Object-relational Mapping Revised - A Guideline Review and Consolidation

Martin Lorenz[1], Günther Hesse[1] and Jan-Peer Rudolph[2]
[1]*Hasso-Plattner-Institut, University of Potsdam, August-Bebel-Str. 88, Potsdam, Germany*
[2]*LZN Laser Zentrum Nord GmbH, Am Schleusengraben 14, Hamburg, Germany*

Keywords: Object-relational Mapping, O/R Middleware, Enterprise Patterns, Data Model Design.

Abstract: Object-relational mapping (ORM) is a mechanism to link classes of an object-oriented (OO) programming language to tables of a relational database management system (RDBMS). When designing a mapping for an application's domain model, different strategies exist to map associations and inheritance relationships to database tables. Each strategy has a different impact on the application's quality characteristics. Developers need to understand the impact of a mapping strategy to make informed decisions. In the absence of cost models to quantify the impact, guidelines and best practices have been developed to allow differentiated considerations of strategies. However, looking closer at these guidelines, two major flaws become apparent - incompleteness and inconsistency.

In this paper, a comprehensive literature study is presented, which includes an analysis of guidelines and best practices from industry and academia. We propose a consolidation approach, which identifies relevant aspects of mapping strategies that impact a system's quality characteristics. The approach derives a multi-level organization, which describes the relation between mapping strategy aspects and quality characteristics of a system. The identified mapping aspects and the organization can serve as a framework to improve existing guidelines and to resolve inconsistencies.

1 INTRODUCTION

Using relational database management systems to persist objects of an OO domain model is a common approach. Object-relational (O/R) mapping represents a mechanism to bridge the semantic gap between OO programming languages and RDBMS. Due to fundamentally different concepts of relational algebra and the OO programming paradigm, a number of challenges exist, when mapping objects to tables. A particularly interesting challenge is the mapping of class inheritance structures to database tables. From a functional point of view, multiple correct approaches exist that map objects of a class hierarchy to database tables. These different approaches are commonly referred to as mapping *strategies*. Each strategy has a different impact on the application's non-functional characteristics, i.e., its quality. For the remainder of this paper, the term non-functional characteristics and quality characteristics are used synonymously. The semantics of the different strategies have been defined and documented in a number of publications (Ambler, 2003; Fowler, 2002; Keller, 1997; Holder et al., 2008) and are widely accepted. The challenge for a developer is to decide what mapping strategy is optimal with respect to a set of non-functional system characteristics, e.g., maintainability, efficiency, usability. Today, there exist many ORM frameworks, such as Hibernate[1], Doctrine[2] or SQLAlchemy[3], which provide implementations of mapping functionality for various OO programming languages, but they do not choose a mapping strategy for class inheritance structures. In general, a framework is preconfigured to apply a single mapping strategy to all inheritance hierarchies, defined in the domain model. It then provides configuration options that developers may use to control the application of mapping strategies for different inheritance hierarchies in the domain modal. A mapping framework does not provide any intelligence or automation approach that would suggest a mapping strategy for a particular hierarchy. It is the developer that has to decide what mapping strategy fits best to

[1]http://hibernate.org/
[2]http://www.doctrine-project.org/projects/orm.html
[3]http://www.sqlalchemy.org/

Lorenz, M., Hesse, G. and Rudolph, J-P.
Object-relational Mapping Revised - A Guideline Review and Consolidation.
DOI: 10.5220/0005974201570168
In *Proceedings of the 11th International Joint Conference on Software Technologies (ICSOFT 2016) - Volume 1: ICSOFT-EA*, pages 157-168
ISBN: 978-989-758-194-6

the desired non-functional characteristics of the system. In the absence of cost models to quantify the impact of a mapping strategy, developers rely on guidelines and best practices in their decision making process. However, looking closer at these guidelines, two flaws become apparent - incompleteness and inconsistency.

In this paper, we present a literature study, which compares and consolidates best practices and guidelines that are supposed to aid developers in the decision making process. The selection of documents comprises of academic publications, text books and technical documentations of vendors of O/R mapping frameworks. Our analysis reveals that the aspects of the mapping process, addressed by the different authors, vary greatly. The individual selection of mapping aspects is arbitrary and depends largely on the author's background and the scope of the document. This presents a problem for software developers, who seek for an answer to the question what mapping strategy to implement. Based on our analysis, we propose a consolidation approach to identify inconsistencies and missing parts, to allow a reasonable comparison of existing works. We provide a comprehensive evaluation that elaborates on the identified insufficiencies and provides reasonable explanations, why they can be observed. One result of our consolidation is a comprehensive set of mapping aspects. This set of aspects can be applied to a number of purposes. First, it can serve as a frame of reference for evaluations, discussions or comparisons of O/R mapping strategies. Second, it can help software developers understand the impact of mapping strategy aspects on the system's quality. This allows to make distinctive decisions regarding the use of a mapping strategy. Furthermore, it can serve as a reference for further research in the area of ORM concepts.

The remainder of this paper is structured as follows: Section 2 explains our document selection process. The consolidation approach is described in Section 3. The results of the consolidation are presented in Section 4. Section 5 contains the evaluation of our results. Section 6 summarizes and concludes the paper and Section 7 gives an outlook to future work.

2 PAPER SELECTION PROCESS

A vital part of our literature study is the selection of relevant documents. In Section 2.1, we describe our search approach and the selection criteria that have been applied to identify a set of relevant documents. Section 2.2, gives a brief introduction to the selected documents.

2.1 The Search Process

For the identification of potentially relevant documents, we followed the systematic literature review (SLR) strategy, proposed by Kitchenham and Charters (Kitchenham and Charters, 2007). We started our document selection process by searching for publications about non-functional characteristics of inheritance mapping strategies. The initial search was conducted using ACM's digital library, the online archive of the digital library of IEEE Computer Society, Google Scholar, and Microsoft Academic Search. We included the terms: "object", "relational", "mapping", and "non-functional characteristics", which led us to a number of potentially relevant publications. Based on references and citations, we employed a forward/backward reference search to identify further documents. For a detailed analysis, we selected those documents that contain explicit discussions about the impact of mapping strategies on system quality characteristics. Holder et al. (Holder et al., 2008) describe the most structured analysis of quality characteristics in the context of object relational mappings. Ambler (Ambler, 2003), Fowler (Fowler, 2002), Keller (Keller, 1997) and Philippi (Philippi, 2005) provide information about the behavior of inheritance mapping strategies. These five documents represent the core of our literature study. Additionally to research papers and textbooks, we decided to look into technical documentations of O/R mapping frameworks, e.g., Hibernate. We believe, a lot of software developers start looking for guidance regarding their mapping design in such documentations. Mostly using Google, we searched for ORM tools for various object-oriented programming languages. A brief introduction to the selected documents is given in Section 2.2.

2.2 Chosen Documents

We started out, by looking at the textbooks of Fowler (Fowler, 2002) and Ambler (Ambler, 2003). Both are focused on the conceptual level and explain the semantics of the three basic strategies, namely *Single-Table Inheritance*, *Table-per-Class Inheritance* and *Table-per-Concrete-Class Inheritance* (Fowler, 2002). Since the terminology is not consistent in the literature, the strategies are referred to by different names in some publications, more on this follows in Section 3.1. Fowler gives a short recommendation for use and an implementation example for each strategy. Ambler explains the strategies based on an example hierarchy and lists advantages and disadvantages.

Philippi (Philippi, 2005) suggests a model driven approach to automatically generate object-relational mappings based on non-functional software requirements. Therefore, the different inheritance mapping strategies get a fixed rating with respect to their understandability, maintainability, storage space consumption and performance. Based on trade levels, which express the requirements of the user, a strategy is chosen accordingly. Thereby, Philippi (Philippi, 2005) extends previous approaches of automatic mapping generation. He builds on the idea of Keller et al. (Keller et al., 1993), which only supports one inheritance mapping strategy for the entire class hierarchy. Hybrid designs are not considered. Philippi's rating of the strategies has the three levels positive (+), moderate (o) and negative (-). Especially for understandability and maintainability, the reasoning or metrics, which build the basis for the rating, are not clearly described. The number of joins builds the measure for performance, and the number of NULL values and redundancy for storage space. Additional influencing factors like inheritance depth or occurring query patterns are not considered.

Holder et al. (Holder et al., 2008) suggest a metric suite for object-relational mappings. Based on a number of software quality characteristics that are influenced by underlying object-relational mappings the authors suggest four metrics. In their future work, the authors consider a normalization of the metrics to enable an automated selection of mapping strategies. However, a cost model, based on these metrics, has not yet been presented. Holder is more precise than Philippi in the consideration of the influenced non-functional characteristics. Especially the mentioned quality characteristics are a good starting point for the analysis of the non-functional requirements in Section 3.2.

Keller (Keller, 1997) describes several patterns to map objects to tables on a structural level. This includes an explanation of the three basic inheritance mapping strategies with an example. In addition, Keller considers the consequences of the chosen strategy on the characteristics of an application, for example query performance and maintainability.

The examination of ORM tool or framework documentations shows how the industry deals with the topic of strategy selection. We analyzed 18 current ORM tools with respect to the support of the three well-known inheritance strategies. In general the tool documentations rarely give recommendations for use or information about the impact of the strategies on non-functional software characteristics like performance or maintainability. The reference of OpenJPA (Apache Software Foundation, 2013) lists some advantages and disadvantages per strategy, which mainly relate to performance and space consumption. Apache Torque's (Apache Software Foundation, 2012) and Propel's (Propel Community, 2015) documentation make short statements about the performance of the strategies. The reference of Doctrine (Doctrine Team, 2015) contains considerations about the impact of the two implemented strategies on performance, design-time and the database schema. In general, one can say that technical documentations of O/R mapping frameworks focus on explaining functionality and configuration. The impact and consequences of choosing a particular mapping strategy are widely discarded by the analyzed documentations.

3 THE CONSOLIDATION APPROACH

As introduced in Section 1, the aim of this paper is to identify inconsistencies and incompleteness in existing guidelines and best practices for choosing an optimal mapping strategy. To achieve this goal, we use the method of consolidation as a process to compare and combine the information and discussions from the different sources.

We identified two challenges in this process. The first is a lack of a standardized terminology. This is problematic, because it is difficult to compare argumentations and discussions from different authors, when they do not use a common vocabulary and definitions. The second challenge is the different levels of abstraction that are used to discuss aspects and impact of a mapping strategy. A high level of abstraction leaves room for interpretation. This is problematic, because the risk of misunderstandings or misinterpretation is high. To address these two challenges, we propose a three-step consolidation process. The first step is a consolidation of terminology. The second step is a consolidation of mapping aspects. The third step proposes an organization of mapping aspects that allows to understand the relationship between functional requirements and non-functional system characteristics. It also visualizes the different levels of abstraction involved. That way, we are able to follow the argumentation and reasoning of different authors and we reduce the room for interpretation.

3.1 Step 1: Terminology Consolidation

Analyzing the documents presented in Section 2, it becomes obvious that there exists no commonly accepted terminology. Across all papers, we observe

a variety of synonyms that are used to describe the same concepts. We identified this fact as a problem, because ambiguity is a possible source of misunderstandings and eventually may lead to inconsistencies. To perform a comprehensive and thorough consolidation, we need to define a basic set of terms, which represent a core set of vocabulary. We are convinced that all authors have considered their vocabulary carefully, to support their argumentations and reasoning. However, we observed that most authors omit the introduction of semantics regarding their key vocabulary. By giving a semantic definition for terms that we consider key to understanding our arguments, we want to reduce the risk of ambiguity and we allow to relate argumentations from other authors to our work. Fortunately, there exists a standard (ISO/IEC 9075) for the Structured Query Language (SQL) (ISO, 2011), which provides a comprehensive glossary for database artifacts and concepts. The most current version of that standard is ISO/IEC 9075:2011. Any database related terminology used in this document is based on that standard.

- **O/R Mapping** - or short ORM is an umbrella term that describes the concepts and processes of linking classes of an OO programming language to tables of an RDBMS. It resembles conceptual considerations rather than practical implementations. Often the term is used synonymously for O/R middleware implementations such as Hibernate. In our discussions, we refer to ORM only when we mean conceptual consideration.
- **O/R Middleware** - is a technical implementation of the process of linking classes of an OO programming language to tables of an RDBMS. We refer to O/R middleware, when we talk about the manifestation of mapping concepts or strategies in coding artifacts. This includes full-fletched frameworks, such a Hibernate or Doctrine as well as custom coding that implements O/R mapping functionality.
- **Entity** - is a conceptual component of the application. It is represented as a class definition that captures the entity's attributes, which are relevant for the application. An example for an entity could be a customer or a product.
- **Object Model** - is the collection of all class definitions of the respective application. A synonym would be domain modal or business modal. It comprises, but is not limited to all entities the application considers.
- **Data Model** - is the SQL database schema, which is used to persist objects of the object model.
- **Mapping Strategy** - is a concrete design decision how to map an inheritance relationship between two or more classes to one or more relational database table(s). A single strategy does not necessarily need to be applied to the complete hierarchy. Within the tree of classes of an inheritance hierarchy, each branch can be implemented using a different strategy. In that, we talk about a hybrid approach. Figure 1 illustrates such hybrid approach for a hierarchy of three classes, where the relationship between *Employee* and *ExternalEmployee* is implemented using the *Single-Table Strategy* and the relationship between *Employee* and *InternalEmployee* using the *Table-per-Class Strategy*.
- **Mapping Specification** - is the overall mapping design for a complete inheritance hierarchy. What *strategy* is implemented for which branch of the hierarchy is defined by the *mapping specification*. Figure 1 explains the relationship between *mapping strategy* and *mapping specification*.
- **Single-Table Strategy** - this strategy maps all classes of a branch from an inheritance hierarchy, which follows this strategy, to the same relational table. It requires an additional relational field in the shared table, which indicates the class of each row. Single Table Inheritance is also known as *One Inheritance Tree - One Table* (Keller, 1997) or *Union Superclass* (Holder et al., 2008). For the remainder of this paper, we abbreviate it with *ST* (Single Table).
- **Table-per-Class Strategy** - this strategy implements a one-to-one mapping between classes and tables. The corresponding table of a class contains a relational field for each non-inherited class attribute. Thus, object persistence is achieved by distributing object data over multiple tables. In order to link these tables, all tables share the same primary key. In addition, the primary keys are also foreign keys that mimic the inheritance relationships of the object schema [9]. Each row in a table then, maps to objects in that tables corresponding class and subclasses. It is also known as *One Class One Table* (Keller, 1997), *Joined Table Inheritance* (Apache Software Foundation, 2013) or *Joined Subclass* (Holder et al., 2008). For the remainder of this paper, we abbreviate it with *TPC* (Table-per-Class).
- **Table-per-Concrete-Class Strategy** - this mapping strategy only maps each concrete class to a table. Thereby, all inherited and non-inherited attributes of a class are mapped to the same table. Each table then only contains instances of its cor-

responding concrete class. The strategy is also known as *One Inheritance Path One Table* (Keller, 1997) or *Union Subclass* (Holder et al., 2008). Philippi (Philippi, 2005) presents several variants of the strategy. We focus on the most common form, which is referred to as *One flattened table for each concrete class*. Some ORM documentations (e.g. OpenJPA) use the name *Table Per Class Mapping* (Apache Software Foundation, 2013), which can easily be taken for *Class Table Inheritance*. For the remainder of this paper, we abbreviate it with *TPCC* (Table-per-Concrete-Class).

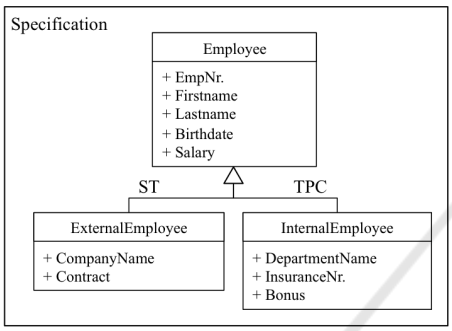

Figure 1: Example of classes used in section 3.2.

3.2 Step 2: Mapping Aspect Consolidation

With a clear set of vocabulary, we are now able to consolidate the mapping aspects (MA) discussed in the analyzed set of documents. In the first part of this step, we collect all MA from the analyzed literature to guarantee that our list of MA is complete with respect to these documents. A large part of the collected MA reflects functional system requirements. The following enumeration lists the collected requirements and explains their semantics. For a better understanding, we provide examples, based on the class inheritance structure, depicted in Figure 1. For reason of compactness, we will not interpret the semantics for every mapping strategy. We believe our descriptions to be detailed enough to derive the respective differences for each strategy.

- **MA1** - Create new object: A new object has to be persisted into the database. The characteristic "new" means that the object refers to an entity, whose logical identity has not yet been stored. All attributes of the object have to be persisted in the respective columns of the table(s), the object's class is mapped to. The identity of the object is directly related to the primary key, the database assigns to that object. An example would be the storage of a new employee with the employee number E-2132 in the database.
- **MA2** - Read an object: Reading an object, represents the materialization of a single object from the database. In this context, the object is accessed using the key attribute(s), which relate to the logical identity of the object and the entity it represents. An example would be the retrieval of the employee with the employee number E-2132.
- **MA3** - Update object: Updating an object is the process of modifying all or just number of a selected attributes for a specific object. The object is accessed via its identity or primary key respectively. An example would be an update of the lastname of employee with employee number E-2132.
- **MA4** - Delete object: Deleting an object results in the removal of all records and attributes that are linked to the identity of the object. An example would be the deletion of the employee with employee number E-2132.
- **MA5** - Polymorphic read: The "is-a"-relationship that constitutes inheritance hierarchies is transitive in nature. That means that a query on a class that has subclasses needs to know whether or not it should include its subclasses. Following that line of thought, a polymorphic query includes considers all records of class that it was executed on, including all of its subclasses. An example for a polymorphic read would be a query that lists all employees where the lastname starts with the letter "L". All employees in the context of our example means all internal and all external employees.
- **MA6** - Non-polymorphic read: In contrast to the polymorphic read, the non-polymorphic variant is restricted only to records that are of the type, the query was executed on. It does not consider records of any subclasses.
- **MA7** - Create new class attribute: A type of change in the object model might be that a class gets a new attribute. Looking at our example in Figure 1, this could mean that we want to store the middle name for all our employees. Consequently, the class definition for the type Employee is changed. That changed needs to be propagated to the data model as well.
- **MA8** - Update attribute data type: An update of the attribute data type means that the data type of the table column that stores a particular attribute is changed. The reason for that might be technical (data base optimizations) or logical (salary is changed from floating point number to double

precision floating point number to increase precision).

- **MA9** - Push attribute up: During the refactoring process of an application, it might happen that one or more attributes of a class are pushed to its superclass. Looking at our example in Figure 1, t could be that bonuses are now available for internal an external employees. In that case, *Bonus* would become an attribute of the employee class.
- **MA10** - Push attribute down: Similar to MA8, it can happen that an attribute is pushed to a specialized class. In our example, it could be that, for privacy reasons, the *Birthdate* is stored for internal employees only.
- **MA11** - Delete class attribute: MA11 can be considered as the reverse operation to MA7.
- **MA12** - Create new class from scratch: Application systems evolve and new functionality is added. In case new entities will be introduced to support that new functionality, new classes have to be added to the object model and consequently need to be mapped to the data model.
- **MA13** - Create new class via splitting: Refactoring may result in the splitting of a single entity into two or more specialized entities. That means, that an existing class is replaced by two or more classes. The objects of the existing class need to be distributed across the newly created classes, depending on their new role.
- **MA14** - Delete class: Similar to MA12 and MA13 a refactoring may trigger the deletion of a class from the object model. Depending of the specific semantic of that refactoring, objects of the deleted class may be erased or moved to other classes.

Subsequently, we added MA, which represent relevant functionality that has not been addressed directly in any of the analyzed documents. These MA are derived from the authors' experience with the implementation of O/R middleware. MA15 and MA16 result from refactoring approaches that consolidate two or more classes of the object model. Two different semantics can be found when consolidating classes.

- **MA15** - Merge classes via pushing up: This happens, when the consolidation is done by moving all attributes of a class to its superclass. Objects of the subclass need to be added to the superclass.
- **MA16** - Merge classes via moving horizontally: From a functional point of view, MA16 is very similar to MA15. However, there is an important difference. Where MA15 requires only an change and movement of object of the subclass, MA16 requires the adaptation of objects of both classes.

Additionally to these functional mapping aspects, we collected a set of non-functional requirements from the analyzed documents

- **MA17** - Space consumption: the required storage space of the database tables including indexes
- **MA18** - Mapping conformance: expresses the relational schema's resemblance of the object model
- **MA19** - Query complexity: the complexity to formulate queries

MA1 through MA19 represent a comprehensive set that covers relevant mapping aspects, which should be addressed by a document that intends to provide guidelines to understand the impact of a mapping strategy.

We do not claim this list to be complete. We would like to see it as a starting point for further discussion and invite researchers to add new aspects to that list.

3.3 Step 3: Requirements Organization

The requirements organization proposed in this paper is inspired by a paper from Holder et al. (Holder et al., 2008), where an approach towards a metric suite for O/R mappings is proposed. Their approach is based upon the ISO/IEC 9126 (ISO, 2001) standard for software quality characteristics. The ISO/IEC 9126 (ISO, 2001) standard is meanwhile superseded by the ISO/IEC 25010 (ISO, 2010). Its main contribution is a mapping from high level non-functional quality characteristics down to lower level characteristics. Figure 2 depicts their approach. Not all of the

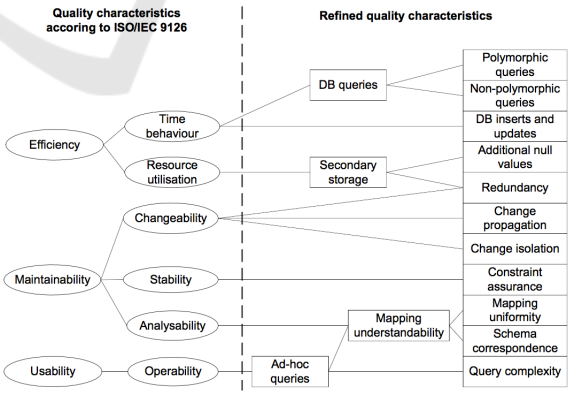

Figure 2: O/R Mapping Characteristics (Holder et al., 2008).

characteristics defined in (Holder et al., 2008) are applicable to inheritance relationships, e.g., constraint assurance, redundancy. For a comprehensive discussion about the considered quality characteristics, we refer to (Holder et al., 2008). Holder et al.'s approach

has two advantages. First, it allows the identification of different abstraction levels and secondly, it describes a relationship from non-functional system characteristics down to lower level mapping strategy aspects. Because we observed arbitrary switches between different levels of abstraction across and even within the analyzed documents, we propose a number of extensions to the existing approach. The main purpose of these extensions is to resolve possible misunderstandings or misinterpretations, which result from imprecise definitions or abstractions. Sections 3.3.1 through 3.3.3 elaborate on our proposed extensions.

3.3.1 Efficiency

Efficiency can be subdivided into *Time behavior* and *Resource utilization*. Holder et al. map *Resource utilization* to *Additional null values*, which is often used synonymously for storage space consumption. Documents that broached the issue of null values always related it to memory consumption, that is why we decided to replace *Additional null values* with *Memory consumption*. *Time behavior* is an abstraction of the query performance of the database, regarding the chosen data model. Holder et al. chose *Polymorphic*, *Non-polymorphic queries*, and *DB insert and updates* as appropriate subdivisions of *Time behavior*. We believe this subdivision to be sufficient, but to prevent any misunderstandings, we chose a more precise and elaborate naming. We subdivide *Time behavior* into the *CRUD operations Insert*, *Lookup*, *Update*, and *Delete* and the *Set operations polymorphic* and *non-polymorphic read*. From a database perspective, this categorization is more suited, because it allows a more precise distinction of the actual mapping aspects discussed.

3.3.2 Maintainability

Following ISO/IEC 25010 (ISO, 2010), *Maintainability* can be subdivided into *Changeability*, *Stability*, and *Analysability*. We disregard *Stability*, since there are no inheritance mapping related aspects that impact this characteristic. For a detailed discussions on that topic, we refer to (Holder et al., 2008). *Changeability* aggregates any system behavior that is related to change. Change in this context means a structural adaptation of the domain model and its effects on both O/R middleware and data model. Holder et al. propose two characteristics - *Change isolation* and *Change propagation*. These two characteristics are a good example for vague definition and loose semantics that we criticize. Holder et al. define *Change propagation* "as the extent to which it is necessary to adapt the relational schema and the O/R mappings to changes in the object model" (Holder et al., 2008). *Change isolation* on the other hand is defined as "the need to modify tables for adding or deleting classes" (Holder et al., 2008). The deletion of a class is a change in the object model, which raises the question why this aspect is considered to be part of *Change propagation*. We decided to adapt Holder's definition to make it more concrete. *Change propagation* reflects operations, which change the structure of a class, e.g., adding or deleting an attribute. *Change isolation* reflects all operations that change the structure of the inheritance hierarchy, neglecting the structure of the individual classes, e.g., adding or deleting a class. The resulting mapping aspects are derived from a simple enumeration of object model changes, which can occur in an OO programming language. The last subdivision of *Maintainability* is *Analysability*. Holder redefines this as *Mapping understandability* and further into *Mapping uniformity* and *Schema correspondence*. *Mapping uniformity* indicates the degree of hybridness, which reflects the variation of mapping strategies used for the hierarchy. It refers to the overall mapping specification and is not applicable to individual mapping strategies. That is why we do not regard it for further considerations. *Schema correspondence* reflects the object model's resemblance in the relational schema.

3.3.3 Usability

Usability is translated into *Operability* and is redefined by Holder to *Ad-hoc queries*. In its essence, *Usability* reflects the complexity to formulate queries for a particular strategy. Working with relational databases requires a fairly sophisticated understanding of relational algebra. This presents an increased complexity for a developer. Even though an ORM middleware is able to hide a good part of the complexity of SQL, there are very few systems that are able to hide the underlying relational database completely. Depending on the chosen strategy, the complexity to formulate efficient database queries varies. The aspect *Query complexity* reflects that.

With the structure depicted in Table 1, a developer can directly see, which mapping aspect impacts what non-functional characteristic of the system. With respect to our literature analysis, this structure allows to compare arguments and discussion points on different levels of abstraction. We will show that this is beneficial, because we are able to present reasonable explanations why some of the inconsistencies can be observed. It also helps authors of guidelines to structure and organize their documents, so developers have a more precise understanding of the guidelines scope

Table 1: Mapping Aspects Organization based on ISO/IEC 25010 (ISO, 2010).

ISO/IEC 25010 Characteristics	ISO/IEC 25010 Sub-characteristics	Refined Characteristics	Mapping Aspects
Efficiency	Time behavior	CRUD operations	MA1 - Insert
			MA2 - Lookup
			MA3 - Update
			MA4 - Delete
		Set operations	MA5 - Polymorphic read
			MA6 - Non-polymorphic read
	Resource utilisation		MA17 - Space consumption
Maintainability	Changeability	Change propagation	MA7 - Create a new attribute
			MA8 - Update attribute data type
			MA9 - Push attribute up
			MA10 - Push attribute down
			MA11 - Delete an attribute
		Change isolation	MA12 - Create a new class from scratch
			MA13 - Create a new class via splitting
			MA14 - Delete a class
			MA15 - Merge classes via pushing up
			MA16 - Merge classes horizontally
	Analysability		MA18 - Schema correspondence
Usability	Operability		MA19 - Query complexity

and intention.

4 CONSOLIDATION RESULTS

The consolidation approach proposed in Section 3 was applied for each of the three strategies - *Single-Table*, *Table-per-Class*, and *Table-per-Concrete-Class*. Tables 2, 3, and 4 depict our results. The selected documents were analyzed with regard to the mapping aspects identified in Section 3. For each document, we tried to identify discussion points that make a statement regarding the different mapping aspects. A (+) indicates a positive statement, meaning that the authors assign a positive impact of the mapping strategy to the particular mapping aspect. A (-) indicates a negative statement. We used (+/-) to indicate that the author assigned both a positive and a negative effect to the mapping aspect. This happens, when an author leads an argumentation on an abstraction level, which contains a number of lower level mapping aspects, e.g., *Change isolation*. On some of the lower level aspects, the strategy has a positive and on some it has a negative effect. If the author did not clarify the exact mapping aspect, but chose to focus the discussion on a more abstract level, it is not clear to the reader what the author's actual intention was.

As mentioned before, a high level of abstraction leaves room for interpretation. We tried to resolve imprecise discussions to the best of our knowledge and understanding, to map all statements of the analyzed documents to our mapping aspects. Basis for any interpretation are the mapping aspect semantics defined in Section 3.1. If a document contains a statement regarding a mapping aspect, we record the reference and the location of that statement within the document. Columns two through five refer to the selected publications and textbooks. Column six, labeled *ORM tools*, contains results derived from the analysis of technical documentations of ORM frameworks. In general, the technical documentations that we looked at focus on functionality. In most cases, the discussion of inheritance mapping strategies, with regard to the impact on non-functional system characteristics is very brief or not done at all. For that reason and for constraints to layout and space, we decided to summarize the results in one column.

5 EVALUATION

The results of the consolidation approach presented in Section 4 shows that there exist a number of inconsistencies and missing parts. In this section, we elaborate on the results of the consolidation approach. Furthermore, it contains reasonable explanations to why inconsistencies may occur.

5.1 Incompleteness

Incompleteness, i.e., the omission of relevant mapping aspects in the discussion of mapping strategies, can be identified by the empty cells in the result tables (Table 2, Table 3, Table 4). For each empty cells,

Table 2: Discussion Single Table.

	(Keller, 1997)		(Fowler, 2002)		(Ambler, 2003)		(Philippi, 2005)	ORM tools
MA1	+	p. 12 l. 4						+
MA2	+	p. 12 l. 4	+/-	p. 279 l. 15 / p. 279 l. 27	+	p. 240 l. 11	+	+
MA3	+	p. 12 l. 4						+
MA4	+	p. 12 l. 4						+
MA5	+	p. 12 l. 6	+/-	p. 279 l. 15 / p. 279 l. 27	+	p. 240 l. 8	+	+
MA6			+/-	p. 279 l. 15 / p. 279 l. 27	+	p. 240 l. 11	+	+
MA7	+	p. 13 l. 1					-	+/-
MA8	+	p. 13 l. 1					-	+
MA9	+	p. 13 l. 1	+	p. 279 l. 16			-	+
MA10	+	p. 13 l. 1	+	p. 279 l. 16			-	+
MA11	+	p. 13 l. 1					-	+
MA12	+	p. 13 l. 1			+	p. 240 l. 3	-	+/-
MA13	+	p. 13 l. 1			+	p. 240 l. 3	-	+/-
MA14	+	p. 13 l .1					-	+
MA15	+	p. 13 l. 1						
MA16	+	p.13 l. 1						
MA17	-	p. 12 l. 10	-	p. 279 l. 21	-	p. 240 l. 11	-	-
MA18			+/-	p. 279 l. 14 / p. 279 l. 19			-	+
MA19	+	p. 13 l. 4			+	p. 240 l. 13		

Table 3: Discussion Table per Class.

	(Keller, 1997)		(Fowler, 2002)		(Ambler, 2003)		(Philippi, 2005)	ORM tools
MA1	-	p. 14 l. 6			-	p. 242 l. 7		-
MA2	-	p. 14 l. 6	-	p. 286 l. 26, p. 286 l. 30	-	p. 242 l. 7	-	-
MA3	-	p. 14 l. 6			-	p. 242 l. 7		-
MA4	-	p. 14 l. 6			-	p. 242 l. 7		
MA5	+	p. 15 l. 1	-	p. 286 l. 26, p. 286 l. 30	+	p. 242 l. 5	-	-
MA6			-	p. 286 l. 26, p. 286 l. 30			-	-
MA7	+	p. 15 l. 7			+	p. 242 l. 10	+	+
MA8	+	p. 15 l. 7			+	p. 242 l. 10	+	+
MA9	+	p. 15 l. 7	-	p. 286 l. 28			+	
MA10	+	p. 15 l. 7	-	p.286 l. 28			+	
MA11	+	p. 15 l. 7			+	p. 242 l. 10	+	+
MA12	+	p. 15 l. 7			+	p. 242 l. 10	+	+
MA13	+	p. 15 l. 7			+	p. 242 l. 10	+	+
MA14	+	p. 15 l. 7					+	
MA15								
MA16								
MA17	+	p. 15 l. 5	+	p. 286 l. 21	+	p. 242 l. 15	+	+
MA18			+	p. 286 l. 23	+	p. 242 l. 3	+	
MA19	-	p. 15 l. 9	-	p. 286 l. 32				

it was not possible to find a passage in the document that discusses this mapping aspect. Consequently, a software developer, who uses this document as a resource to get information about the consequences of implementing a selected mapping strategy, would not find information regarding this particular aspect.

We note that the effort to understand and interpret existing literature is considerate and not trivial. We assume that it is not realistic that software developers would naturally consider such detailed examination. Omitting this effort, due to time or complexity reasons, would result in a lot more blank cells in the table. That is why we consider our effort to define semantics and resolve abstract discussions as an important contribution, which is valuable for software developers and guideline authors.

5.2 Inconsistencies

The inconsistencies become apparent, when comparing individual predicates (+, -, +/-) within each row. Rows that contain a positive statement as well as a negative statement show that there must be an inconsistent argumentation regarding the impact of that par-

Table 4: Discussion Table per Concrete Class.

	(Keller, 1997)		(Fowler, 2002)		(Ambler, 2003)		(Philippi, 2005)	ORM tools
MA1	+	p. 16 l. 10	-	p. 295 l. 17				+
MA2	+	p. 16 l. 10	+	p. 295 l. 14	+	p. 241 l. 7	+	+
MA3	+	p. 16 l. 10						+
MA4	+	p. 16 l. 10						+
MA5	-	p. 17 l. 1	-	p. 296 l. 7			+	-
MA6			+	p. 295 l. 13			+	+
MA7	-	p. 17 l. 7	-	p. 296 l. 5	-	p. 241 l. 3	-	
MA8	-	p. 17 l. 7	-	p. 296 l. 5	-	p. 241 l. 3	-	
MA9			-	p. 296 l. 1			-	
MA10			-	p. 296 l. 1			-	
MA11	-	p. 17 l. 7	-	p. 296 l. 5	-	p. 241 l. 3	-	
MA12	-	p. 17 l. 8					-	+
MA13	-	p. 17 l. 8					-	+
MA14							-	
MA15								
MA16								
MA17	+	p. 17 l. 4					+	
MA18			+/-	p. 295 l. 10 / p. 295 l. 18			+	
MA19	+/-	p. 17 l. 13 / p. 17 l. 11			+	p. 241 l. 3		

ticular mapping aspect. We observed two types of inconsistencies, which have different root causes. The first type of inconsistency results from different levels of abstraction and is discussed in Section 5.2.1. The second type is caused by ambiguity and imprecise semantics and is discussed in Section 5.2.2.

5.2.1 Inconsistencies through Abstraction

An example for contradicting opinions on different abstraction levels is MA12 - *create new class from scratch* - for the Single Table strategy in Table 2. An overview of the different abstraction levels and the corresponding discussions is depicted in Table 5.

Keller comes to a positive rating (Keller, 1997). He argues that the simplicity of the mapping allows an easy evolution. Ambler also comes to a positive rating. He argues that the creation of a new class requires merely the addition of new columns to the table (Ambler, 2003). Ambler's reason is on a lower level than this of Keller, but both come to the same rating. Philippi's evaluation of this point is negative. He argues that maintainability of the Single Table strategy is difficult for complex object models (Philippi, 2005). A more detailed reasoning is not mentioned. The documentation of Doctrine ORM argues that the addition of new columns also impacts existing index structures (Doctrine Team, 2015). Torque (Apache Software Foundation, 2012) evaluates maintainability positive, without giving any reason regarding this decision. Philippi and the Torque documentation discuss maintainability on the highest level of abstraction. They come to a contradicting rating. Ambler and Doctrine lead the discussion on a lower level. They mention the concrete use case of creating a new class, but they also are not of the same opinion. Whereas Ambler argues that the addition of new columns is simple. Doctrine remarks possible problems in the adaptation of existing indices. Both authors mention different aspects and consequently come to different ratings. Keller argues on the level of modifiability with a positive judgment.

Summarizing, it becomes clear that on different levels the authors come to divergent ratings. We believe that this observation is a clear indicator for missing standards and understanding, regarding abstraction levels and mapping aspects.

5.2.2 Inconsistencies through Missing Semantics

The evaluation of MA12 in Section 5.2.1 showed that argumentations on different levels of abstraction may lead to different results. A second important observation is inconsistency, caused by missing or imprecise semantics. An example for that is MA5 - *Polymorphic read* for the Table-per-Class strategy in Table 3. Fowler (Fowler, 2002) and Philippi (Philippi, 2005) argue that the Table-per-Class strategy is problematic due to the high cost of joins, which need to be executed to reconstruct a complete object. Keller (Keller, 1997) and Ambler (Ambler, 2003) argue that a *Polymorphic read* needs only one table access to identify the requested objects, because all objects have also an entry in their superclass tables. All authors discuss the use case of a *Polymorphic read*, but they consider different aspects of this complex operation. In that case,

Table 5: Example for Inconsistencies due to Abstraction (Table 2: MA12 - New class from scratch)

ISO/IEC 25010 Characteristics	**ISO/IEC 25010 Sub-characteristics**	**Refined Characteristics**	**Mapping Aspects**
...			
Maintainability **- Philippi (Philippi, 2005)** **+ Torque (Apache Software Foundation, 2012)**	Changeability **+ Keller (Keller, 1997)**	Change propagation	...
		Change isolation	Create a new class from scratch **+ Ambler (Ambler, 2003)** **- Doctrine (Doctrine Team, 2015)**
			...
	Analysability	Schema correspondence	
...			

a positive or negative judgment is merely a reflection of the author's interpretation of what a *Polymorphic read* actually consists of and what part of that has a relevant impact. To resolve such inconsistencies, we need detailed semantics and a quantification approach to clarify and measure the impact of that operation in different strategies.

6 CONCLUSION

The goal of this paper is to identify inconsistencies and incompleteness in existing guidelines and best practices for O/R inheritance mapping strategies. We described a systematic approach to identify and select literature, containing guidelines and best practices for O/R mapping strategies. We proposed a consolidation approach to compare and combine the information and discussions from the selected documents. The result of that approach is twofold. First, we derive a comprehensive set of 19 mapping aspects. This set merges the individual aspects identified in the analyzed literature and adds two new aspects, which are relevant, but have not been found in any of the analyzed documents. Second, the consolidation contains a proposal for a hierarchical structure of mapping aspects from high-level quality characteristics of the application, i.e., efficiency, maintainability, and usability to lower level features of O/R middleware. Our evaluation shows that none of the analyzed documents provides a comprehensive discussion of all 19 mapping aspects. This is problematic for developers, who need to understand the impact of a mapping strategy on their application's quality characteristics. Furthermore, the evaluation shows that there are inconsistencies across the literature. Due to missing or imprecise semantics and arbitrary abstractions, different authors come to inconsistent guidelines and best practices.

7 OUTLOOK AND FUTURE WORK

Our literature study and consolidation revealed inconsistencies and missing parts in existing guidelines and best practices for O/R inheritance mapping strategies. The reasons for that are complex, but we see the root cause of that in missing standards and imprecise definition of concepts. This becomes obvious by just looking at the different naming conventions for mapping strategies. Our analysis shows that the structures and relations between a system's quality characteristics and aspects of O/R mapping strategies are complex. Disregarding that complexity makes it difficult for software developers, to understand the impact of their architectural designs. In the future, we see two possible directions.

First, there should be an active discussion about the derived set of mapping aspects and the proposed organization of non-functional quality characteristics and mapping strategy aspects. We would like to see an active involvement of practitioners with experience in the development of O/R mapping layers to validate and possibly extend our proposed set of relevant mapping aspects. Furthermore, there may also be valid reasons to adapt the organization in depth or width. The contribution of this paper can be a first step towards a discussion for a frame of reference or even a standard, which can be used by scientists and practitioners to revise existing or develop new guidelines for O/R mapping strategies. Comprehensive and consistent guidelines are key to efficient and informed architectural decisions, when implementing O/R mapping layers.

The second direction that we would like to follow is the development of a quantification approach to actually measure the impact of a mapping strategy on the system's quality characteristics. During our literature study, none of the analyzed documents contained any cost model or empirical evidence that would sub-

stantiate the presented discussions. A quantification approach would be a valid and less vague replacement for informal guidelines and best practices. We plan to use the mapping aspects, defined in this paper as a starting point to conduct an empirical study and eventually develop a cost model. Such a cost model could be the basis for an automated strategy selection that allows to optimize an application towards a set of prioritized quality characteristics.

REFERENCES

(2001). Iso/iec 9126-1:2001 software engineering – product quality – part 1: Quality model.

(2010). Iso/iec 25010:2011 systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models.

(2011). Iso/iec 9075:2011 information technology - database languages - sql.

Ambler, S. (2003). *Agile Database Techniques: Effective Strategies for the Agile Software Developer*, chapter 14 - Mapping Objects to Relational Databases, pages 231–244. John Wiley & Sons, Inc., New York, NY, USA.

Apache Software Foundation (2012). Torque inheritance guide. [Accessed: June 22, 2015].

Apache Software Foundation (2013). Apache openjpa 1.2 user's guide. [Accessed: June 22, 2015].

Doctrine Team (2015). Doctrine 2 orm 2 documentation. [Accessed: June 22, 2015].

Fowler, M. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Holder, S., Buchan, J., and MacDonell, S. G. (2008). Towards a metrics suite for object-relational mappings. In *Proc. of the 1st Int. Workshop on Model-Based Software and Data Integration (MBSDI)*, pages 43–54, Berlin, Germany.

Keller, A. M., Jensen, R., and Agarwal, S. (1993). Persistence software: Bridging object-oriented programming and relational databases. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 523–528, New York, NY, USA.

Keller, W. (1997). Mapping objects to tables - a pattern language. In *Proc. of the European Conf. on Pattern Languages of Programming Conf. (EuroPLOP)*, Berlin, Germany.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report.

Philippi, S. (2005). Model driven generation and testing of object-relational mappings. *J. Syst. Softw.*, 77:193–207.

Propel Community (2015). Propel documentation. [Accessed: June 22, 2015].