ABSTRACT
A reoccurring problem in software engineering constitutes ensuring sufficient completeness of requirements specifications with economically justifiable efforts. Formulating precise quality requirements and especially security requirements is elaborate as they depend on many stakeholders and technological aspects that are often unclear in early project phases. Threats that may have a severe impact on the software product are sometimes not even known. One approach to tackle this situation is reusing quality requirements, because they are to a high degree similar in different software products. The effect can be higher quality while at the same time saving time and budget.

Quality models are a way to explicitly specify quality. Based on activity-based quality models an approach for specifying reusable quality requirements in early project phases is proposed that also allows a direct derivation of suitable quality requirements for new projects. The applicability of this approach and the resulting reuse potential is investigated in a case study, which concentrates on the security requirements of six industrial projects.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specification; D.2.13 [Software Engineering]: Reusable Software—Reuse Models

General Terms
Security, Management

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...$5.00.

Keywords
Quality requirement, reuse, quality model, security

1. INTRODUCTION
Although it is a well-known fact that defects in requirements cost 10 to 200 times more once fielded [3], a high percentage of system malfunctions result of errors in early stages of development [21]. Yet, requirements are often poorly handled in practice. This is especially true for quality requirements. One of the most important reasons for neglecting quality requirements lies in the difficulty to formulate them precisely due to their wide variety of facets and aspects [16]. In particular, the more the quality requirements depend on technological aspects and volatile environments, the more elaborate their precise definition [18]. While requirements engineering gained attention over the last years, security requirements have traditionally been ignored in software engineering methodologies [18].

As an observable consequence, security requirements, for example, are often described implicitly as functional requirements (e.g., solutions in terms of protection mechanisms like a use case description that demands a login procedure) [11] rather then stating them as assessable quality requirements. This leads to the inability to assess the real security requirements especially since the original rationale is often not explicitly given.

A further obstacle in requirements engineering is the difficulty of refining high-level requirements to low-level, assessable requirements. A variety of quality engineering approaches try to tackle the problem, among them techniques that are based on meta-models [7,20], others that are structured as security guidelines [4, 10], and even approaches that integrate into existing security processes [17]. Further, many standards were developed to help in this situation (ISO 17799, 13335, 15408). However, these approaches are either hardly integrated in software engineering processes, do not provide methodological support, or are too focused on single quality attributes disregarding dependencies to other quality attributes, non-functional, and functional requirements. The need for applying proven requirements engineering (RE) methods in the field of security issues remains [1].

1.1 Problem Statement
Requirements engineering is facing two main problems regarding the handling of quality requirements and espe-
cially security requirements: First, there is no methodological guidance for refining high-level quality requirements to a concrete and assessable level. Second, it is often unclear how reuse of quality requirements can be properly integrated in the development process. It is economically questionable to write detailed quality requirements for each system from scratch.

1.2 Contribution

In this paper we show an approach that overcomes the mentioned problems of elaborating quality requirements by making use of activity-based quality models (ABQM) [7]. It allows a clearly structured specification of quality requirements that leads to (1) more normalized requirements for reuse, (2) requirements with an explicit rationale, and (3) clear refinement hierarchies. For an efficient and facilitated approach that supports reuse over large-scale projects we furthermore extend the ABQM with aspects that characterize projects according to project parameters and goals. This supports the integration of the ABQM into volatile project environments of individual companies.

Our approach is applied to six industrial projects in a case study. We show evidence on the applicability of the extended model as well as the reuse potential it offers for real-world requirements specifications using the example of security.

2. RELATED WORK

The research area of requirements engineering offers various approaches for quality requirements. Most approaches emphasize processes and techniques for eliciting, analyzing, and managing quality requirements. Such approaches (e.g., [5, 9, 19]) do not impose sufficient structure for managing and especially requiring reuse. Approaches like UMD [2] provide more structure or Haley et al. [14] support constructing satisfaction arguments, they do not support reuse directly. As the structure of quality requirements strongly depends on the understanding of the term “quality” and “quality requirements” itself [12, 16], one observable tendency is the use of quality models and guidelines. Both tackle the shortcoming as they describe structure and semantics of quality requirements and thereby the meaning of quality that is needed for an appropriate management of the requirements.

Guidelines include standards that were consolidated by a community, company, or organization and usually include technical aspects that developers have to take into account regarding quality, for example, the Common Criteria (CC) catalog [4] and the German BSI IT-Grundschutz Manual [10]. Usually guidelines do not motivate their demands and the given rationales for their mostly technical assertions are generic [7]. Hence, they do not guide through a structured process, are often read only once and followed in a sporadic manner. Furthermore, it is often not checked whether guidelines are followed at all [7].

Instead, quality models emphasize the semantics of quality and rely on a (semi-)formal model of the term “quality”, referred to as quality definition models [6]. One approach that concerns the elicitation of quality requirements using a quality model as backbone for reuse is the non-functional requirements method from Doerr et al. [8]. Another example is ATAM [15] that evaluates architectures and includes the definition of quality aspects by the use of “-ility”-trees. Still, ATAM is not intended for persisting the gained knowledge. Instead, the analysis is conducted separately for each new software architecture.

The approach described in this paper is based on activity-based quality models as proposed in [7]. In [22, 23] we described how to use these quality models for refining quality requirements and how to incorporate them into a general requirements process. In contrast, this paper investigates requirements reuse.

3. EXTENDED QUALITY MODEL

The approach presented in this paper represents an extension of the activity-based quality model (ABQM) proposed by Deissenboeck et al. [7] with the purpose of efficient reuse. The ABQM was initially designed to structure and formulate software quality. It allows the creation of a repository to persist knowledge concerning arbitrary quality attributes of software and software-intensive systems. However, the extension of the ABQM is performed to enable the re-appliance of knowledge among several software development projects. Reusability is achieved by relating elements of the model that describe requirements to characteristics of software products and processes, i.e., product and process properties, and goals.

3.1 Activity-Based Quality Model

Figure 1 illustrates the meta model of the ABQM. The model abstracts from activities, facts, and the dependencies between both, i.e., the impacts of facts on activities.

Figure 1: The Core ABQM Meta Model

Activities describe actions that can be performed on or with the support of the system under consideration as an outcome of the system’s quality. For instance, the activities related to “attacks” are influenced by the system’s security. Activities can be hierarchically composed and, thus, can be used to express common activity breakdown structures. An activity can be given in different shapes, such as use case or misuse cases.

Facts are characteristics of the situation the system is developed in. The situation may be decomposed to system properties such as “the existence of an empty line between each two code lines”, or it may contain organizational properties, such as the existence of security audits or the existence of a change management.

Facts and activities can be related to each other – a fact has a positive or negative impact onto an activity, depending on whether it positively or negatively contributes to the activity. Facts are further divided into an entity and an attribute supporting the composition of facts. For instance,
considering the fact “correctness of cookie”, the “cookie” that is described by this fact cannot be further split (e.g., into its specific contents) because of the attached attribute “correctness”. Hence, the ABQM defines facts to be divided into an entity enriched by an attribute. In the given example the fact is divided into the entity “cookie” and the attribute “correctness” so that the entity “cookie” can be further decomposed while remaining with the same attribute. We subsequently use the short-hand syntax [Entity|ATTRIBUTE] → [Activity]. This triple of entities, attributes and activities builds a quality element. For instance, the fact [Secure Attribute | CORRECTNESS] has a negative impact on the activity Modifying HTTP Cookies since it mitigates the attack.

Furthermore, facts also have an assessment type, which defines how a fact’s realization can be assessed: manually, semi-automatically, or automatically.

3.2 Extending the Model for Reuse

To effectively and efficiently support the reuse of requirements among differing volatile project environments, the ABQM needs a notion of projects and its goals and parameters that further characterize the project. Using this additional information, relevant quality elements can be selected for further projects. Although, quality elements could also be selected using the activity and fact tree, single projects cannot be efficiently characterized by the exclusive description of activities and facts.

Figure 3 illustrates therefore how the ABQM is extended by additional means needed for the purpose of reusing quality elements among different projects with individual characteristics. The resulting extended ABQM (eABQM) enriches the basic idea of the ABQM with subsequently described means.

**Figure 2: Two-Dimensional ABQM Example**

Since facts and activities are decomposed and have dependencies via the impacts, the ABQM can be arranged and persisted as a matrix. Figure 2 illustrates an exemplary matrix. The structure on the left side illustrates an excerpt of a facts tree, the structure on the upper part an activity tree. The two facts [Protocols | EXISTENCE] and [Restrictive Rights | EXISTENCE] represent for example the communication of a server. The activities show exemplary attacks on the envisioned system. The center of the figure illustrates the impacts. For instance, the facts [Protocols | EXISTENCE] and [Restrictive Rights | EXISTENCE] have both a negative impact on the activity Manipulate Data using own User Account.

Reusing requirements on the basis of facts and activities is economically reasonable for single projects or product lines where goals and project characteristics, such as the technical infrastructure, do not change. The ABQM allows the identification of conflicts between quality elements and provides the possibility to tackle software quality from different angles. It provides the benefit of an easy generation of checklists (lists with relevant facts selected by activities).

Yet, when reusing requirements the whole activity tree has to be investigated to identify relevant activities. This task is not only time consuming but also error-prone. Hence, the model lacks a condensed view on the activity tree that matches with the needs of individual projects.

**Figure 3: Extended Quality Meta-Model**

Projects are characterised by a set of goals and project parameters. Both are means that provide better search facilities to the repository and for retrieving a set of quality elements. When setting up a new project, it is infeasible to decide for suitable parameters, i.e., finding the right constraints and making the correct early design decisions. Having a repository with projects and their parameters allows taking known similar projects as basic templates. This prevents the project management from missing important parameters and shows alternatives.

Project parameters characterize in particular the environment of a project as well as describe characteristics of the product itself. They have a unique name with a description and may be of technical nature like “LDAP” or domain-driven like “public sector”. Project parameters are coupled to quality elements and enable different stakeholders to analyze the requirements for parameters for which they are responsible (having the best domain knowledge). We propose to persist the parameters using a tree data structure. Within such trees, we see category as an element that contains several hierarchically decomposed project parameters and a project parameter as the leaf that is not further decomposed. This distinction allows a clean data structure with individually (company-specific) chosen categories while
In order to efficiently filter and reuse a chosen set of quality elements, project parameters are coupled to the impact element by means of a relevance association and, thus, the parameters act as a filter on top of the ABQM repository. Having identified and selected project parameters, the association supplies all corresponding relevant quality elements since a fact’s relevance implicates the relevance of the activity linked with an impact. Because several quality elements can be relevant for a project parameter and one quality element can be linked by many parameters, the relevance association includes a description of the rationale.

Having an ABQM repository and project parameters, it is easy to infer a set of related quality elements. For example, we might infer the quality element advising to encrypt the database. However, what if our application will not be critical in security at all? What if security is not a project goal?

In addition to parameters as fixed characteristics of a product or its development environment, we provide means of goals that represent prescriptive statements of intents [13], i.e., mostly product-related objectives that shall be reached (often referred to as business requirements). For instance, while the use of “LDAP” is a project parameter (a concrete design decision), “security” would represent a goal that shall be supported by the product’s characteristics. Parameters indirectly support goals via the connected quality elements as parameters can be inferred from goals during the requirements elicitation phase. Because goals abstract from activities, an activity can support or avert a goal. The influence edge needs to have a description to identify and fix subjective relations. An activity can influence an arbitrary number of goals and one goal can be influenced by several activities.

For example, the activity DELIBERATE ACTION hinders the goal SECURE DATABASE. Now it is possible to filter quality elements in a second dimension. The parameter DATABASES without the goal SECURE DATABASE does not result in the requirement [Database ENCRYPTION]. To be more precise, it is possible to specify sub goals. The high level goal SECURITY can be divided into BASIC SECURITY and SECURE HIGHLY CRITICAL DATA. The fact [Database ENCRYPTION] is relevant for highly critical data, but not for BASIC SECURITY.

Finally, a requirement is derived from one quality element and combines one fact with one activity. The statement of a requirement (its description) is given by the fact, the rationale is given by the activity and the impact. A requirement differs from a quality element in its additional project-specific attributes. Requirements are, e.g., prioritized and have assigned states. A quality element rather provides the initial shape of a requirement with its very basic data, the text and the rationale. Each quality element that is selected for one project can therefore be homogeneously mapped to exactly one requirement. Table 1 illustrates an exemplary requirement with corresponding attributes as it can be inferred from a quality element for the above encrypted database example.

Furthermore, since quality elements that are contained in the EABQM repository shall be reused within projects that may create a product with a completely different functionality, quality elements have to be formulated generically to be reusable. Project-specific instances of these quality elements with concrete values like a specific amount of maximally expected workloads must not be persisted in the EABQM.

<table>
<thead>
<tr>
<th>ID</th>
<th>Text</th>
<th>Rationale</th>
<th>Stakeholder</th>
<th>State</th>
<th>Owner</th>
<th>Source</th>
<th>Acceptance</th>
<th>Criteria</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQ-01</td>
<td>The database has to be encrypted.</td>
<td>Users should be prevented from reading data by bypassing the application.</td>
<td>N/A</td>
<td>Initial</td>
<td>N/A</td>
<td>ABQM</td>
<td>Manual review</td>
<td></td>
</tr>
</tbody>
</table>

In summary, the extension of the activity-based quality model enables easy access and filtering of facts, activities, and therefore complete quality elements. Filtering is done using project parameters and goals that are often a result of project acquisition. Therefore, it is possible to infer relevant requirements regarding quality of software in early stages of a project, which is important for correct estimation of efforts. The reuse of quality elements (i.e., requirements) using the EABQM repository further supports the inference of high quality requirements as well as enables the early detection of conflicts and trade-offs.

4. CASE STUDY

The case study validates the EABQM approach on real-world projects from a software consulting company. We first define the study in detail and derive two research questions that the study aims to answer. Then the study design used to reach these aims is described. The study implementation and execution are documented next. Finally, results and threats to validity are given and discussed.

4.1 Study Definition

The above introduced EABQM approach needs to be evaluated in order to validate whether it is useful in a realistic environment. In this paper, we validate our approach for security requirements. For this, we are mostly interested whether the approach is applicable to real security requirements and whether there is a reuse potential that can be exploited by the approach. Therefore, industrial projects are investigated from the point of view of the requirements engineer. First, we derive two explicit research questions.

RQ 1. Can real-world security requirements specifications be modeled with the EABQM approach?

This question investigates the applicability or suitability of the approach w.r.t. security requirements that are needed in practice. An approach for reusing security requirements can only make sense if it is able to model the needed aspects of quality that occur in real development projects.

RQ 2. How much reuse is possible using the EABQM approach?

The second question quantifies what potential there is in real-world projects for the approach. If the security requirements differ almost completely in practice or the similarities cannot be exploited by our approach, the return-on-investment of using it will probably be negative.
4.2 Study Design

The study is planned with 3 phases of which the last 2 phases are repeated for all but the first project. It is designed to be conducted at a single company. The idea is to “simulate” what would have been possible if the eABQM approach was used. Therefore, the oldest project is used first in the training phase in order to build up the eABQM repository. For this, the goals, parameters and requirements are modeled using eABQM. In phase 2, the next project is analyzed for its goals and parameters. Based on this, corresponding requirements are inferred from the eABQM repository. These inferred requirements are then compared to the specified requirements from the project documentation. This gives an indication for the reuse potential (RQ2). In phase 3, the eABQM repository is complemented with additional requirements from the existing requirements. Then, phase 2 and 3 are repeated for the next project until all projects are investigated. This approach is depicted in Figure 4.

![Figure 4: 3 Phase Study Design](image)

This design allows us to answer the two research questions defined in Section 4.1. RQ 1 considers the applicability of the eABQM approach. This is investigated in phase 1 and phase 3. We check whether all existing security requirements can be modeled using eABQM. To operationalize this, we measure the abstraction ratio, which is the number of requirements that have to be abstracted for use with the eABQM by the number of total requirements per project.

RQ 2 that considers the reuse potential provided by the eABQM approach is investigated in phase 2. A comparison between the inferred requirements and the existing requirements yields the reuse ratio. The ratio is represented by the number of inferred security requirements that are contained in the existing requirements by the total number of existing requirements. It indicates the amount of work that could have been saved by the eABQM approach.

4.3 Study Implementation

The complete case study is conducted at the software consulting company Capgemini sd&m. Since the company works with different customers in projects within a variety of domains it is well-suited as a case study. We analyzed first the internal project database of Capgemini sd&m for suitable projects with extensive specifications of security requirements. The contact persons from 20 projects were interviewed. From this analysis, we found 6 projects (A-F) that were suitable for the case study. The sizes of the selected projects range from one person month to 333 person years, the average size was 163 person years. The requirements specifications and related documents of all these projects were available. Project A was identified as the oldest and was hence chosen to be the training project. As guideline for modeling the security requirements the ABQM on security [23] was used. It provides a basic structure on how security requirements can be modeled.

For the ABQM, a tool is available that was presented in [7]. The tool does not support the extension, yet. However, as only specifications of 6 projects were analyzed, parameters and goals were documented separately. Parameters were taken from a project database that is maintained by Capgemini sd&m. Additional parameters were manually inferred from existing requirements. The need for additional parameters became obvious whenever security requirements could not be related to any parameter.

The derivation of the two ratios needed for answering the research question was straightforward as all security requirements specifications had clearly distinct and numbered requirements.

4.4 Study Execution

The initial training project (project A) resides in the public sector and therefore the developers had to use the German security standard IT-Grundschutz manual. We mapped the resulting requirements to activities and facts from the ABQM. Goals and parameters were inferred from the specification documents and in interviews with contact persons.

According to the study design, we identified parameters and goals for project B and used the repository to infer relevant security requirements (i.e., facts and activities). For these two projects, we showed the applicability views on the eABQM repository, such as project parameters and goals.

As described in the study design, phase 2 and 3 were repeated for the remaining projects C–F. Thereby we did not come up with goals and parameters, but compared the corresponding requirements from each project with the whole requirements base from the ABQM repository in detail. This allows a more elaborated analysis of the reusability of security requirements that were modeled using the ABQM.

4.5 Results

The results of the case study are shown in Table 2. The column # Sec. Reqs gives the number of security requirements modeled in the eABQM. Abstr. Ratio the share of abstracted requirements of the total security requirements and Reuse Ratio the share that could have been reused. The range of the size of the requirements is high, which stems from the diversity of the analyzed systems and their specifications. On average more than 100 security requirements were specified per project.

For RQ 1, the abstraction ratio, we found that on average 23% of the security requirements needed to be abstracted to be included in the ABQM. This means that 77% could be directly modeled. The abstraction was necessary mostly because of project-specific thresholds for metrics (see also Section 3.2). For instance, the requirement “The session length is restricted to 10 seconds” had to be abstracted to “Sessions must include a time-out.” since this kind of information is hard to reuse as it strongly depends on the
concrete application. The range of the abstraction ratio is high with 4%–70%. However, project B is an outlier as the second highest value is only 20%.

For RQ 2, the reuse potential, we found that on average 47% of the security requirements could have been reused using the eABQM approach. The range goes from 0.18 up to 0.87. The ratio, however, depends strongly on the size of the specification. In smaller specifications, it is by far easier to come to a high reuse ratio.

### 4.6 Discussion

With an abstraction ratio of 23% we can answer RQ1 positively. More than two thirds of the requirements could be directly modeled as requirements or goals in our approach. This is an indication that the levels of abstraction of practical security requirements and the ABQM are close. Only project B had a much higher abstraction ratio of 70%. We suspect the reason to be that in this specification the requirements are exceptionally detailed. Hence, their reuse potential is low if no abstraction is originally made. Treating project B as an outlier would reduce the average abstraction ratio to 12%.

Regarding RQ2 the results are promising, too. With an average reuse ratio of 47%, almost every second specified requirement could have been reused employing our approach. As we have no baseline of reuse that would be possible without the eABQM approach, we cannot give an improvement caused by the approach but only show its potential. Among other factors, the size of the specification has a large effect on the reuse ratio. Obviously, the eABQM repository can only deliver as many requirements as are contained in it. Hence, the reuse potential of a large specification seems to be lower. Another effect, especially found in project F, is that larger specifications tend to have more redundancies. Because of the fixed structure of requirements in the ABQM, these redundancies can largely be reduced. The highly redundant requirements cannot be inferred from the repository.

Beyond our research questions, we found several interesting aspects related to our approach. First, in most cases we found additional requirements by derivation from the repository that were not contained in the actual requirements specification.

Furthermore, we see an improvement of the specification of individual requirements. As mentioned, the structure of the eABQM prescribes a uniform way to document each requirement. This has two major effects: (1) avoidance of redundancy and (2) explicit rationales. The clear structure prevents redundant or similar parts in general. Hence, these requirements can be abstracted and reused. Moreover, the explicit impact on an activity specifies a rationale of the requirement. It shows an explicit effect on a product's security. This improves the overall requirements quality as well.

Finally, we observed that the application of the eABQM approach improved quickly while more and more projects were included in the repository. Then the identification of requirements became fast. Hence, it seems to be adequate for usage during acquisition or similar situations. The gains in speed were achieved by providing a condensed access to quality elements, i.e. via parameter and goal trees. On the long run, such a repository promises to become a central quality knowledge base for a company.

### 4.7 Threats to Validity

We measure the applicability of our approach by the ratio of requirements that could be directly modeled and the reuse potential by the ratio of requirements that could have been reused. For the applicability, this could be only part of the answer to the question as it would be also interesting to investigate how well practical requirements engineers can understand and benefit from the approach.

Since we as researchers filled the eABQM repository, it is possible that practitioner would have modeled the security requirements differently as we did. We mitigated this threat by dividing the specifications among all researchers and developed the eABQM models independently. Some of the researchers had little prior knowledge of ABQM and hence are suitable substitutes for practitioners.

A major threat to the generalization of the study is that all considered projects were conducted at the same company. Hence, it is unclear whether comparable results will be found in other environments. However, all projects had distinct project teams and different customers. Furthermore, one practical scenario for the eABQM is that it is applied inside a single company. Moreover, we concentrated on security requirements. It is not entirely clear whether the same results could have been achieved with other types of quality requirements. However, Basili et al. [2] argue that at least dependability requirements can be treated with the same approach.

### 5. CONCLUSIONS

In this paper we tackled the problems of an efficient refinement of security requirements and their reuse in software projects. Therefore, we proposed an extended quality modeling framework that allows the efficient reuse of quality requirements. To this end, we extended an existing quality model by means to describe quality requirements explicitly along with project characteristics and goals. Those means can be used to select a subset of requirements from the repository that is relevant for a new project. Using a case study, we showed the applicability and efficiency of the approach regarding security requirements as well as the increased quality of requirements and thus provided the basis for engineering high-quality software.

The knowledge base allows a company to persist highly valuable knowledge about the hard to grasp quality of software. Using constraints from the customer as project parameters and goals enables responses to customer needs in early project phases. Thereby, more elaborate time estimations based on persisted knowledge and an early satisfaction of customer needs by presenting specific, detailed know-how become available.

The case study helped adapting the model to industry needs while at the same time evaluating the efficiency of the
approach. Furthermore, it allowed introducing the model into the company and make preparations for an implementation of the approach within the company. Last but not least, the case study helped revealing relevant future research needs regarding the quality model and reuse of quality requirements, among them the following.

The concept of relating parameters could be more powerful while not loosing its ease of use. For instance, a parameter X could be modeled to imply a quality requirement only if another parameter Y is present, too. To this end, parameter sets could be introduced. However, this would significantly increase the complexity of modeling project parameters in the EABQM and thus should be carefully investigated within industry. Furthermore, the impact, relevance, and support relations are at present not assigned with a value. Instead, this problem is evaded by relating relevant sub-trees with each other. For example, instead of linking the goal SECURITY with assigned value x because security is only necessary to a specific extent, one has to link or select the relevant sub-goals of security. However, it could be valuable to introduce some kind of well-defined scale to further improve the handling with the EABQM repository. The conducted case study revealed the need for more assistance while identifying facts and activities for projects that shall be persisted in the EABQM. Not shown in this paper is the integration of this approach with the quality engineering and requirements engineering processes. This will be evaluated in further case studies and has further potentials of making the quality requirements process more efficient.

The EABQM approach is also further developed in the project Quamocino in which alongside Capgemini sdkm the industrial partners SAP, Siemens and itestra will evaluate the approach for their quality requirements needs.

Acknowledgments

We gratefully acknowledge the effort and cooperation of all employees of Capgemini sdkm that participated in the interviews and feedback meetings.

6. REFERENCES


