

WHITE PAPER

OBJECT PERSISTENCE AND AGILE SOFTWARE DEVELOPMENT

By Dirk Bartels and Robert Benson

Agile Manifesto

We are uncovering better ways of developer software by doing it and helping others do it. Through this work, we have come to value:

- » Customer collaboration over contract negotiation
- » Working software over comprehensive documentation
- » Individuals and interactions over processes and tools
- » Responding to change over following a plan

That is, while there is value in the items to the right, we value the items on the left more.

ABSTRACT

In the past 10 years, Agile Software Development has proven itself to be one of the most effective methodologies to develop application software.

One of the challenges to the Agile Software Development approach can be the integration of non-object oriented sub-systems, such as MySQL, Oracle, or other relational database management system (RDBMS). RDBMS require mapping code to translate between the object model and the relational model of the database, generally known as “object to relational mapping” (ORM).

Managing and maintaining the mapping is not only time consuming, but the RDBMS and its schema is often managed under rather restrictive, and potentially “non-agile” policies. In addition, data already stored in an RDBMS may become inconsistent with the changes in the application made by the Agile development team.

The purpose of this white paper is to examine and compare RDBMS with several object persistence methods within the context of Agile Software Development.

We will quantify and qualify the impact of these methods on the overall velocity and success of an agile application development project. We believe that there are significant financial and time-to-market advantages in selecting true object persistence tools and methodologies over traditional relational database system in order to maintain an agile development approach. In other words, true object persistence tools and methodologies are necessary to reap true agile development benefits, and therefore achieve the business advantages of this development process.

69%

**of organizations
have adopted
agile.**

The programming magazine Dr. Dobbs Journal periodically surveys programming practices. In 2008, a survey⁽²⁾ found that 69% of the respondents indicated that their organizations are doing one or more agile projects. The respondents overwhelmingly indicated that agile teams are producing higher quality, have greater productivity, and enjoy greater satisfaction from the business.

OBJECT ORIENTED AND AGILE SOFTWARE DEVELOPMENT

Object-Oriented Programming (OOP) has gone mainstream, e.g., POJOs (plain old Java objects) are the standard software architecture approach in Java, and Microsoft developers are likely using the object oriented .NET framework to build their applications.

Agile Software Development is a reaction to the understanding that it is almost impossible to define, up-front, the functional requirements for a software development project. Functional specifications continue to evolve, and the application environment may change as well, requiring continuous changes to the code.

It is safe to say, that pretty much every agile software development project is based upon an object oriented analysis and design; there is a close relationship between the two, which we will examine further in this paper.

OBJECT PERSISTENCE

Object persistence emerged as a programming concept from object-oriented programming. Using object oriented design and an object oriented language, such as Java or C#, it becomes apparent that it is only natural and beneficial to consider some form of object persistence. Object persistence is a method of making “transient” objects outlive the lifecycle of objects held in memory by the application via a persistence mechanism irrespective of the encapsulated data structures of such objects.

“Persistent” objects will be stored, typically to a database on disk, so that these objects can be reloaded into memory at a later time as needed, possibly after restarting the application. Persistent objects continue to maintain their state and behavior in an application.

ORM and .NET

In the .NET framework, a capability known as language integrated query (LINQ) provides consistent querying locally and remotely for a variety of data types and sources, but does not provide mapping beyond the default case. LINQ does aid overall agility by making the code more powerful and more general. However it does not make the relational database schema more agile or aid in database migration.

There are several commonly accepted ways to implement object persistence:

- » **Manually writing code that maps the objects into a relational database (RDBMS).**
- » **ORM frameworks and/or standard APIs, such as Hibernate, Java Data Objects (JDO), Java Persistence API (JPA), and the Microsoft Entity Framework, that at least partially automate the OR mapping process.**
- » **Object oriented database systems (ODBMS), which can store objects natively, e.g., no mapping code required, and additionally may offer advanced tools for agile development, such as schema evolution. These systems also utilize industry standard APIs like JDO or JPA. More on this later.**

MAPPING APPLICATION OBJECTS INTO RELATIONAL DATABASES

An application's object model and a corresponding relational data model (also called database schema) can be quite different, hence the requirement for mapping to translate between the object state and the tables and columns of the relational database. The principle causes for mapping are:

- » **Objects may be any “shape”, and the objects must be mapped to sets of relational data records, expressed in tables, columns and foreign keys.**
- » **Objects may include single or bidirectional links to other objects, that must be mapped to relational links. Relational links can only be modeled via one-way foreign key relationships, requiring the developer to build and maintain association tables to represent the broader object model capabilities.**
- » **Objects support many design concepts, such as embedded containers and class hierarchies, which are not available in a relational database and therefore must be programmed by the application and mapped into a relational abstraction for such design concepts.**

Code Refactoring

Code Refactoring” was introduced by Martin Fowler in 1999 in his book *Refactoring: Improving the Design of Existing Code* (3). Refactoring is defined as small changes to the code which result in better design and clearer code, more able to respond to change, without changes to the behavior or semantics. Code refactoring is an essential ingredient to agile development.

General purpose ORM frameworks provide a number of tools, APIs, and mechanisms to alleviate the application programmer from writing the object to relational mapping code.

These ORM frameworks are prevalent in Java, and to a lesser extent also available for C# and C++ programming. ORM frameworks are known for providing significant productivity gains over home grown OR mapping or direct data access programming, however, most developers experience performance and maintenance issues with the generated code, which often leads to additional coding, e.g., to tweak the generated mapping code and tune the database access layer. These tools are generally speaking very agile the first time you map your model to the database, however, they are significantly less agile as your model evolves.

Home grown, manually written, and proprietary persistence frameworks are also very common. Those frameworks may result in up to 40% of the application code base addressing persistence and OR mapping⁽⁷⁾.

Regardless of the approach OR mapping may look like a trivial task at the beginning of a project; however, it typically continues to ‘mushroom’ as the application model evolves and introduces more and more complex requirements for the database developer. Object oriented application developers often spend many development man years to maintain and improve the ORM layer.

AGILE DEVELOPMENT, CODE REFACTORING AND DATABASES

When a database is part of an agile application development project, it typically requires refactoring of the database code (e.g., schema, mapping and data) like other application code.

Traditionally, database developers and administrators follow the so called “waterfall software development methodology⁽⁴⁾.

Change Management

Change Management is a technique to control change in systems. In complex production systems, controlling rogue change is an important issue. In development projects, controlling change using the same methods and processes as in complex production systems is often destructive behavior, an anti-pattern. The Change Management process prevents change, prevents agility, and prevents the creation of value.

It requires a careful and time consuming planning, design and review process known as Change Management to make any changes to an existing database schema. This approach to schema management has a substantial impact on the velocity of an agile project.

Agile database refactoring implies changing the schema of the database, re-programming its access layers, as well as migrating the current stored data to the new database schema frequently, as opposed to the slow and highly managed waterfall methodology. There is an inherent conflict between the agile software developer and the traditional relational database developer / administrator that must be addressed.

Scott Ambler and Pramod Sadalage discussed refactoring of relational databases in the context of agile software development in their book *Refactoring Databases*⁽⁵⁾. The book describes very advanced techniques that are still rarely practiced.

The question remains, how can an agile software development process integrate with a database without a significant loss in project velocity, given the challenges of OR mapping, and limited capabilities and skills for the refactoring of relational databases?

DATABASE REFACTORIZING READINESS

The velocity of an agile software development project can easily be limited by the readiness for database refactoring, and the utilized persistence method, e.g., OR mapping. The project team's ability to refactor the database depends on the database design and development perspective, as well as the utilized tools and methodologies.

Three database design categories are described in the table below, on a continuum from Agile Object Persistence, to Traditional Relational Database Management, to further facilitate this discussion. We assume a project with a reasonably large database schema with some inherent complexity. We found that the time required to refactor a database can vary significantly, from just about a few minutes to several months.

Obviously this also depends on the overall project size, size of the database schema and the requested code changes:

- » **The Agile Object Persistence team can refactor the database, e.g., modifying the schema, and migrating the data, most efficiently, e.g., by using tools and automation for most of the necessary tasks.**
- » **The Relational Refactoring team can refactor the database, e.g., modifying the data access layer and database schema, tune the OR mapping layer, and migrating the data, however, many of the refactoring tasks are performed manually, which can be error prone and less efficient.**
- » **The Traditional Relational Database Management team really doesn't endorse "refactoring", nor does it utilize tools to support code and data refactoring. Any change to the database schema must go through a well defined Change Management process, which obviously will take significant more time given the people and the process involved.**

As we can see easily, agile database development comes down to a couple of important aspects.

1. The database developer and the database administrator must be integrated into the agile application development team; ideally, the database development is directly managed by the application developer.
2. The tools and methodologies used to implement Object Persistence are critical to achieve fast iterations. Fast application iteration cycles require a complete abstraction for object persistence in order to continue to evolve the application code, and not be hampered by the data access layer. It is obvious that the more automation is provided by the object persistence layer, the lesser the impact on the application development. ODBMS provide a clear advantage here.
3. Using a traditional database development methodology and an advanced, agile software development methodology together is counter productive, and is not going to support agile development projects.

Database Refactoring Readiness

Category	Agile Object Persistence	Relational Refactoring	Traditional Relational Database Management
Approach	<p>Expert adoption of agile techniques for all aspects of the project, e.g., utilize best tools and designs to facilitate fast iteration cycles.</p> <p>Uses an agile data access technology, for example an ODBMS, or implements a fully transparent OR mapping.</p> <p>Application developers have full control over the database refactoring and drive the evolution of the entire application and database.</p> <p>Adoption of the techniques described in “Refactoring Databases” if RDBMS is implemented.</p>	<p>No prior expertise at refactoring databases.</p> <p>Database design and tool support doesn’t fully support fast iteration cycles, and may require tedious manual coding. The database schema may still be exposed to some extent in the application code.</p> <p>Separate database and application programming teams may require several iterations to complete one application evolution cycle.</p>	<p>Implements full “Change Management” without considering agile software development aspects.</p> <p>Complete separation of application development and database design / administration.</p> <p>Application developers have no influence on database design, and may have to redesign the application based on the database restrictions and limitations.</p> <p>Refactoring of databases viewed as poor practice</p>
Tools	<p>Complete implementation of ORM or ODBMS tools necessary to reduce or eliminate dependencies between application and database.</p> <p>Fully automated test suite, e.g., to quickly validate code and database iterations.</p>	<p>The application team may have implemented an OR mapping tool or layer, however, it relies on a separate database developer to synchronize changes made in the application code.</p>	<p>Change Management*, ORM possible, but not exploited as a rapid iteration tool.</p>
Database Refactoring Timeline	<p>Refactoring is very efficient and fully supported by tools and automation.</p>	<p>Refactoring is part of the agile development project management, however, tasks are being performed more manually and are by definition less time efficient and more error prone</p>	<p>A lot of time is necessary to run the change management process through its various stages and stake holders.</p> <p>A lot of time will be used in analysis, design and project coordination, so for example, application development requests are being considered by the database admin, but only partially implemented, which may lead to a series of iterations to complete one development cycle.</p>
Technology	<p>Transparent Object Persistence, implemented via ODMBS or ORM</p>	<p>„Explicit“ object persistence, impemented with RDBMS, possibly utilizing an ORM tool</p>	<p>RDBMS, Change Management</p>

Common Examples for Database Refactoring

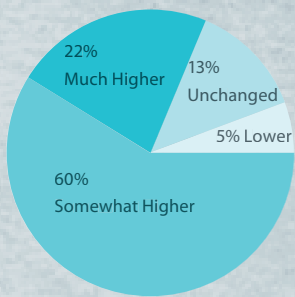
Change	Relational Database Code	Relational Database	Versant Object DB
Add attribute	One or more tables needs attribute column added, depends on class hierarchy, “flattened out” tables,e.g., producing redundant attribute values to eliminate joins.	Initial values must be filled in for all effected tables rows.	Recompile code, run schema utility. Database automatically handles schema evolution.
Drop attribute	Remove use of attribute in code.	Remove column in required table(s) based on class hierarchy.	Recompile code, run schema utility.
Add 1:N relationship	<p>Add a foreign key attribute, program relational join operation to retrieve related object and map into program object.</p> <p>Difficulty comes in if the relationship must model references to objects of different classes. (see polymorph relationship).</p> <p>Deleting objects will require substantial programming to avoid “dangling” relationships.</p>	<p>May require index definition on foreign key, database reorganization to create the index.</p> <p>Additional index, and join operation will impact performance and general resource requirements.</p>	<p>Recompile code to verify object relationships.</p> <p>The class’ destructor easily handles deletions of dependent objects.</p>
Add N:M relationship	Add two foreign key attributes into the database schema, one on each side of the relationship. Add a relationship table, program join operations on each side of the relationship, and map retrieved information into program objects.	<p>Requires index definition in at least 2 tables, create a “relationship table” and index on each relationship.</p> <p>Requires database reorganization to create the index. Additional table, index and join operations will impact performance and general resource requirements.</p>	Recompile code to verify object relationships.
Add polymorph relationship	<p>Additional complexity in all relationships must be addressed if references are polymorphic, and relationship must determine the type of each object before determining the join operation.</p> <p>Additional programming required for all join operations.</p>	Depending on implementation strategy, there are a lot of different things to consider. If not, one of the key benefits of object technology, polymorph relationships, must be abandoned	Recompile code to verify object relationships.

82%

of agile approaches result in higher productivity.

How have agile approaches affected your productivity?²

- 22% Much Higher
- 60% Somewhat Higher
- 13% Unchanged



LACK OF AGILITY IN DATA SLOWS VELOCITY

A lack of agility in implementing and maintaining the database access layer clearly slows the overall project velocity, and prevents fast iteration cycles that are fundamental to Agile Development. Furthermore the lack of agility prevents rapid feedback from the users on new versions of the application defeating another main purpose of agile development.

An agile project must be able to implement changes to the domain objects. Ideally the project utilized tools that trigger automated processes to make the necessary changes in the database schema and the data access layer, and help migrating the actual database, often in minutes.

On the traditional Relational Database Management extreme, a change to the database schema and migration of the database may take an eternity – relatively speaking of course – and therefore is not compatible with the Agile Development Manifesto.

OBJECT PERSISTENCE, AGILITY, AND ODBMS

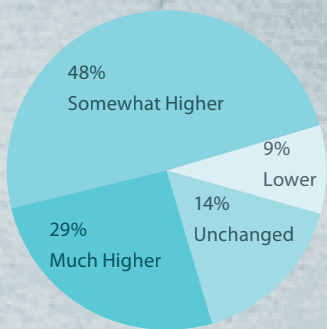
ODBMSs such as the Versant Object Database and Versant db4o provide tools which automate schema evolution and data migration and make this process easy for the developer. Furthermore, there is no OR mapping code necessary. The ODBMS manages objects “natively” in the database. Therefore, there are no code changes required to the data access layer – it’s called “transparent object persistence”. Any change to the application code that relates to the database requires only a rebuilding of the application to have the changes reflected in the ODBMS. These capabilities enable the team to achieve code iterations whenever needed, and provide the least obstacle to agile development.

77%

of agile systems experience higher quality.

How have agile approaches affected the quality of systems deployed?

- 29% Much Higher
- 48% Somewhat Higher
- 14% Unchanged
- 9% Lower Quality



Object relational mapping software, such as Hibernate, DataNucleus, or TopLink, provide tools to facilitate relatively agile processes to map the application objects into a relational database. These ORMs are frameworks, and use specialized metadata to enable the developer to declaratively define most aspects of object relational mapping. Certain changes, such as a name change or adding / dropping a simple attribute from a class, can be made reliably in minutes by changes to the metadata, and therefore support agile development. Other changes, for example modifications to the application class structure and its object hierarchies and graphs, are significantly more complex and not really automated by an ORM tool. In these cases, the ORM tool may support agility in the programming interface, however, the ORM requires code changes in the underlying data access layer and complicated database reorganization / refactoring in ways described in Refactoring Databases. In these cases, even when utilizing ORM, the development process is delayed until those database changes are fully implemented and tested.

THE BOTTOM LINE

Even though it is clear that projects differ, business models differ, and circumstances differ, it is obvious that dollars are saved when development cycles are shortened, and faster iterations and more feedback from the consumer early in the process will help develop a better product.

The general value of agile software development is reasonably well understood. However it is difficult to establish a definitive dollar value resulting from agile development and object persistence. The following somewhat simplified approaches provide some perspective on the potential value of integrating the database development into the agile development process.



A successful agile project brings financial benefits

INCREASED PROJECT VELOCITY TRANSLATED TO DOLLARS SAVED

If a change in agility doubles the overall project velocity (development rate) of the team, the cost savings is the run rate of the development expenditure times the project duration at the original velocity. For example, a change in agility that doubles the velocity of a project originally taking one year and spending \$100K per month, cuts the total development time in half and saves \$600K. $((12 \text{ months} - 6 \text{ months}) \times \$100\text{K}/\text{mo})$ This calculation assumes that the project is completed, without ongoing development. In the case of continuing ongoing development, cost savings are even greater.

PROJECT VELOCITY INCREASES PROFITABILITY

A successful agile development project brings financial benefits in the following principle way

- » **It lowers the costs of developing the product by accelerating the development process and reducing the overall project duration.**
- » **It helps developing a better product, by providing continuous feedback on product iterations, e.g., from test harnesses as well as pre release users.**
- » **It increases the business value of the software because it shortens the time to market, with immediate and long term effects on pricing, competition, market share, and more.**

3x

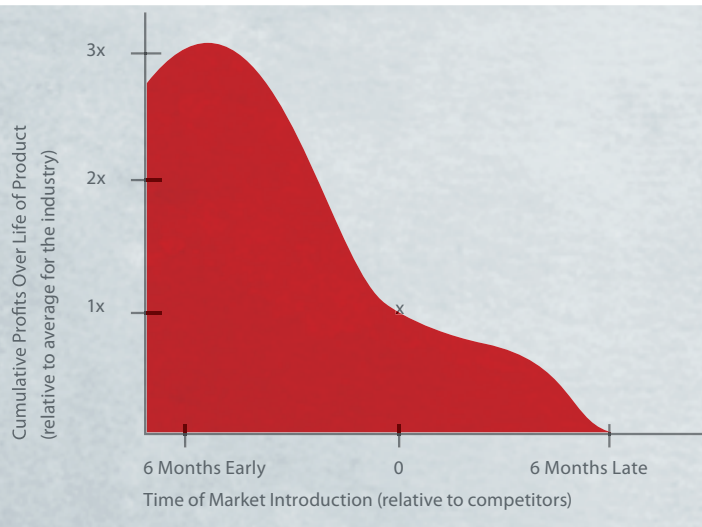
introducing a product to market early can lead to up to three times more profits over its lifecycle

Noted expert Steven Wheelwright⁽⁶⁾ estimated that a six months advantage in time to market in a consumer product could result in three times the profit over the lifetime of the product.

The direct cost of a software project is straightforward; shortening the development process will save money. The true business value of agility may be much greater when looking at the “time to value” and the customer acceptance of the software.

Noted expert Steven Wheelwright⁽⁶⁾ estimated that a six months advantage in time to market in a consumer product could result in three times the profit over the lifetime of the product. In the case of a six months delay, his research shows profit degradation to break-even. It is easy to imagine how delays caused by a slow iteration of the data layer can derail a project’s profitability under these assumptions. Using a traditional RDBMS without any object persistence tooling may cause delays of many days to several months in each iteration cycle, essentially rendering agile development more or less useless. Conversely, the use of agile Object Persistence can enable the agile development process, and enables the inherent benefits of the agile development methodology, e.g., better, cleaner code, higher user acceptance, shorter project schedule, etc.

The Impact of Market Introduction Timing on Lifetime Profits of a Major New Product





Dirk Bartels

Vice President Strategic
Product Management,
Versant Corp.

Robert Benson

Robert Benson is a
consultant in middleware,
distributed computing and
cloud computing.

CONCLUSION

This paper shows clearly that when changes to the data model and schema are required in agile development, the tools, capabilities of object persistence, and in particular combined with an object database, provide a huge advantage compared with implementing relational databases. Changes to the schema (objects) and migration of the database will take the least amount of time using an object database, compared to the same changes using relational databases, with or without an ORM. There is no question that utilizing ORM when using a relational database is beneficial, and gets the agile developer as close as possible to be agile, however, even ORM are not able to compete with an ODBMS in regards to managing code changes during the project development phase.

Thinking out of the box, a developer may consider standardizing on an object persistence API, such as JDO, JPA or .NET and using a compatible ODBMS during the development, and switching over to a compatible ORM tool before deploying the application. Taking this route, the developer can take full advantage of the fastest iteration cycles using an ODBMS, and still manage to deploy on a relational target platform, if so desired.

For more information on Versant object databases go to www.versant.com

REFERENCES

1. http://en.wikipedia.org/wiki/Agile_Manifesto
2. Results from Scott Ambler's February 2008 Agile Adoption Survey posted at www.agilemodeling.com/surveys/
3. Fowler, Martin. (1999) Refactoring: Improving the Design of Existing Code, Menlo Park, CA: Addison-Wesley Longman.
4. http://en.wikipedia.org/wiki/Waterfall_model
5. Ambler, Scott and Sadalage, Pramod (2006) Refactoring Databases: Evolutionary Database Design, Menlo Park, CA: Addison-Wesley Longman
6. Wheelwright, Steven C., Clark, Kim B., (1992) Revolutionizing Product Development, Simon & Schuster (page 22)
7. Barry & Associates, Object Relational Mapping, http://www.service-architecture.com/object-relational-mapping/articles/writing_your_own_mapping_layer.html