УДК 004.415.2.043

АНАЛИЗ ВЛИЯНИЯ РАЗЛИЧНЫХ ПОДХОДОВ ОРГАНИЗАЦИИ СЛОЯ БИЗНЕС-ЛОГИКИ НА КАЧЕСТВА ПРОГРАММНОЙ РЕАЛИЗАЦИИ КОРПОРАТИВНЫХ ИНФОРМАЦИОННЫХ УПРАВЛЯЮЩИХ СИСТЕМ.

Власенко А.Н.
Харьковский национальный университет радиоэлектроники
Кафедра программного обеспечения ЭВМ
E-mail: vlasenko.alexander1986@gmail.com

Аннотация

Власенко А.Н. Анализ влияния различных подходов организации слоя бизнес-логики на качества программной реализации корпоративных информационных управляющих систем. *В статье оценивается влияние использования основных типовых решений организации слоя бизнес-логики (сценарий транзакции, модуль таблицы и модель предметной области) на достижение основных качеств программной реализации информационных управляющих систем, ориентированных на нужды бизнеса. Рассмотрены основные принципы и идеи базовых типовых решений, а также их достоинства и недостатки.*

Общая постановка проблемы. Современные бизнес - ориентированные приложения представляют собой весьма сложные программные системы. В сердце этих систем находится база данных, ведь зачастую хранимая и обрабатываемая информация является главным ресурсом предприятия.

Процесс проектирования является фундаментальной фазой процесса разработки программного обеспечения [1], и в ходе эволюционного развития принципов проектирования данных программных комплексов и систем возникла насущная необходимость в стандартизации не только отдельных архитектурных решений, но и архитектур в целом. Наибольшего распространения достигла архитектура, основанная на расслоении программной системы на три слоя: представление, домен (бизнес-логика) и источник данных [2]. Также существуют архитектурные модели с другим набором слоев, но всех в том или ином виде присутствует слой бизнес-логики [2]. В слое бизнес-логики сконцентрированы основные функции приложения, предназначенные для достижения поставленной перед ним цели.

Исторически сложились три основных варианта построения бизнес-логики в приложении: типовые решения сценария транзакции, модели предметной области и модуля таблицы. Выбор и обоснование выбора одного из них для конкретного приложения является сложной задачей для архитектора программного обеспечения (ПО). В более общем виде задачу можно сформулировать как проблемы определения и обобщения критериев, по которым следует принять выбор типового решения. Также следует упомянуть, что названные подходы не являются взаимоисключающими [2] и задача их комбинирования также является сложной проблемой в процессе разработки архитектуры.

Основным параметром, на который необходимо ориентироваться при выборе типового решения построения бизнес-логики является сложность программной системы. Проблема лишь в корректной оценке сложности проектируемой системы. Ведь она не поддается точной количественной и даже качественной оценке. Сложно сделать оценку, базируясь лишь на общей спецификации. Тем более следует учитывать возможность серьезных изменений требований к системе в будущем. Ведь после выбора архитектуры возможности ее изменения уменьшаются на каждой новой стадии/итерации процесса разработки, и ошибки, допущенные на данном этапе, обходятся особенно дорого.

Также на выбор влияют возможности используемых технологий и инструментальных средств, опыт разработчиков и доступность источника данных.

Целью исследования является анализ и оценка влияния выбора типового решения для построения слоя логики предметной области на различные внутренние и внешние качества программной системы и качества процесса разработки для программных инженеров и архитекторов программного обеспечения. В данной работе рассмотрены только те качества ПО, на которые применение исследуемых типовых решений влияет непосредственно – например, удобство работы пользователя не рассматривается, так как оно преимущественно зависит от слоя представления.

Обзор качеств ПО. Для инженерного подхода в построении программного обеспечения как части информационно-управляющей системы очень важна оценка различных качеств получаемой системы. Качества бывают внешние, которые видны пользователям, и внутренние, которые непосредственно касаются разработчиков. Внутренние качества, хоть и не видны пользователям, очень важны, так как внешние качества очень от них зависят, и иногда грань между ними очень тонка. Помимо качеств продукта, ценность представляют качества самого процесса разработки, которые являются базисом для достижения качеств продукта.

Корректность устанавливает отношение эквивалентности между программной системой и ее спецификацией функциональных требований. Неразрывно связанным с корректностью качеством является надежность – способность ПО работать корректно в течение продолжительного интервала времени. Устойчивость же означает способность системы правильно реагировать на непредусмотренные спецификацией действия пользователя или поведение внешних систем. Это важнейшие внешние свойства с пользовательской точки зрения.

Производительность отвечает за скорость обработки информации системе. Это внешнее свойство является производным от внутреннего свойства – эффективности, но в данной работе мы ограничимся рассмотрением производительности.

Верифицируемость – качество, которое определяет насколько легко протестировать систему. Это очень важное свойство, ведь приложение, которое невозможно протестировать, или процесс верификации которого предельно усложнен, обречено на провал.

Сопровождаемость и ремонтопригодность отвечают за простоту добавления новой и модификации старой функциональности в системе и исправление ошибок. В более широком смысле эти два свойства можно рассматривать как способность программной системы к эволюции.

Продуктивность – качество процесса, которое определяет скорость разработки. Связанным с ним качеством является своевременность – способность развернуть систему в поставленные сроки.

Анализ типовых решений. Рассмотрим упомянутые выше типовые решения.

Сценарий транзакции.

Логика предметной области разбивается на набор процедур, каждая из которых предназначена для обработки одного запроса из слоя представления. Каждой бизнес-транзакции ставится в соответствие собственный сценарий транзакции и все функциональные возможности полностью содержатся в отдельном методе.

В качестве примера можно привести часть автоматизированной системы автомобильного страхования. При данном подходе у нас будет класс с названием СтраховойСервис с методами создания нового клиента страховой компании, подсчета страховки для оформленного клиента, подсчета страховки в режиме консультации (без предварительно сохраненных данных клиента) и оформления страховки. Все операции с базой данных будут инкапсулированы в данном классе. Уже в этом простом примере видны зачатки проблем с дублированием в методах подсчета страховки.

Главным преимуществом данного типового решения является его простота. Также не следует опасаться работы параллельных транзакций.

Основным недостатком является дублирование кода – поскольку каждый сценарий транзакции решает только одну поставленную перед ним задачу, общие фрагменты кода неизбежно приходится копировать.

А теперь рассмотрим влияние на качества программного продукта:

– корректность, надежность и устойчивость достигаются сравнительно легко при наличии достаточно полной спецификации функциональных требований, покрывающей все бизнес-транзакции системы;

– производительность высокая, та как между получением запроса от слоя представления и ответом находится минимальная последовательность операций;

– на верифицируемость негативно влияет необходимость проверять лишний раз функциональность, дублированную в нескольких транзакциях. К плюсам можно отнести легкое покрытие тестами;

– на сопровождаемость (ремонтопригодность и способность к эволюции) позитивно влияет легкость определения участка кода, ответственного за логику, которую необходимо модифицировать в связи с изменением требований (способность к эволюции) или обнаружением ошибки (ремонтопригодность). Также к положительным аспектам можно отнести отсутствие скрытых зависимостей, что позволяет не беспокоится о неожиданных последствиях изменений. Негативно влияет многократное внесение изменений в случае наличия дублирующихся участков кода – растет вероятность ошибок и рассинхронизации спецификации и реализации программного продукта;

– продуктивность высокая для системы с несложной логикой, так как весь процесс разработки представляет собой банальную последовательность добавления реализаций бизнес-транзакций без внесения каких-либо архитектурных модификаций. Чем сложнее бизнес-логика, тем дольше длиться разработка с применением данного шаблона проектирования;

– своевременность выхода продукта на рынок (или поставки конкретному заказчику) относительно высокая, так как фаза проектирования предельно сокращается из-за простой структуры приложения и способности к поэтапному выпуску продукта. Нивелировать данные преимущества способна длительность фазы разработки, которая непропорционально возрастает при росте сложности бизнес-правил.

Модель предметной области.

Типовое решение «модель предметной области» базируется на сети взаимосвязанных объектов, каждый из которых представляет некоторую осмысленную сущность. В приложение добавляется богатый слой объектов для описания различных сторон бизнеса. Как правило, чем сложнее модель предметной области, тем больше различий в ее структуре с структурой базы данных. Сложные модели содержат иерархии наследования и, чем точнее модель предметной области представляет сложною и разветвленную бизнес-логику, тем труднее отобразить данные в реляционную базу данных.

Данное типовое решение предназначено для работы со сложной ухищренной бизнес логикой, предусматривающей сложные вычисления, проверки и ветвления.

В качестве примера рассмотрим всю ту же подсистему в автостраховании. Теперь у нас будут отдельные классы для сущностей автомобиля, клиента, страхового полиса и т.д. с соответствующими связями между ними. Мы можем использовать различные вспомогательные типовые решения для реализации самых изощренных пользовательских требований. Например, использовать типовое решение Стратегия [2] для выражения и комбинирования различных подходов при страховании автомобилей (с учетом износа или без, на одного пользователя или на многих и т.д.). Или мы можем применить типовое решение Состояние [3] для обеспечения возможности поэтапного оформления страховки

(допустим, изначально доступны не все документы, но процесс оформления начать возможно). Реализация подобных возможностей с другими типовыми решениями слоя предметной области была бы предельно усложнена.

К достоинствам можно причислить возможность работы с очень сложной бизнес-логикой, а также способность к масштабируемости и расширяемости.

Перечислим основные недостатки:

– Необходимость работы с сеансами и состояниями.

– Возможность чрезмерного увеличения количества объектов.

– Необходимость работы с набором сложных вспомогательных типовых решений (преобразователь данных, слой служб и т.д.).

– Необходимость использовать развитые средства ОО моделирования.

Влияние на качества программного продукта:

– корректность, надежность и устойчивость испытывают сильную зависимость не только от качества и полноты спецификации, но и большую, в сравнении с другими типовыми решениями организации бизнес-логики, зависимость от использования испытанных методологий, процессов и вспомогательных типовых решений и библиотек (к примеру, средств объектно-реляционного преобразования);

– производительность ниже из-за дополнительных промежуточных слоев. Часто возникает необходимость использовать типовые решения, призванные решить проблемы с производительностью (например «Загрузка по требованию» [2]), что в свою очередь усложняет структуру приложения еще больше. Также популярны решения, когда модель предметной области используется не во всем приложении, а только для наиболее сложных операций, когда для простых операций (к примеру отображения общей информации) используются менее нагруженные решения, но данный подход нельзя назвать чистым с архитектурной точки зрения.

– верифицируемость и сопровождаемость достигаются весьма тяжело из-за сложной структуры приложения, где вместе с прорехами проектирования появляются скрытые зависимости, которые могут создавать проблемы при модификациях кода системы. Поэтому в процессе разработки сложных систем с применением проектирования предметной области зачастую используются методы и практики разработки посредством тестирования, что в свою очередь требует дополнительных навыков и опыта от членов коллектива в частности и более высокой корпоративной культуры разработки в общем.

– продуктивность и своевременность при применении данного типового решения зависит от опыта разработчиков и уровня организации процесса разработки сильнее, чем при использовании других рассматриваемых типовых решений. Это непосредственно связано с перечисленными выше недостатками.

Модуль таблицы.

Данное типовое решение основано на максимальном совмещении структуры приложения с реляционной моделью используемой базы данных. Предусматривается конструирование по одному классу на каждую таблицу базы данных. В большинстве случаев модулю таблицы соответствует табличная структура данных. Для сохранения информации обычно используется множество записей (Record Set), заполняемое DML-запросом. От самого же модуля таблицы требуется инкапсулировать данные и предоставить методы их обработки.

Возможны ситуации, когда несколько модулей таблицы оперируют данными из одного множества записей. Также иногда модули таблицы проектируются не только для таблиц, но и для представлений и наиболее важных запросов.

Модуль таблицы может быть реализован как экземпляр класса так как и совокупность статических методов. Распространена реализация с применением шлюза таблицы данных [2].

В качестве примера использования модуля таблицы представим всю туже подсистему в автостраховании. Тут у нас каждой таблице в используемой реляционной базе данных (Клиенты, Автомобили, СтраховыеПолисы и т.д.) будет соответствовать модуль таблицы с необходимыми методами-оболочками, представляющими запросы к базе данных. Возможно создание модулей таблицы для наиболее важных представлений и виртуальных таблиц (к примеру для представления связывающего автомобили с клиентами) с необходимой логикой.

Главным достоинством данного типового решения является эффективное использование базы данных при сочетании данных и использующих их функций. Самым существенным недостатком смело можно назвать ограниченность использования объектно-ориентированного подхода к решению поставленных задач.

Влияние на качества программного продукта:

– корректность, надежность и устойчивость достаточно высокие и достигаются за счет использования проверенных временем СУБД и фреймворков доступа к данным.

– на производительность негативно влияет необходимость передавать большие объемы данных от сервера баз данных и повышенные требования к памяти для хранения множества записей.

– высокая верифицируемость и сопровождаемость достигается за счет общей простоты решения (в первую очередь простоты интерфейсов к реляционной базе данных). Найти и исправить проблему или внести модификацию довольно просто из-за очевидной структуры приложения.

– продуктивность и своевременность высоки на начальной стадии, как за счет недлительной фазы проектирования, так и за счет сбалансированной (по отношению со случаями применения других рассматриваемых типовых решений) длительности фазы разработки. С усложнением логики скорость разработки резко падает. Возможность поэтапного выхода продукта относительно ограничена по причине необходимости создания и верификации необходимых модулей таблицы даже для небольших частей функциональности, хотя последующее приращение функциональности упрощено.

Выводы. В ходе исследования были оценены последствия выбора типового решения слоя логики предметной области на качества продукта и процесса разработки. Была установлена основополагающая зависимость между сложностью бизнес-правил и достигаемыми качествами для каждого из рассмотренных типовых решений. Результаты могут быть полезными архитекторам и разработчикам сложных информационных систем, ориентированных на нужды бизнеса.

По результатам данного исследования перспективным направлением дальнейших исследований в данной области является разработка формального метода оценки сложности логики корпоративных информационных управляющих систем на базе спецификации.

Список литературы

1. Гецци, К. Основы инженерии программного обеспечения [Текст] / Карло Гецци, Мехди Джазайери, Дино Мандриоли. – СПб.: БХВ-Петербург, 2005. – 832 с.

2. Шаблоны корпоративных приложений [Текст] / Мартин Фаулер, Дейвид Райс и др.; под общ. ред. Мартина Фаулера. – М.: Вильямс, 2010. – 544 с.

3. Нильссон, Дж. Применение DDD и шаблонов проектирования. Проблемно-ориентированное проектирование приложений с примерами на C# и .NET [Текст] / Джимми Нильссон. – М.: Вильямс, 2008. – 560 с.