VERSANT

对象持久与敏捷软件开发

软件设计白皮书

Dirk Bartels 与 Robert Greene

编制：Versant 中国

2009 年 11 月

Versant China

上海市昆明路 572 号 B 区 415-419 室

邮箱： info@versant-china.com

电话： (021) 5172 1968

传真： (021) 5172 1967

网址： www.versant-china.com

概述

在过去的 10 年中，敏捷软件开发已经被证明是最有效的软件开发方法之一。敏捷软件开发方法的挑战之一是需要将非面向对象子系统，例如 MySQL，Oralce 或者其它关系型数据库管理系统（RDBMS）与现有的，面向对象的系统集成在一起。关系型数据库要求将源代码在对象模型和关系模型之间进行“翻译”，也就是众所周知的“对象关系映射——ORM”。管理对象到关系的映射不仅仅非常消耗时间，而且关系型数据库及其 Schema 经常处于受限，或者无法“敏捷开发”的状态下。而且，存储在关系型数据库中的数据经常可能会无法与采用敏捷开发方法的团队对应用所进行的变更相匹配。

敏捷开发宣言

我们通过开发和帮助别人开发发现了实现的软件开发最佳方法。通过这种方法，我们获得了一下价值：

个人与交互：通过进程与各种工具

工作软件：通过大量文档

客户协作：通过大量合同谈判

快速响应变更：通过遵循计划

虽然过程和方法也很重要，但是最终的结果才是更重要的。

实际上，关系型数据库的数据由于是采用独立管理、独立建模的方式，它的自成体系必然会要求开发团队将有限的开发资源分成两部分。团队的管理者不仅仅需要管理源代码的同步，而且还需要管理实体关系模型的同步，进一步还要保持二者之间映射的同步。在这种情况下，大量的资源就被浪费甚至内耗掉了。尤其是当采用敏捷开发方法，源代码处于经常性的快速变动过程中时，保持对象模型和实体关系模型的一致就是一个费时费力的工作。

这种情况不仅仅会发生在采用敏捷开发方法的团队中，而且也会大量发生系统构建原型期。由于在这个阶段系统需求无法完全明确，可能会根据系统产出有快速迭代的要求，尤其是在中国的特殊国情下更是如此。在一些复杂的、对性能要求很高或者复杂度很高的系统中，这种对象-关系维护成本造成的开发团队的注意力不集中甚至会危及整个原型产品的生命。

本文将通过检讨和比较在采用敏捷开发方法的团队中，采用关系型数据库以及集中常见对象数据库的数据持久方法，对整个开发效果所带来的不同。同时，我们将对不同的持久方法对敏捷应用开发项目的开发速度和开发质量所带来的冲击进行量化的描述。我们相信在采用敏捷开发方法的项目中，采用真正的对象持久工具，相比较于传统的关系型数据库管理系统，能够大大改善开发团队的财务状况，以及通过提高开发速度来提高产品抢占市场

先机的能力。换句话说，只有真正的对象持久工具以及持久方法才能保证开发团队获取应得的利益，并在残酷的商业竞争中不断保持优势。

面向对象与敏捷软件开发

面向对象程序开发（Object-Oriented Programming）已经日渐成为主流，也就是说POJO（Plain Old Java Objects）是 Java 的标准软件架构方法。而且 Microsoft 的开发者们也日渐开始喜欢采用面向对象的.NET Framework 来构建他们的应用。

敏捷软件开发是一个针对一些特别场景的解决方案，在这些场景中，用户几乎无法完整的定义目标软件的功能需求并保持稳定。这些功能需求处于不断、持续的演变过程中，并且应用环境也可能会随之不断发生变化（我们可以考虑一个业务处于快速增长状态下的应用，例如一个网络游戏或者在线社区或者高级客户服务系统，其服务器以及网络环境必然会随着业务系统的增长而不断发生变化，例如增加服务器内存、CPU 数量甚至增加服务器的个数，提高网络带宽、对服务器进程进行优化等等、等等），这种场景的一个共性就是要求应用的程序代码也要经常性的进行变化来保持同步。

我们可以确定的说，基本上所有的敏捷开发项目都采用了面向对象的分析、设计和开发方法。敏捷开发方法和面向对象开发方法之间有着非常紧密的联系。本文随后将对这种情况进行深入讨论。

对象持久

对象持久是从面向对象程序开发中形成的一个概念。在使用诸如 Java 或者 C#之类的面向对象程序开发语言的项目中，很显然只有使用面向对象的对象持久形式才是最自然和节约的。对象持久是一种可以使“临时”的对象的生命周期扩展到应用程序的内存之外，而无须关心封装在这些对象中的数据细节的方法。

“持久”的对象在通常情况下会被数据库系统保存在磁盘上，这样在我们需要这些对象的时候，例如重新启动应用程序时，就可以被重新装载。持久对

敏捷开发的成熟与推广

程序开发杂志 Dr. Dobbs 会定期回顾一些软件开发方法实践。在 2008 年，一次回顾显示有 69%的受访者表示在他们的企业中曾或多或少的执行过敏捷开发项目。受访者普遍认为使用敏捷开发方法能够提高项目的代码质量，提高生产率，并从最终用户那里获得更多的积极反馈。

象在这种情况下可以连续地保存在应用中的状态以及行为方式。目前有一些常见的方法用于实现对象持久：

- 通过手动编程方式将对象映射成为关系型数据库中的数据；
- 利用诸如 Hibernet、Java Data Objects（JDO）、Java 持久 API（JPA）以及 Microsoft Entity Framework 之类的 ORM 框架和/或标准 API，至少部分地自动完成 OR 映射；
- 使用可以直接保存对象，也就是无需映射代码的面向对象数据库系统（ODBMS）来保存对象，这些对象数据库系统还往往可以提供一些包括 Schema 演化工具在内的额外工具，来加快敏捷开发的进程。这些系统或多或少的也可以支持诸如 JDO 或者 JPA 之类的标准 API。

映射应用程序对象到关系型数据库

一个应用程序的对象模型及其对象的关系数据模型（也被称为数据库 Schema）可能是很不一样的，因此经常需要将对象状态和关系型数据库的库表以及列的数据之间做一个映射。映射的原则可以被总结为：

- 虽然对象可能是任何“形状”，但是如果需要持久，就必须被映射成为一组关系型数据库的记录，这些记录可以用表、字段和外键来进行描述；
- 对象可能会包含到其它对象的单向何双向引用，这种引用应该被映射为关系型数据库的连接。关系型数据的连接只能通过单向外键引用关系来进行描述，这就需要程序开发人员通过建立关联表来描述这种更加宽泛的对象引用关系。
- 对象可以支持许多设计概念，例如嵌入式容器、类继承等等，而且高级面向对象开发语言中还有一些设计模式，可以使问题更加复杂化。而所有这些概念都不是一个关系型数据库系统所能够支持和管理的。因此，这一切都要求程序员来通过自己的努力将这些模型映射成抽象。

ORM和.NET

在.NET 框架中，一个名为语言集成查询(LINQ)的特性虽然提供了本地和远端，数据结构和数据源均具有一致性的查询能力，但是并没有提供必要的映射机制。LINQ 虽然确实能够帮助敏捷开发将目标代码变得更加有效和通用。然而，它并不能使得关系数据模型更加的敏捷，也不能在数据库的迁移方面提供任何帮助。

现有的一些 ORM 框架可以提供一组工具、API 以及映射机制来减轻程序员将对象模型映射为关系模型的代码工作量。这些 ORM 框架在 Java 世界中非常流行，而且在 C#和 C++环境中也可以实现类似的功能。ORM 框架相比较于开发团队自己的 OR 映射实现，以能

够快速实现 OR 映射功能，显著降低 OR 映射代码开发量而著名。但是，绝大多数程序开发人员都会在使用 ORM 框架的过程中，遇到比自己开发相应的映射代码要更多的性能和维护等方面的问题。在性能方面遇到问题的解决方案往往需要通过再开发额外的映射代码，或者对自动生成的映射代码进行修改，或者调整数据库访问层相关参数的方法来解决。这时我们就会看到，虽然这些 ORM 框架工具在第一次将模型映射到数据库时非常敏捷，但是在模型发生演变之后会变得非常的不敏捷。

在现在的开发环境中，也往往会出现自己开发持久框架的情况。这些框架往往会导致仅仅为实现 OR 映射，就要额外编写高达 40%的代码的情况。

虽然 OR 映射在项目建设初期可能会被看作是一个很小的任务，但是这个“小”任务经常会随着应用程序模型的增长而持续增长，这样会给数据库开发人员带来越来越多，越来越复杂的工作量。面向对象的应用程序开发人员经常需要花费更多的人/月来维护和强化 ORM 层。

敏捷开发，代码重构与数据库

当一个敏捷开发项目中使用到数据库时，就经常会需要重构数据库代码（也就是数据库存储方案、映射以及数据），就像其它的应用程序代码一样。一般情况下，数据库开发人员和管理人员也遵循所谓的“瀑布软件开发方法”。瀑布方法要求在对现有数据库存储方案进行修改之前，仔细地完成对规划、设计以及审阅流程的时间投入计划，就像大多数变更管理系统所能够完成的那样。这种对数据库存储方案的开发方法对敏捷开发的开发速度有潜在的不利影响。

变更管理

变更管理是一个用于控制系统变更的技术。在复杂的生产系统中，控制各种复杂的变更是一个重要问题。在开发项目中，控制变更所采用的方法往往对于复杂生产系统会产生破坏性的效果，是一个反模式。变更管理过程阻碍了变更，阻碍了敏捷性，阻碍了最佳价值的产生。

敏捷数据库重构意味着对数据库存储方案的变化，对数据存储方案的访问代码的重新编程，以及经常性地将原有数据存储方案中的数据迁移到新的数据存储方案中，这与瀑布开发法所期望的场景正相反。需要强调的是，在敏捷开发工程师和传统数据库开发工程师/系统管理员之间存在固有的冲突。

Scott Ambler 和 Pramod Sandalage 在他们的新作：***关系型数据库重构*** 中讨论了在敏捷软件开发方法下的关系型数据库重构。该书描述了目前还难以被推广的高级技术。

但是问题依然存在，敏捷软件开发过程应该如何与数据集成，同时还不严重损失开发速度，不会在 OR 映射上投入大量工作，以及不受到缺乏关系型数据库重构经验和能力的困扰？

数据库重构的充分性

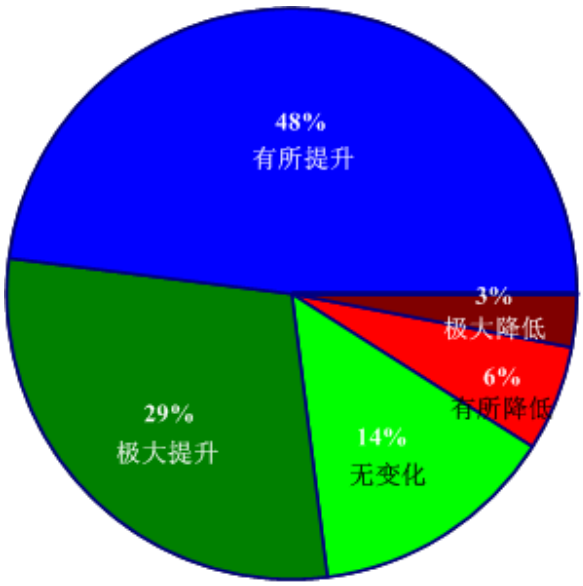
敏捷开发项目的开发速度很容易受到数据库重构的可行性以及对象持久方法（也就是 OR 映射）的限制。项目组重构数据库设计的能力往往依赖于数据库设计、设计的可扩展性以及使用到的工具和方法。

代码重构

“代码重构”由 Martin Fowler 于 1999 年在他的新书“重构：强化现有代码的设计”首次提出。重构被定义为对代码的一系列小修改，这些修改可以在不影响业务逻辑和语法的情况下，使得代码可以有更好的设计，并且更加清晰和易于修改。代码重构是敏捷开发的重要组成部分。

以下的表格中描述了数据库设计的三个分类，深入的体现了从敏捷对象持久到传统关系型数据库的变化关系。我们假设的项目具有合理规模的数据库存储方案，并且包含一定的内在复杂性。我们发现重构数据库所需要花费的时间可能会有很大差异，从仅仅几分钟到数月。很显然这依赖于整个项目的大小，数据库存储方案的大小以及可能会受到影响的代码的范围：

- 敏捷对象持久开发团队可以更有效的使用工具以及自动化流程重构数据库，也就是修改数据库存储方案以及执行数据迁移，来实现大多数任务。
- 关系型数据库重构开发团队同样也可以重构数据库，也就是修改数据库访问层，调整 OR 映射层，以及迁移数据。但是，许多这样的工作都必须依赖手工完成，这不仅很容易出错，而且效率不高。
- 传统关系型数据库开发团队几乎不能被认为是“重构”数据库，也几乎不可能利用工具来支持代码和数据的重构。对数据库存储方案的任何修改都必须通过明确定义的变更管理流程，以保证任务可以得到正确执行。这种方式显示要涉及到更多的时间投入和人力投入。



图表 1 敏捷开发对应用部署的影响

由此我们可以明显看出，敏捷数据库开发最终要通过以下特性来体现：

1. 数据库开发工程师和数据库管理工程师必须投入敏捷应用开发团队中。在理想情况下，数据库开发工作应该直接受到应用开发工程师的控制。
2. 实现对象持久的工具和方法对于实现快速迭代而言具有至关重要的作用。快速应用迭代的每一个周期都要求对对象存储有完整的抽象定义以保证应用代码可以顺畅发展，而不会受到数据访问层的阻碍。
3. 使用传统数据库开发方法在软件生产率方面是和使用高级的、敏捷软件开发方法对立的，无法为敏捷项目开发提供有效支持。

数据库重构的准备成熟度比较表

分类	敏捷对象持久	关系型重构	传统关系数据库管理
方法	敏捷项目开发的专业解决方案，使用最好的工具和设计来实现快速迭代。 使用敏捷数据访问技术，例如，使用对象数据库，或者实现一个全透明的OR映射机制。 应用开发工程师对数据库重构具有全面的控制能力，并能能够实现整个应用数据库的演化。 如果使用关系型数据库，能够实现“数据库重构”中所描述的技术。	在数据库重构方面，在项目内没有可供借鉴的经验。 数据库设计和工具并不能完全支持快速开发迭代周期，可能只能通过乏味的手工编程实现。数据库存储方案可能会被一定程度的暴露给应用程序代码。 数据库和应用开发可能需要由不同的团队来完成，这样就可能会需要几个小的迭代来完成一个完整的应用演进迭代。	完全通过“变更管理”系统来实现，无法实现敏捷开发的特性。 应用开发与数据库设计/管理全部割裂。 应用开发工程师可以对数据库设计没有影响，而且反而会由于数据库的限制和局限而不得不修改应用程序的设计。 数据库重构基本上是不现实的。
工具	完整的ORM 或ODBMS工具支持，可以减少或者消除应用程序和数据库之间的依赖关系。 完整的自动化测试工	应用开发团队可能要实现一个OR映射工具或者层，但是，这取决于一个独立的数据库开发工程师来将应用程序中发生的变化同步到数据	变更管理*, ORM虽然存在，但是不能被认为是一个高效的迭代工具。

	具，能够快速验证代码和数据库迭代。	库存储方案中。	
数据库重构时间周期	重构非常高效，并且支持由工具自动化完成。	重构是敏捷开发项目的组成部分，然而，相关的任务必须部分通过手工完成，这样必然会更加耗时，而且更容易出错。	需要大量的时间来执行变更管理系统的流程，需要通过流程中所定义的各种审批。 分析、设计和项目协调需要耗费大量时间。因为程序开发工程师和数据库管理员各自都需要完成部分工作，因此一个产品开发迭代需要由两个甚至多个小迭代来实现。
核心技术	通过ODBMS或者ORM实现“透明”对象存储。	“精确的”对象持久，通过关系型数据库实现，可能会使用ORM工具。	关系型数据库，变更管理系统

常见的数据库重构例子

变化需求与常见问题	**关系数据库代码**	**关系型数据库**	**Versant对象数据库**
一个或者多个表需要增加属性字段，根据层次化的类结构定义，需要被分拆到不同的表，这会需要有冗余的属性字段来描述属性间的关联关系。	**增加属性**	所有受影响表的所有相关字段都必须被赋役正确的初始值。	重新编译代码，执行Schema工具，数据库会自动处理存储方案的更新。
如果必须通过关联来反映不同类的对象之间的映射关系，如多态关系，就会为应用开发造成困难。 增加外键属性，开发联立查询程序来查找相关对象并映射成程序可用的内存对象。 需要额外开发程序来处理后续的对象删除操作，并且处理可能会发生的孤儿对象问题。	**增加1:N 关系**	可能会需要在数据库中建立外键索引关系，并且重新组织数据库以创建索引。 额外的索引和联立操作会严重影响性能和对系统资源（尤其是硬件资源）的使用需求。	重新编译源代码以验证对象关系。 类的析构函数可以很容易地处理相关联对象的删除工作。

增加多对多N:M的属性关系	在数据库存储方案中增加两个属性，相关联的两边各增加一个属性。 增加一个关联表，通过编写联立操作来将要建立连接关系的两边通过关联表连接起来，之后再通过映射形成内存对象。	需要在至少2个表中增加索引，形成一个“关联表”，以及针对每一个关系的索引。 需要执行数据库重组来创建索引。额外的表，索引和联立会严重影响性能和对系统资源（尤其是硬件资源）的使用需求。	重新编译源代码以验证对象关系。
增加多态复杂关系	在多态关系的情况下所有关联关系都会非常复杂，在执行联立之前必须检查每一个对象的类型。 需要有额外的代码来处理全部的联立操作。	根据不同的实现策略，需要考虑不同的问题。 如果不这样，作为面向对象技术的重要特性——多态关系就必须要被放弃。	重新编译源代码以验证对象关系。
删除属性	删除对目标属性的使用代码	删除根据层次化类定义中定义的所有表的指定列。	重新编译代码，运行schema工具。

数据存储领域缺乏敏捷特性会严重延迟开发速度

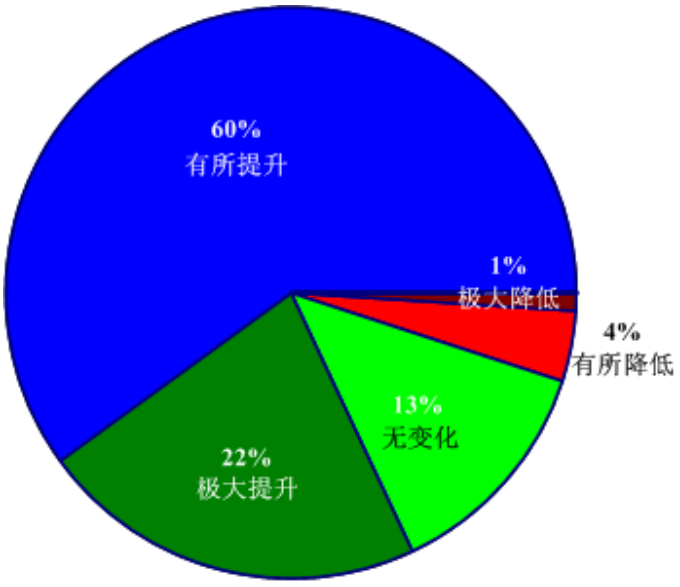
在数据库访问层实现和维护中缺乏敏捷开发特性明显会降低项目的整体开发速度。同时也会阻碍敏捷开发的基础，快速迭代周期，的顺畅进行。更进一步的，缺乏敏捷开发特性会阻碍来自用户的，针对新版本应用的快速反馈，这也从根本上磨灭了敏捷开发的另一个根本目标。

一个敏捷开发项目必须能够支持对领域对象的变更。理想的状态是，项目可以通过使用能够自动触发必要数据库和数据访问层变更的工具来实现这个能力，从而使得花费仅仅几分钟时间就可以实现数据库的演进。

在传统关系型数据库的条件下，对数据库存储方案和变更和数据的迁移可能会需要一辈子的时间——当然这是比喻——这当然与敏捷开发的现状是根本不匹配的。

对象持久，敏捷，以及对象数据库

诸如 Versant 对象数据库之类的 ODBMS 提供了诸如自动存储方案更新工具之类的工具，同时也提供数据迁移工具，这为程序开发工程师提供了极大的便利。而且，使用对象数据库也就避免了大量的对象关系映射工作。对象数据库以“本地”的方式管理着 对象。因此，在数据访问层实际上也就不需要任何代码。这也就是所谓的“透明对象存储”。对应用程序有关数据存储的任何变更都只需要通过重新编译应用程序的方式来完成，对象数据库自身会完成对应的映射工作。这些能力使得开发团队能够在任何时候完成代码的迭代周期，也就基本上不会对敏捷开发方法有任何的阻碍。



图表 2 敏捷开发对软件开发生产率的影响

对象关系映射软件，例如 Hibernate， JDO，或者 JPA，提供了一种相对敏捷的方案将应用程序对象映射到关系数据库中。这些 ORM 产品都是框架，都通过特定的元数据描述方案来使得开发工程师能够显式地定义大多数对象-关系映射。特定的变化，例如名字的变化或者从类中增加/删除简单属性可以相对很容易的通过对映射用的元数据文件的修改来实现，因此也是符合敏捷开发原则的。但是对于其它修改，例如修改应用程序的类结构，或者类层次或者类图关系，就可能会非常负责，并且往往不可能由 ORM 工具来自动完成。在这些情况下，ORM 工具虽然可能在程序接口方面支持敏捷开发， 但是确需要在底层的数据访问层进行额外的代码开发，同时还可能会需要执行诸如《数据库重构》一书中所描述的复杂的数据库重组或者重构。在这些情况下，即使使用 ORM，整个开发进程也必须等到数据库变更完成并测试通过之后，才能继续进行。

底线

虽然很明显项目可能会不同，商业模式可能会不同，具体情况也可能会不同，但是很显然的是当开发周期被缩短，迭代周期被加快，更多的客户反馈被及时地传递给开发人员，成本也必然会被节约下来，在同样的时间内所开发的产品也必然会更好。

敏捷软件开发的价值是很容易被理解的。然而，要判断敏捷软件开发和对象持久所带来的具体成本节约是困难的。以下的方法只能从某种程度上简单地就将数据库开发和敏捷软件开发过程集成所能够带来的潜在价值提供一些描述：

加快项目开发速度可以被理解为节省成本

如果使用敏捷开发会使整个团队的项目速度（开发速度）提高一倍，那么节约的成本就是原来开发周期内的项目开发团队的全部成本。例如，假如一个变更使得原来每个月花费 10 万美金，项目周期为 1 年 的开发团队的开发速度提高 1 倍，那么节约的成本约为 6 万美金（((12 个月 – 6 个月) x $100K/月)）。而且这还是假设项目立即开发结束，而没有持续性的维护和修改。在存在持续性维护和修改的条件下，成本节约还要明显。

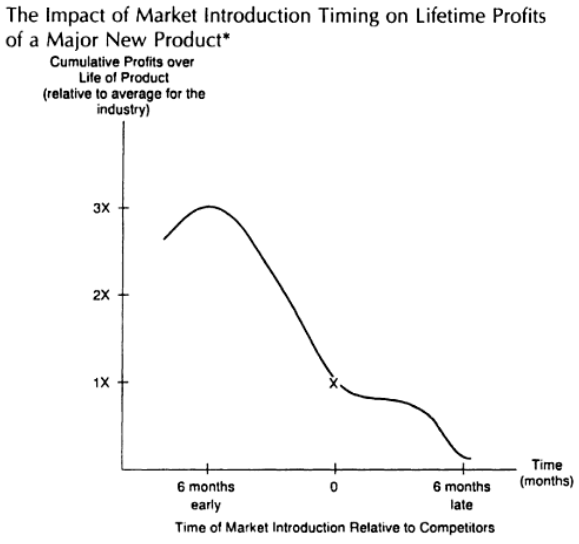
加快项目开发速度可以增加盈利能力

一个成功的敏捷开发项目可以带来以下两种主要财务优势：

它可以通过加快开发进程和降低总体开发周期来降低开发成本。

它可以通过在产品更新循环上的持续客户反馈使产品更加成功，不仅仅通过测试工具，而且还可以通过使用预发布的早期参与用户。

它可以通过加快上市时间来强化软件的商业价值，这可以体现在短期和长期的价格优势、竞争力优势、市场占有率优势等多方面。



软件项目的直接成本是最现实的，缩短开发周期一定能够节约大量的金钱。

敏捷开发的真正价值可以更加直接的被看作是“时间到成本”的转换关系以及客户对软件的更加有效的接受度。

知名专家 Steven Wheelwright 曾经估算过，一个软件产品提早六个月上市，可以在整个产品生命周期中带来三倍的利润。如果延期六个月上市，根据他的研究，成本和利润会持平。在这些假设的前提下，我们是可以很容易地想象出来，数据层的延迟将会如何使得产品的盈利能力大大降低。使用传统关系型数据库而不使用任何对象持久机制可能会造成产品在每个发布周期都会有几天到几个月的延迟，这使得在其它部分采用敏捷开发变得没有意义。反过来看，使用敏捷开发的对象持久机制可以加快敏捷开发进程，并且强化敏捷开发的固有优势，也就是更好的，更清晰的代码，更高的用户接受度以及更短的项目周期等等。

结论

本文很清晰地显示出在敏捷开发过程中如果需要对数据模型和存储方案进行修改，对象持久功能和能力，特别是使用对象数据库时，相比较于关系数据库能够提供巨大的优势。对数据库存储方案（以及对象）的修改和迁移的时间成本在使用对象数据库的情况下可以降到最低，尤其是相比较于在使用 ORM 或者不使用 ORM 的关系型数据库的条件下实现相同变更的情况。当然，在使用关系型数据库的情况下，使用 ORM 会带来一定的优势，可以使得使用敏捷开发方法的工程师能够尽可能的敏捷。然而，在整个项目开发周期中，在管理代码变更方面，即使 ORM 也无法与对象数据库相比。

一个打破陈规的方案是，一个程序员可能会考虑使用标准对象持久 API，例如 JDO，JPA 或者.NET ，并在项目开发阶段使用对象数据库，而在应用发布的时候切换到一个兼容的 ORM 框架上。通过这种方法，开发工程师可以充分利用对象数据库在加快项目开发周期方面的优势，而且同时还可以将其部属到他可能期望部属到的关系型数据库平台上。

参考材料

http://en.wikipedia.org/wiki/Agile_Manifesto

Results from Scott Ambler's February 2008 Agile Adoption Survey posted at www.agilemodeling.com/surveys/

Fowler, Martin. (1999) Refactoring: Improving the Design of Existing Code, Menlo Park, CA: Addison - Wesley Longman.

http://en.wikipedia.org/wiki/Waterfall_model

Ambler, Scott and Sadalage, Pramod (2006) Refactoring Databases: Evolutionary Database Design, Menlo Park, CA: Addison - Wesley Longman

Wheelwright, Steven C., Clark, Kim B., (1992) Revolutionizing Product Development, Simon & Schuster (page 22)

Barry & Associates, Object Relational Mapping, http://www.service - architecture.com/object - relational - mapping/articles/writing_your_own_mapping_layer.html

作者

Dirk Bartels is responsible for Strategic Product Management and Product Marketing for the Versant Object Database. Dirk is a true pioneer in the Object Database market. He conceived one of the first commercially available object database products and founded POET Software, which subsequently developed the FastObjects ODBMS. Dirk was an elected director at the Object Database Management Group (ODMG), and worked on ODBMS API standards for C++ and Java which evolved over the past few years into the Java community standards JDO (Java Data Objects) and JPA (Java Persistence API).

Robert Benson, Robert Benson is a consultant in middleware, distributed computing, and cloud computing. He has been part of industry‐leading efforts such as POJOs, object persistence, Jini, JavaSpaces, TurboPascal, and Smalltalk. He helps companies communicate clearly about technical software in a way that links the technical and the business issues. Robert lives in the San Francisco Bay area.

想要了解更多有关 Versant 产品组合、开发者信息或者其它的指引应用程序，请访问 www.versant.com, www.versant-china.com。

Versant 美国总部

Versant 公司总部

255 Shoreline Drive, Suite 450, Redwood City, CA 94065

电话: +1 650-232-2400, 传真: +1 650-232-2401

info@versant.com

Versant 欧洲总部

Versant GmbH

Wisesnkamp 22b, 22359 Hamburg, Germany

电话: +49.40.60990-0, 传真: +49.40.60990-113

info@versant.com

Versant 中国

恒尧信息

上海市昆明路 572 号 B 区 415-419 室

电话:+86-21-51721968, 传真:+86-21-51721967

info@versant-china.com

Versant 日本

TechMatrix Corporation

Tokyo, 140-0001 Japan

电话:+81-3-5792-8608

versant-sales@techmatrix.co.jp