge demands as part of regular meetings (e.g., department meetings).
- **Explicit Knowledge Meetings:** Hold explicit meeting about specific topics (e.g., a new technology, project retrospectives) with the goal of distributing knowledge and eliciting knowledge gaps.

In general, there are a lot more refactoring activities that could be used to remove these and other anti-patterns or generate patterns. Be aware that these refactoring activities are not classified and are of various granularity. Some are used to change or transform a knowledge element while others are used to change the visual representation or KM system.

## 6   CONCLUSION AND FUTURE WORK

Knowledge management or software reuse in learning (software) organizations is often accompanied by the poor quality of the knowledge, experiences, or decisions within a KM system as well as the quality of the KM system itself.

We described an approach to structure knowledge in knowledge management systems in the form of so called *knowledge patterns*. These patterns and anti-patterns can be used to develop KM systems and improve the quality of the systems themselves as well as that of the knowledge within (i.e., the *quality of knowledge*). Furthermore, we transferred the concepts of software refactoring and software quality to describe the effect of knowledge patterns as well as countermeasures (i.e., *knowledge refactorings*) to remove knowledge anti-patterns.

Our argument is that the use of knowledge patterns to describe how knowledge and KM systems should or should not be structured generated several positive effects. First, researchers and practitioners in the KM field can reuse these patterns to build or reconstruct their knowledge bases. Second, the patterns might be used as a language extension to efficiently and precisely communicate about knowledge and KM systems using these patterns.

While this paper represents the first step in the formalization of best and worst practices (i.e., knowledge) about knowledge and KM systems, it is not the ultimate set of knowledge patterns. The knowledge patterns in this paper are based upon experiences from the development of several KM systems in projects such as RISE, ESERNET, SFB-EB, or VSEK within the German and European culture area. However, several of the patterns described might not be applicable or be misleading in a specific context or in other cultural backgrounds. Furthermore, there are many other patterns that could not be described in the context of this paper. In order to make the knowledge stored in the semi-formal patterns applicable in other contexts they should be applied in a non-SE context. Furthermore, if more experiences about a specific topic exists (e.g., knowledge structuring) this corpora of knowledge might be further generalized (i.e., de-contextualized) as described in the "knowledge dust to pearls" approach (Basili et al., 2001).

Hopefully, this paper will stimulates the discussion about the meaning of quality in the context of knowledge management, about how knowledge should (and should not) be described in a KM system, and what is needed to generate a fruitful socio-technical KM system. Furthermore, we envision that more researchers and practitioners will document experiences about knowledge and KM systems in form of patterns like in the software engineering domain. Additionally, this might even encourage and foster the development of automatism and knowledge refactoring tools.

## 7   ACKNOWLEDGEMENT

## 8 REFERENCES

Awazu, Y. (2004). *Knowledge management in distributed environments: roles of informal network players.* Paper presented at the System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on.

Basili, V. R., Caldiera, G., & Cantone, G. (1992). A reference architecture for the component factory. *ACM Transactions on Software Engineering and Methodology, 1*(1), 53-80.

Basili, V. R., Caldiera, G., & Rombach, H. D. (1994). Experience Factory. In J. J. Marciniak (Ed.), *Encyclopedia of Software Engineering* (Vol. 1, pp. 469-476). New York: John Wiley & Sons.

Basili, V. R., Costa, P., Lindvall, M., Mendonca, M., Seaman, C., Tesoriero, R., et al. (2001). *An experience management system for a software engineering research organization.* Paper presented at the 26th Annual NASA Goddard Software Engineering Workshop.

Basili, V. R., Lindvall, M., & Costa, P. (2001, June 13-15, 2001). *Implementing the Experience Factory concepts as a set of Experience Bases.* Paper presented at the Proc. 13th Int. Conf. on Software Engineering & Knowledge Engineering, Buenos Aires, Argentina.

Basili, V. R., & Rombach, H. D. (1988). The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Transactions on Software Engineering, 14*(6), 758-773.

Basili, V. R., Zelkowitz, M. V., McGarry, F., Page, J., Waligora, S., & Pajerski, R. (1995). SEL's software process improvement program. *IEEE Software, 12*(6), 83-87.

Bick, M., Hanke, T., & Adelsberger, H. H. (2003). A process-oriented analysis of knowledge distribution barriers. *Industrie Management, Germany * vol 19 (2003), no 3, p 37 40, 13 refs.*

BPC. (2005). Best Practices Clearinghouse.  Retrieved 11. December, 2005, from https://acc.dau.mil/bpch

Briand, L. C., Basili, V. R., Yong, M. K., & Squier, D. R. (1994). *A change analysis process to characterize software maintenance projects.* Paper presented at the In Proceedings. International Conference on Software Maintenance, Los Alamitos, CA, USA.

Brown, W. J., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). *AntiPatterns: refactoring software, architectures, and projects in crisis.* New York: John Wiley & Sons, Inc.

Damodaran, L., & Olphert, W. (2000). Barriers and facilitators to the use of knowledge management systems. *Behaviour and Information Technology, UK * vol 19 (Nov. Dec. 2000), no 6, p 405 13, 21 refs.*

Dangle, K., Dwinnell, L., Hickok, J., & Turner, R. (2005). Introducing the Department of Defense Acquisition Best Practices Clearinghouse. *CrossTalk*, 4-5.

Davenport, T. H., & Probst, G. (Eds.). (2000). *Knowledge Management Case book: Siemens Best Practices* (2nd ed.). Erlangen: Publicis MCD.

Decker, S., Park, J., Quan, D., & Sauermann, L. (2005, 6. November 2005). *Workshop2: The Semantic Desktop - Next Generation Information Management and Collaboration Infrastrucutre.* Paper presented at the International Semantic Web Conference (ISWC 2005), Galway, Ireland.

Eberle, M. A. (2003). *Barrieren und Anreizsysteme im Wissensmanagement und der Software-Wiederverwendung.* Unpublished Student Thesis, University of Kaiserslautern, Kaiserslautern.

Efimova, L., & Swaak, J. (2002). *KM and (e)-learning: towards an integral approach?* Paper presented at the EKMF.

Fahey, L., & Prusak, L. (1998). The Eleven Deadliest Sins of Knowledge Management. *California Management Review, 40*(3), 265-276.

Favaro, J. (1991). *What price reusability?: a case study.* Paper presented at the Proceedings of the first international symposium on Environments and tools for Ada, Redondo Beach, California, United States.

Feldmann, R. L. (1999). *On developing a repository structure tailored for reuse with improvement.* Paper presented at the Workshop on Learning Software Organizations (LSO) co-located with the 11th International Conference on Software Engineering and Knowledge Engineering, SEKE'99, Kaiserslautern, Germany.

Feldmann, R. L., & Carbon, R. (2003). Experience Base Schema Building Blocks of the PLEASERS Library. *Journal of Universal Computer Science (JUCS), 9*(7), 659-669.

Feldmann, R. L., & Pizka, M. (2003, 6 Aug. 2002). *An on-line software engineering repository for Germany's SME - an experience report.* Paper presented at the Advances in Learning Software Organizations. 4th International Workshop, LSO 2002, Chicago, IL, USA.

Feurstein, M., Natter, M., Mild, A., & Taudes, A. (2001). Incentives to share knowledge. *Proceedings of Hawaii International Conference on System Sciences. HICSS 34, Maui, HI, USA, 3 6 Jan. 2001 * Los Alamitos, CA, USA: IEEE Comput. Soc, 2001, p 8 pp.*

Fincher, S., & Utting, I. (2002). Pedagogical patterns: their place in the genre. *SIGCSE Bulletin, USA * vol 34 (Sept. 2002), no 3, p 199 202, 18 refs.*

Fowler, M., & Beck, K. (1999). *Refactoring : improving the design of existing code.* Reading, MA: Addison-Wesley.

Gamma, E., Richard, H., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software* (3rd printing ed. Vol. 5): Addison-Wesley.

Haerder, T., & Reuter, A. (1983). Principles of transaction-oriented database recovery. *ACM Comput. Surv., 15*(4), 287-317.

Halvorsen, K., & Nguyen, M. (1999, 16-19 June 1999). *A Successful Software Knowledge Base.* Paper presented at the 11th Conf. on Software Engineering and Knowledge Engineering (SEKE'99), Kaiserslautern, Germany.

Harrison, W. (2004). Best Practices: Who Says? *IEEE Software, 21*(1), 8-9.

HPL. (2005). Hillside Pattern Library.  Retrieved 10. Oct., 2005, from http://hillside.net/patterns/

ISO/IEC-9126-1. (2003). *Software engineering : product quality. Part 1, Quality model* (Ed. 1. ed.). Pretoria: International Organization for Standardization/International Electrotechnical Commission.

Jedlitschka, A., & Ciolkowski, M. (2004, 19-20 Aug. 2004). *Towards evidence in software engineering.* Paper presented at the International Symposium on Empirical Software Engineering, Redondo Beach, CA, USA.

Jen, H. W., & Yu, M. W. (2006). Measuring KMS success: A respecification of the DeLone and McLean's. *Information and Management, 43*(6), 728-739

Jennex, M. E., & Olfman, L. (2004). *Assessing knowledge management success/effectiveness models.* Paper presented at the SO: Proceedings of the 37th Annual Hawaii International Conference on System.

Jennex, M. E., & Olfman, L. (2006). A model of knowledge management success. *International Journal of Knowledge Management, USA * vol, 2*, July-Sept.

Judicibus, D. d. (1996, 8-9 Jan. 1996). *Reuse: A cultural change.* Paper presented at the Proceedings of the International Workshop on Systematic Reuse: Issues in Initiating and Improving a Reuse Program, Liverpool, UK.

Kamel, A., M., C., & Sorenson, P. (2001, 30 July - 2 August 2001). *Building an Experience-Base for Product-line Software Development Process.* Paper presented at the Fourth International Conference on Case-Based Reasoning, Vancouver, British Columbia, Canada.

Kari, L. (1996). Estimating understandability of software documents. *SIGSOFT Softw. Eng. Notes, 21*(4), 81-92.

Koennecker, A., Jeffery, R., & Low, G. (2000, 28-29 April 2000). *Implementing an experience factory based on existing organisational knowledge.* Paper presented at the Proceedings 2000 Australian Software Engineering Conference, Canberra, ACT, Australia.

Lindvall, M., Frey, M., Costa, P., & Tesoriero, R. (2001, 12-13 Sept. 2001). *Lessons learned about structuring and describing experience for three experience bases.* Paper presented at the Third International Workshop on Advances in Learning Software Organizations (LSO 2001), Kaiserslautern, Germany.

Lippert, M., Becker, P. P., Breitling, H., Koch, J., Kornstadt, A., Roock, S., et al. (2003). Developing complex projects using XP with extensions. *Computer, USA * vol 36 (June 2003), no 6, p 67 73, 8 refs.*

Long, J. (2001). Software reuse antipatterns. *Software Engineering Notes, 26*(4), 68-76.

Markkula, M. (1999, 16-19 June 1999). *Knowledge management in software engineering projects.* Paper presented at the Proceedings of SEKE'99: Eleventh International Conference on Software Engineering and Knowledge Engineering, Skokie, IL, USA.

Marwick, A. D. (2001). Knowledge management technology. *IBM Systems Journal, 40*(4), 814-830.

Mathi, K. (2004). *Key Success Factors for Knowledge Management.* Unpublished Master Thesis, University Of Applied Sciences/ Fh Kempten, Kempten, Germany.

McGarry, F., & Pajerski, R. (1990, November 1990). *Towards Understanding Software - 15 Years in the SEL.* Paper presented at the 15nd Annual Software Engineering Workshop, NASA Goddard Space Flight Center, Greenbelt, MD, USA.

McIllroy, M. D. (1968, 7th to 11th October 1968). *Mass-produced Software Components.* Paper presented at the NATO Conference on Software Engineering, Garmisch, Germany.

Mendonca, M., Seaman, C. B., Basili, V. R., & Kim, Y. M. (2001, June 2001). *A Prototype Experience management System for a Software Consukting Organization.* Paper presented at the 13th International Conference on Software Engineering and Knowledge Engineering, Ottawa, Canada.

Mertins, K. E. H., Peter (Ed.); Vorbeck, Jens (Ed.) (Ed.). (2003). *Knowledge Management. Concepts and Best Practices* (2nd ed.). Berlin: Springer-Verlag.

Monteiro, A., Almeida, A. B., Goulao, M., Abreu, F. B., & Sousa, P. (1999). A software defect report and tracking system in an intranet. *Proceedings of the Third European Conference on Software Maintenance and Reengineering, Amsterdam, Netherlands, 3 5 March 1999 * Los Alamitos, CA, USA: IEEE Comput. Soc, 1999, p 198 201.*

Morisio, M., Ezran, M., & Tully, C. (2002). Success and Failure Factors in Software Reuse (Vol. 28, pp. 340-357): IEEE Press.

Nicholls, D. (2004). Web site drives Gold Practice initiative. *IEEE Software, 21*(3), 108.

PPR. (2005). Portland Pattern Repository.   Retrieved 10. Oct., 2005, from http://c2.com/ppr/, http://en.wikipedia.org/wiki/Portland_Pattern_Repository

Prieto-Díaz, R. (1993). Status Report: Software Reusability. *IEEE Software, 10*(3), 361-366.

Prieto-Díaz, R., & Freeman, P. (1987). Classifying Software for Reusability. *IEEE Software, 4*(1), 6-16.

Ras, E., Avram, G., Waterson, P., & Weibelzahl, S. (2005). Using Weblogs for Knowledge Sharing and Learning in Information Spaces. *Journal of Universal Computer Science, 11*(3), 394-409.

Ravindran, S., & Sarkar, S. (2000). Incentives and mechanisms for intra-organizational knowledge sharing. *Proceedings of 2000 Information Resources Management Association International Conference, Anchorage, AK, USA, 21 24 May 2000 * Hershey, PA, USA: Idea Group Publishing, 2000, p 858.*

Riege, A. (2005). Three-dozen knowledge-sharing barriers managers must consider. *Journal of Knowledge Management, UK * vol 9 (2005), no 3, p 18 35, 80 refs.*

Rising, L. (2000). *The pattern almanac 2000.* Boston: Addison-Wesley.

Rombach, H. D., & Ulery, B. D. (1989, October 1989). *Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL.* Paper presented at the International Conference on Software Maintenance (ICSM 1989).

Roock, S., & Havenstein, A. (2002). *Refactoring Tags for automatic refactoring of framework dependent applications.* Paper presented at the Extreme Programming Conference XP 2002, Villasimius, Cagliari, Italy.

Schneider, K., & Schwinn, T. (2001). Maturing experience base concepts at DaimlerChrysler. *Software Process: Improvement and Practice, 6*(2), 85-96.

Scott, L., & Jeffrey, R. (2003). *The anatomy of an experience repository.* Paper presented at the International Symposium on Empirical Software Engineering, 2003 (ISESE 2003).

Senge, P. M. (1995). *The Fifth Discipline: The Art and Practice of the Learning Organization*: Currency/Doubleday.

Simons, C. L., Parmee, I. C., & Coward, P. D. (2003). 35 years on: to what extent has software engineering design achieved its goals? *IEE Proceedings Software, 150*(6), 337-350.

Sommerville, I., & Sawyer, P. (1997). *Requirements Engineering: A Good Practice Guide*: John Wiley & Sons, Inc. New York, NY, USA.

Sun, P. Y. T., & Scott, J. L. (2005). An investigation of barriers to knowledge transfer. *Journal of Knowledge Management, UK * vol 9 (2005), no 2, p 75 90, 45 refs.*

SWEBOK. (2004). Guide to the Software Engineering Body of Knowledge (SWEBOK, Ironman Version). Ironman. from http://www.swebok.org/

Thomas, B. D. (2006). *An Empirical Investigation Of Factors Promoting Knowledge Management System Success.* Unpublished PhD Thesis, Texas Tech University.