From Software Documents to Experience Knowledge Based Artifacts

Jouni K Kokkoneni and Lasse Harjumaa
Department of Information Processing Science
University of Oulu
P.O. BOX 3000, FIN-90014 University of Oulu, Finland
jouni.kokkoneni@oulu.fi, lasse.harjumaa@oulu.fi

Abstract

Acquisition, preservation, exploitation, and management of knowledge assets of an organization have become prominent tasks in order to make the software development more efficient. Documentation written during a development lifecycle can include more information than just project schedules, distinct requirements of software products or descriptions of product quality. Decisions, design rationales and other problem solving experiences are also implicitly documented to some extent in the documentation. This paper represents an approach for identifying this tacit knowledge and documenting it as experience knowledge based artifacts. We focus on knowledge that essentially originates from individual developers, and their individual human experiences. Documentation related to software inspections have been used as an example in our studies, but the approach can be generalized to all software development documentation. The findings presented in this paper are based on observations in two research projects in case companies.

1. Introduction

In addition to material assets, such as machines, equipments, and buildings, companies have also other important resources that are immaterial by nature. Immaterial capital has typically been classified as human capital, relational capital and structural capital. Immaterial capital can be considered equal to intellectual capital presented by Steward [23]. Steward categorized intellectual capital in two parts as Human capital and structural capital. Also Sveiby [24] has presented same kind of ideas. In software engineering, the role of these immaterial assets is even more important than in traditional industries, because software development is knowledge-intensive human work [7].

Rus and Lindvall [22], has addressed knowledge management in software management. Also for instance Alavi [1], Earl [8], Kankanhalli et al [11], King [12], and Yahua and Goh [27] have studied knowledge management. Knowledge that has been preserved within an organization has a significant role. Acquisition, packaging and making use of existing knowledge have become prominent tasks in software engineering. It is a great loss for an organization if an experienced employee be replaced by a newcomer for whatever reason.

Knowledge that is accumulated within an organization is traditionally referred as organizational memory. It is considered to originate from people, products, and databases that contain implicit or explicit information about the technical and social culture in the company. The amount of accumulated information is overwhelming and it is practically impossible to collect and record all the knowledge from an organization of any size.

This paper addresses issues concerning documentation of knowledge that essentially originates from individual human beings, their personal experiences and working experiences. Experts and experience has been studied for example Dooley et al [6]. We represent observations and experiences gained from software development companies in two previous research projects that focused on improving the efficiency of peer review and inspection processes. Projects have been presented more accurately in [15]. Among several research goals covered by these projects was to establish the relationship between quality assurance techniques and knowledge management. Our findings clearly suggested that software inspections provide an effective way for acquisition and reuse of experience knowledge. This prompted us to continue to study
knowledge management aspects of quality techniques further.

The software inspection process offers inherent linkage between the daily software development and quality assurance work as well as knowledge management. Inspection is an activity carried out in each development step. Recordings of experiences regarding specific aspects of development processes can be implemented as an integral part of the inspection process. Discussions and participants opinions in informal form can also be recorded during inspections. [26]

The research on knowledge management, in the context of software inspections, started on rather general level however during the course of our research more attention was paid to specific issues that link the knowledge firmly to quality assurance techniques. The focus was set especially on checklists, rules and improvement suggestions adopted from inspection process by Gilb and Graham [10]. We call these experience knowledge based artifacts. Experience knowledge based artifacts are documented tools for acquiring, storing, and transferring experiences of individual developers. We have also introduced a concept of experience knowledge that is related to the concept of organizational memory and is mainly based on human experiences [13]. A procedure and model describing how to generate artifacts, which cater for experiences and recognition, preservation and exploitation of the experience knowledge are presented in [15]. Experience knowledge based artifacts and procedure are also linked to the knowledge creation cycle presented by Tervonen and Kerola [25]. Recommendations on what issues should be paid attention to when generating artifacts are presented in [16]. We have also examined artifacts in the context of software development models in general, even though the main focus was on iterative software development models. This research and guidelines on how to use artifacts are reported in [17]. The research reported in [17] has been closely linked to software quality. The inspection process has been used as a primary quality assurance technique that links knowledge-related issues to development lifecycle in a logical and practical manner.

2. Immaterial capital

Assets of an organization can be divided into four groups; people, knowledge, reputation, and material assets. Material assets include items such as machinery, equipment, and buildings. People are the employees of company. Each individual employee may have knowledge that is vital for the company’s overall functioning. Knowledge itself, among other important factors has value for the organization [often significant economical value]. Reputation and company image are equally important factors. Reputation for example can have influence on the stock price of a particular company. It is a generally accepted fact that lost trust is difficult to regain. The same applies to impaired reputation or company image.

Human capital refers to the know-how of employees as well as abilities of individuals. Usually human capital is associated to tacit knowledge. Relational capital depicts the relationship between organization and its stakeholders. Among other things, brand names belong to this group and can be associated to interaction. Structural capital covers know-how that is attached to practices, activities and processes within an organization. These include for example patents as well as information stored in company’s information systems. Structural knowledge is associated to explicit knowledge. Figure 1 is synthesis from literature and depicts the issues that are linked to capital classes mentioned before.

![Figure 1. Immaterial capital.](image-url)
The transfer of software engineering know-how and best modeling experience knowledge is to enable the preservation of experiences. Certain documents serve better used in different phases of software development, can and experience knowledge based artifacts. Our research has addressed the usage, characteristics and lifecycles of software engineering documentation is merely a secondary function of those documents. Although practically all documents generated or accessed during a software development lifecycle are originally created for a specific purpose, such documentation can still be made use of after the original tasks have ended. Documentation is essential during the maintenance phase in particular. Re-engineering as well as further development of a product also requires retrieval of previous design rationales and decisions from the documentation. Potentially all documents may contain relevant experiences, experience knowledge as well as general software engineering knowledge in some unspecific form. This does not mean however that any document can be transformed into meaningful experience knowledge based artifact.

During our research on purposes and lifecycles of document artifacts, we have identified specific categories into which different types of documents belong to. “Managing the knowledge life cycle” has been studied earlier for instance by Birkinshaw and Sheenan [4]. The categorization may be done according to the purpose of a particular document or according to the document’s lifecycle. Table 1 represents the purpose-based categorization of software development documents. Advisory documentation refers to material that are provided to support individual developers to carry out the development activities. Decisive documents contain information that is used in decision making, such as resource allocation and deciding on design alternatives. Descriptive documentation is typically related to the product and validation and verification related documentation provides information for evaluating the quality of the software. Corrective

3. Experience knowledge

Experience knowledge is a concept derived from organizational memory [21] mainly based on human experiences [13]. Experience knowledge originates from the methods known as ‘knowledge management of software engineering lessons learned’ presented by Birk and Tausz [3], the ‘experience factory’ presented by Basili et al. [2], and the work of Nonaka and Takeuchi [18]. Our concept of experience knowledge concept as well as our overall approach have been experimented and evaluated during our previous research on software inspections in software companies ([15], [16]). Our observations suggest that the approach we have chosen is particularly useful in the alignment of the inspection process.

The need for a specific concept of experience knowledge also emerged during our previous studies of knowledge management in the inspection context [13]. The main feature of experience knowledge is to formally capture human experiences and enable learning based on them. Human experience may be defined as an event or occurrence that leaves an impression on one [5]. The definition is quite a general one and contains all possible experiences. On such general level, a particular experience transforms from an experience to experience knowledge once it has been acquired, stored and shared systematically.

Experience knowledge based artifacts are defined as entities that can preserve experiences. Usually these artifacts are created for purposes other than maintaining experiences. Certain artifacts that have been focused on our studies - such as checklists [14], software development rules [10], and software patterns [9] - each have a specific role and purpose in software development whereas capturing experience is merely a secondary function of those documents. Our research has addressed the usage, characteristics and lifecycles of software engineering documentation and experience knowledge based artifacts.

In our experience, nearly all types of documents, used in different phases of software development, can preserve experiences. Certain documents serve better however, in the knowledge package role [2].

One of the main objectives of defining and modeling experience knowledge is to enable the transfer of software engineering know-how and best practices from experienced developers to novices. An individual that has worked in a software development organization for several years would have acquired good professional skills regarding both product development and inspections. Such an individual would, in all probability, have encountered valuable experiences which, in written form could improve the effectiveness of less experienced developers as well as the competitiveness of the whole organization.

As a summary, experience knowledge aims at providing means for capturing and describing informal assets, such as skills of individual developers, practical observations of development work and accumulated know-how. Experiences develop into experience knowledge once they are identified, preserved and systematically made use in written form.

4. Document classification

Although practically all documents generated or accessed during a software development lifecycle are originally created for a specific purpose, such documentation can still be made use of after the original tasks have ended. Documentation is essential during the maintenance phase in particular. Re-engineering as well as further development of a product also requires retrieval of previous design rationales and decisions from the documentation. Potentially all documents may contain relevant experiences, experience knowledge as well as general software engineering knowledge in some unspecific form. This does not mean however that any document can be transformed into meaningful experience knowledge based artifact.

During our research on purposes and lifecycles of document artifacts, we have identified specific categories into which different types of documents belong to. “Managing the knowledge life cycle” has been studied earlier for instance by Birkinshaw and Sheenan [4]. The categorization may be done according to the purpose of a particular document or according to the document’s lifecycle. Table 1 represents the purpose-based categorization of software development documents. Advisory documentation refers to material that are provided to support individual developers to carry out the development activities. Decisive documents contain information that is used in decision making, such as resource allocation and deciding on design alternatives. Descriptive documentation is typically related to the product and validation and verification related documentation provides information for evaluating the quality of the software. Corrective
documents help in identifying shortcomings in processes and products and responding to the problems.

**Table 1.** Document classification by purpose

<table>
<thead>
<tr>
<th>Document type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Advisory documents</td>
<td>Training materials</td>
</tr>
<tr>
<td></td>
<td>Coding conventions</td>
</tr>
<tr>
<td></td>
<td>Document templates</td>
</tr>
<tr>
<td>Decisive documents</td>
<td>Project plans</td>
</tr>
<tr>
<td></td>
<td>Software patterns</td>
</tr>
<tr>
<td>Descriptive documents</td>
<td>Requirements specifications</td>
</tr>
<tr>
<td></td>
<td>Design documentation</td>
</tr>
<tr>
<td></td>
<td>Source code</td>
</tr>
<tr>
<td>Validation and verification</td>
<td>Checklists</td>
</tr>
<tr>
<td>documents</td>
<td>Test case descriptions</td>
</tr>
<tr>
<td>Corrective or reactive</td>
<td>Process improvement suggestions</td>
</tr>
<tr>
<td>documents</td>
<td>Bug reporting database</td>
</tr>
</tbody>
</table>

Documents and artifacts belonging to the advisory or verification and validation categories are typically produced by experienced developers and designers. Outside sources and expertise may have been used to enhance the contents of these documents. For example, checklists may have been derived from previous inspection data by experienced inspection moderators whereas coding conventions may originate from general programming literature and standards. These types of documents obviously contain knowledge that is based on previous experiences. Thus, they can be experience knowledge based artifacts.

Decisive and corrective documents are also potential experience knowledge based artifacts. This is because decisions related to the development of these documents should have been done by proficient individuals based on their expertise. Reactions to problems identified as well as any corrective actions can be seen as special cases of decision making. Documenting rationales for these decisions should also record the facts and knowledge related to the issue.

The category for descriptive documents refers to product documents, such as requirements specifications, software architecture specifications and source codes. These documents have been produced by individuals with different and possibly widely varying levels of expertise. It is possible that both experts and less experienced novices have taken part in the writing process. This does not mean that product documents do not contain useful knowledge although typically these types of documents are constrained to a specific context or application domain and focuses on unique solutions. Extracting ad hoc design solutions, for instance, into reusable design guidelines is likely to be labour intensive and may not even be particularly beneficial. Product documents also tend to provide input for other software development activities or processes. Conformance to requirements specification for instance is verified with checklists and development rules, and the source code is validated with test cases.

Even though some of the experience knowledge based artifacts can be obtained from outside the organization within they are used, they normally contain organization specific knowledge. Document templates relating to advisory document category for example can easily be obtained from software engineering textbooks. We have concluded however that templates that have been imported from literature are unlikely to work properly [16]. Even where the imported template appears to work well in its previous context, it should be modified to cover organization specific requirements. Development and maintenance of artifacts are important in order to sustain the benefits that templates and other artifacts can provide for the development work. During our research we have concluded that the lifecycle and evolution of document templates, in the context of knowledge management, are not quite like checklists or rules. Instead, they are typically imported from literature or other organizations and refined with organization specific know-how.

### 5. Artifact types and lifecycles

Typical lifecycles of experience knowledge based artifacts, such as checklists, rules, and improvement suggestions have been presented in [13]. Different types of lifecycles presented in [13] fit nearly all artifact types with small variation. Here, the artifacts are examined from behavioural viewpoint.

Two basic types of lifecycles can be identified for the documents that include reusable knowledge. The main differences between these two types concern the content, characteristics and usage of the artifacts. Descriptions of the basic lifecycles can be found in Table 2. It is our view that documents that validate or verify specific issues or those that act as advisory documents have a particular type of lifecycle that differs from the lifecycle of decisive or corrective documents. We call these alternative lifecycles Type 1 and Type 2. The descriptive document class has
been omitted in this approach because of the context-specific nature of product documentation.

**Table 2.** Document classification by lifecycle

<table>
<thead>
<tr>
<th>Lifecycle</th>
<th>Description</th>
<th>Document classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type 1</td>
<td>Iterative, evolving, connected</td>
<td>Validation and verification documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Advisory documents</td>
</tr>
<tr>
<td>Type 2</td>
<td>Linear, updated, independent</td>
<td>Decisive documents</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrective and reactive documents</td>
</tr>
</tbody>
</table>

The aspects of these categories will be examined further in the following sections.

**5.1. Type 1 artifact lifecycle**

Figure 2 illustrates the characteristics and evolution of artifacts with Type 1 lifecycle.

![Figure 2. An example of Type 1 artifacts](image)

In Figure 1, development rules are used to guide the software development. Rules can provide guidance regarding the product content. Rule can also carry out the procedures required to produce a particular product i.e. to be followed when implementing the product. Whether rules are strictly followed in implementation can be verified with checklists or validated with test cases. Rules, checklists and test cases all take the role of experience knowledge based artifacts. Rules and checklists should both be composed by experts with wide experience. Obviously, development rules alone are not adequate to implement a product, but other source material is required. For example, where the product to be implemented is a software design, requirements specification is needed in addition to the rules. Furthermore, checklists should not be the only source for verifying the design, as the rules and specifications in turn provide input for the verification.

Update and improvement of rules and checklists as well as all other documents, should be continuous process. Experts and experienced developers play a significant role in keeping the guidance updated. Checklists and rules should evolve over time. Thus, it is possible that checklists become obsolete or they develop into rules. This means that the original checklists may lose their meaning as such and completely new checklists, derived from the new set of rules may need to be compiled. The need to update checklists and rules usually emerges from improvement suggestions that are put forward during software development and peer reviews.

The evolution of the artifacts that have this type of lifecycle – i.e. checklists in particular - seems to be based on the usage and usefulness of the document content. It would appear that if a specific document helps developers in their work, such document is most likely to exist and improve over time. The evolution of individual items in checklists can be generalized to whole artifacts of this type. Figure 3 illustrates how upgrading needs of checklist items can be determined based on the usage of the items.

![Figure 3. Usefulness of checklist questions](image)

Where a question produces significant findings each time it is used in an inspection, it is possible that the question is either too strict or concerns slightly different aspect of implementation from the rule the question is supposed to relate to. The rule itself may be poor and in need as refinement. There may also be
inadequacies in the training or knowledge of the developers.

A particular question may also be a working one, such that produces findings at times. These questions tend to be the most beneficial ones and do not necessarily need to be altered. Significant findings tend to be caused by human errors or by inexperienced persons having been responsible for implementing the artifact.

Where a specific question in a checklist fails to produce any findings in inspections, it is likely that the question is futile. Rules may be oversimplified – i.e. always followed - or the question may concern a completely irrelevant issue, such as a wrong version of development tools for instance. Alternately the developers may be adequately skilled and may be non-existent problems with the validity of the implementation.

It would appear that checklists and rules are tightly connected. Thus, where the lifecycle of one of the Type 1 artifacts changes, it is likely to affect another artifact’s lifecycle. For example, if a particular development rule is removed, questions in the relevant checklist may need to be removed or the entire checklist may lose its meaning. Rules on the other hand may need to be removed when there is no inspection findings related to them. Similarly, where a rule is modified, question derived from it need to be modified accordingly, with the introduction of a new rule, new checklist items would need to be introduced. The relationship between artifacts is typically a two-way relationship. When rules change, checklists need to be changed and on the other hand, checklists may produce valuable input for generating new rules.

As a summary, Type 1 artifact lifecycle is typically iterative. Documents, such as checklists, are not produced for any specific occasion, but rather have a long-term role in the development process. They are upgraded in iterative manner and seldom disregarded completely. Instead, documents improve and evolve over time as a response to changing needs and feedback. Finally, Type 1 artifacts tend to be tightly connected to each other.

5.2. Type 2 artifact lifecycle

Figure 4 represents an example of Type 2 artifact characteristics.

Type 2 artifacts are typically related to the operation and management of the development process, such as process improvement documentation. They may also provide generic resolutions to problems, such as design patterns.

The lifecycle of Type 2 artifacts is different from the evolutionary nature of Type 1 artifacts. Improvement suggestions and design patterns typically arise from problems that are recognized to occur frequently in the development process or product implementation. Suggested solutions for these problems, when discovered, are documented once and perhaps reviewed and refined after that, but this type of documentation is not continuously updated. For example, if a company specific best practice is documented as a pattern, the pattern description is typically tried to accomplish into useful form at once, without iteration. The pattern is then used in the development until it is replaced with a more suitable pattern. Similarly, even though process improvement is a continuous task, documents such as improvement suggestions and process analysis data are created once, made use of when the improvement actions are planned and carried out, and possibly rejected after that.

These types of artifacts are essential in making decisions during the development. The Type 2 artifacts also record the decisions for subsequent analysis. These documents provide valuable information on solving problems regarding either the development process or specific design issues. Thus, these documents are definitely experience knowledge based artifacts. The most difficult part is to get the knowledge in writing. Best practices are a typical form of tacit knowledge, and motivating developers to write ‘the hints and wisdom’ they have into patterns may be challenging. The same applies to process improvement ideas and feedback. However, as long as the ideas and expertise is not in written form, it is not experience knowledge.
Figure 4 illustrates the utilization of patterns, but it applies as well to documented best practices, process documentation and bug databases, for example. The main idea is that Type 2 artifacts tackle shortages in processes and products and record reasoning that has been undertaken to improve the situation. The documentation can have implications on the product, such as pattern-based implementations, or on the process, such as process improvement actions based on the improvement suggestions. That way, the artifacts significantly contribute to process improvement and product quality. All artifacts in this category are based on the expertise of experienced developers and managers.

The lifecycle of a software pattern (or process pattern) is a typical example of Type 2 artifacts. A pattern usually provides a solution to a frequently encountered problem. The solution can concern code, or it can present a series of actions to be taken. Patterns can be public or organization specific.

When the pattern approach is used to solve a known problem, well-justified standard solutions can be achieved. Patterns are especially useful in transferring knowledge from experienced developers to novices. If there are no patterns for a frequent problem, a new pattern should be introduced.

A pattern does not become obsolete until there is a major change in the whole development context. That might be a switching to another programming language or change in the development lifecycle model, for example. Such changes may remove the problems, meaning that patterns are not needed. However, significant changes may introduce new problems and require new patterns.

The development process is working well, if the solutions provided by patterns are integrated into the process along time. For example, if there is a common problem related to development tool configuration and a documented way to solve the problem, the tool should be upgraded to enable easier configuration. After upgrading, the problem is solved permanently, and working practices have been improved by integrating the pattern solution to the process.

An important aspect of patterns is that they are used repeatedly in resolving similar types of problems. This requires that the pattern is documented in adequately general level. Still, the pattern should provide a practical solution. Thus, a pattern can solve rather different kind of problems as long as it has been documented properly.

As a summary, artifacts of Type 2, such as patterns, are useful as long as the problems they address exist. When these types of artifacts achieve certain maturity level to be practicable, they are not changed often. Documents that are created outside the organization are more general in this category than in Type 1 artifacts. These artifacts can also be directly composed for the knowledge management purposes. Patterns, for example attempt to document the best practices of everyday software development. Even though some of the knowledge may originate from outside of the company, these documents can clearly be treated as experience knowledge based artifacts. The usage of these artifacts is more linear than in the case of Type 1 artifacts. For example, a pattern is used to solve a problem, and when the problem has been solved, the pattern is not needed until the next similar problem arises. Updating of Type 2 artifacts is also different from Type 1 artifacts. While Type 1 artifacts are continuously improved, Type 2 artifacts are seldom changed after they have been published. Furthermore, these artifacts are not usually very closely linked to other artifacts, but they can be even transferred from one organization to another with some adjustments.

6. Conclusion

This paper has presented the importance of knowledge assets for organization operating in the current business environment. In order to continuously improve their effectiveness, software development organizations should make use of the immaterial capital they possess. We suggest that human experiences and experience knowledge, which are instances of tacit knowledge, should be extracted from software development documentation and utilized in later development work.

The results presented in this paper are based on the co-operation with partner companies during two previous research projects. The experienced personnel in the partner in particular companies have had a significant role in describing the lifecycles and usage of software documentation. We have presented arguments to support the view that it is reasonable and profitable to create a link from documentation, used in software product development, to experience knowledge. Secondly, we have presented the types of experience knowledge-based artifacts that can be derived from software development documents.

Our investigations demonstrate how different software development documents can act as experience knowledge based artifacts, although not all documents are equally useful as such. For example, documented development practices, design patterns and checklists are useful sources of expert views and experiences. Product documentation on the other hand does not usually provide as much general-level information that could be transferred into experience
knowledge. Yet, all artifacts are needed during development process, and hence should be continuously managed, updated and improved.

We have analyzed the ways in which different types of software documents can be used in development and which they can play in software development processes. We have observed that all documents can include experiences but only a part of the documents can act as experience knowledge based artifacts. Based on the earlier research, we have classified documents into five categories by the purpose and into three categories by the lifecycle of the document. These categorizations help in identifying experience knowledge based artifacts.

Because research based on two previous research project further research is needed on other types of software development documents that quality assurance related documents. In addition, we need to validate these results in other case companies.

7. References


