Лабораторная работа №2
«Стратегическое проектирование»

Цель работы

Познакомиться с ключевыми понятиями стратегического проектирования.

Задание для выполнения

В рамках выбранной в ЛР№1 предметной области:

1) разработайте единый язык, достаточный для описания выбранной функциональности;
2) опишите ограниченные контексты (не менее двух), используя разработанный вами единый язык;
3) разработайте и опишите карту контекстов и смысловое ядро.

Теоретические сведения

В данном пособии речь пойдет о предметно-ориентированном проектировании программного обеспечения с использованием, в первую очередь, стратегических шаблонов.

Данный подход использовал Вон Вернон в своей книге «Реализация методов предметно-ориентированного проектирования». Цель написания этой книги: дать возможность разработчикам совершить полет на самолете DDD (в детстве автор зачастую путешествовал со своей семьей на небольших самолетах). Вид с высоты дает более широкое представление о проблемах моделирования, не давая застрять в различных технических деталях. Наблюдая ландшафт DDD таким способом, можно осознать преимущества как стратегического, так и технического проектирования.

Полет осуществляется с помощью инструментов и шаблонов стратегического моделирования, таких как: **единый язык**, **предметная область**, **предметная подобласть**, **смысловое ядро**, **ограниченный контекст**, **карта контекстов**.

Проектирование с помощью тактических шаблонов, таких как, например, **сущность**, **объект**, **значение**, **репозиторий**, **событие**, **агрегат**, представляют собой облегченный DDD. Хотя очень важно уметь использовать этот подход, эти шаблоны имеют в основном технический характер, и, чтобы увидеть общую картину DDD, необходимо первым делом научиться применять стратегические шаблоны на практике.

Основной целью применения DDD является получение высококачественной модели программного обеспечения, которая будет

максимально точно отражать поставленные бизнес-цели. Для реализации этого требуется объединение усилий как разработчиков, так и экспертов в предметной области. Создание дружной и сплоченной команды позволяет получить большое количество преимуществ для бизнеса. Обмен знаниями между членами команды снижает шансы появления «тайного знания» о модели, достигается консенсус между экспертами предметной области в отношении различных понятий и терминологии, разрабатывается более точное определение и описание самого бизнеса.

Для того чтобы уравнять разработчиков и экспертов предметной области, чтобы было гораздо проще обмениваться полезными знаниями о предметной области, подход DDD предлагает применять общий набор терминов, понятий и фраз, который будет использоваться в общении между членами команды, и который позже отразится в исходном коде результирующей программы.

Единый язык

Этот коллективный язык терминов называется **единый язык** (Ubiquitous Language). Это один из основных и самых важных шаблонов предметного-ориентированного проектирования. Это не бизнес жаргон, навязанный разработчикам, а настоящий язык, созданный целостной командой – экспертами в предметной области, разработчиками, бизнес-аналитиками и всеми, кто вовлечен в создание системы. Роль в команде не столь существенна, поскольку каждый член команды использует для описания проекта единый язык. Процесс создания единого языка более творческий чем формальный, так как он, как и любой другой естественный язык, постоянно развивается, а те артефакты, которые вначале способствовали разработке полезного единого языка, со временем устаревают. В итоге, остаются только самые устойчивые и проверенные элементы. Чтобы перейти от экспериментов к точным знаниям, Вон Вернон в своей книге предлагает использовать следующие способы:

- Создание диаграмм физической и концептуальной предметной областей и нанесение на них меток, обозначающих имена и действия. Такие диаграммы носят неформальный характер, поэтому нет смысла использовать формальное моделирование (как UML).
- Создание глоссария с простыми определениями, а также альтернативными терминами с оценкой их перспективности. Таким образом разрабатываются устойчивые словосочетания единого языка.
- Создание документации вместо глоссария. Эта документация будет содержать неформальные диаграммы или важные понятия, связанные с программным обеспечением.
- Обсуждение готовых фраз с остальными членами команды, которые не могут сразу освоить глоссарий или другие письменные документы.

Так как код будет развиваться достаточно быстро и его необходимо согласовывать с текущим вариантом единого языка, то можно позже отказаться от диаграмм, глоссария или другой документации.

Наиболее важное для разработчика – это умение слушать экспертов, получать максимальное количество полезных знаний о предметной области. В то же время, эксперты также должны прислушиваться к разработчикам и их пожеланиям. Команда учится и растет вместе, если она действует сплоченно, получая более глубокое понимание бизнеса.

В команде, обсуждая модель применения вакцины от гриппа в виде кода, произносят фразу наподобие: «Медсестры назначают вакцины от гриппа в стандартных дозах». Возможные варианты развития событий:

Возможные точки зрения	Итоговый Код
«Кого это волнует? Просто программируйте»	Patient.setShotType(ShotTypes.TYPE_FLU); patient.setDose(dose); patient.setNurse(nurse);
«Мы делаем пациентам прививку от гриппа»	patient.giveFluShot();
«Медсестры назначают вакцины от гриппа в стандартных дозах»	Vaccine vaccine = vaccines.standartAdultFluDose(); nurse.administerFluVaccine(patient,vaccine);

Очевидно, что первый вариант совсем не оптимален, второй – лучше, но не учитывает некоторые важные концепции. Окончательный вариант наиболее приемлем для поставленной задачи.

Зачем нужен единый язык?

Порой, в нашей профессии, сформулировать требование к продукту бывает сложнее, чем реализовать его. Разные предметные области сложны по своему. В перевозках вы столкнетесь с концепцией овербукинга, в продажах со «специальными предложениями», в трейдинге — со «стоп-лосс трейлинг ордерами».

Единый язык — это базовая концепция борьбы со сложностью. Если предметная область итак требует изучения, то давайте не будем делать вещи еще сложнее и заставлять людей разбираться еще и в вашей модели предметной области. Будем называть вещи одинаково как в спецификации, так и в коде. Пример ниже наглядно иллюстрирует использование этой концепции:

Единый язык **on**

var ticket = cashdesk.Sale(seat, count); // Это может понять даже не технический специалист

Единый язык **off**

```
ticket.State = TicketState.Sold; // А это нет
ticketRepository.Update(ticket);
```

Проблемы с отсутствием единой терминологии я ощутил на собственной шкуре. Команда разрабатывала систему для продажи билетов на мероприятия онлайн. Новая версия продукта была призвана заменить старую, расширить функционал и исправить ошибки проектирования первой версии. Один из крупных косяков первой системы состоял в допущении, что можно продать только один билет на место. Но оказалось, что существуют «танцевальный партер» и «столик на 5 человек», которые удобно представлять одним местом в зале.

Так у программистов появилось абстрактное понятие «продаваемое место», связывающее место в зале, с количеством билетов, которое можно продать на это место. Для обычных залов с рядами и местами оно было равно единице, а для танцевального партера — ограничивалось вместимостью зала.

Заметили как сложно воспринимается предыдущий абзац?

С технической точки зрения проблема была решена идеально, но каждому следующему программисту приходилось объяснять, что такое продаваемое место, чем отличается от квоты и почему в UI это называется билет. В итоге эта путаница вылилась в очень неудачный промах при оценке сложности спринта и спринт был провален.

Ограниченный контекст

Очень важно понимать, что в рамках предметной области смысл определенного термина или фразы может сильно отличаться. Существует некая граница, в пределах которой понятия единого языка имеют вполне конкретное контекстное значение.

Эта концептуальная граница называется **ограниченный контекст** (Bounded context). Это второе по значимости свойство DDD после единого языка. Оба эти понятия взаимосвязаны и не могут существовать друг без друга.

Итак, ограниченный контекст – это явная граница, внутри которой существует модель предметной области, которая отображает единый язык в модель программного обеспечения.

В каждом ограниченном контексте существует только один единый язык.

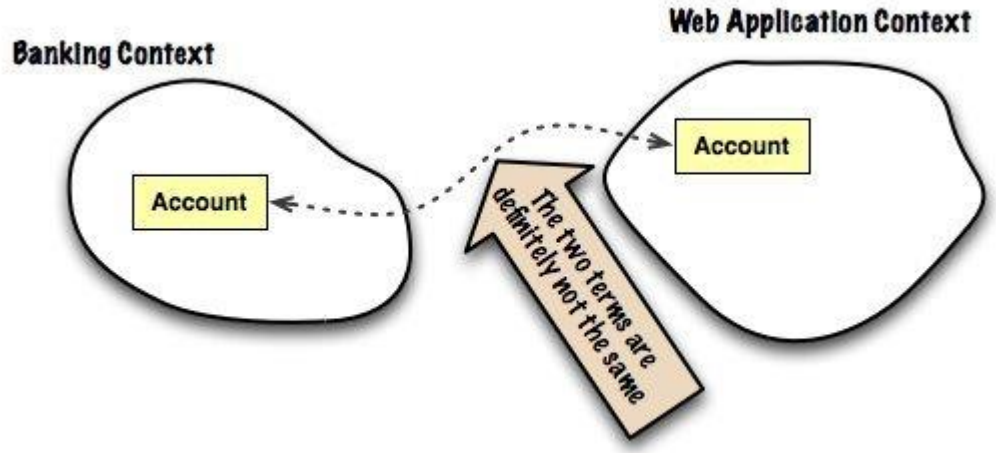
Ограниченные контексты являются относительно небольшими, меньше чем может показаться на первый взгляд. ограниченный контекст достаточно велик только для единого языка изолированной предметной области, но не больше.

Единый значит «вездесущий» или «повсеместный», т. е. язык, на котором говорят члены команды и на котором выражается отдельная модель предметной области, которую разрабатывает команда.

Язык является единым только в рамках команды, работающей над проектом в едином ограниченном контексте.

Попытка применить единый язык в рамках всего предприятия или что хуже, среди нескольких предприятий, закончится провалом.

Для примера, следует рассмотреть контраст между терминами account в контексте банковских услуг и в контексте веб-приложения:



Контекст банковских услуг

Account – счёт – поддерживает запись о дебиторских и кредиторских транзакциях, отображающих текущее финансовое состояние клиента с точки зрения банка

Контекст веб-приложения

Account – учётная запись пользователя в веб-приложении

Различить типы аккаунтов невозможно по именам. Но их можно различить исключительно по названиям их концептуальных контейнеров, т.е. соответствующих ограниченных контекстов.

Эти ограниченные контексты относятся к разным предметным областям. В следующем примере, одно и то же имя используется в пределах одной и той же предметной области.

В некоторых издательствах книги проходят разные этапы жизненного цикла, то есть книга проходит различные контексты.

- *Разработка концепции и предложения к изданию*
- *Контракт с автором*
- *Управление редакционным процессом*
- *Разработка макета книги, включая иллюстрации*
- *Перевод книги на другие языки*
- *Выпуск бумажных и/или электронных версий книги*
- *Маркетинг*

- *Продажа книги реализаторам и/или непосредственно покупателям*
- *Поставка экземпляров книги реализаторам или покупателям*

В этом случае, не существует единственного способа разработки правильной модели книги. Например, до заключения контракта, название книги остается неопределенным и может измениться в процессе редактирования. Для маркетинга не нужно большинство из артефактов литературной и технической редакций, за исключением обложек и аннотаций, а для продажи книги необходимы только название, расположение склада, количество экземпляров, размер и вес. Чтобы избежать путаницы, в рамках подхода DDD необходимо использовать отдельные ограниченные контексты для каждой из стадий жизненного цикла. Модель книги в каждом контексте значительно отличалась бы от всех других. Каждая команда определенного ограниченного контекста говорит о книге и знает точно, чего хочет для своего контекста.

В примере выше используются различные ограниченные контексты в рамках одной предметной области.

Зачем нужны контексты?

Проблема №1

К сожалению, преимущества единого языка мы не получаем бесплатно. Единый язык требует кропотливого анализа требований и консультаций с экспертами предметной области. Зачастую, так называемые эксперты озвучивают противоположные точки зрения о бизнес-процессах и терминах. Разрабатывая приложения в рамках «единого языка» вы инвестируете много времени в создание внутренней базы знаний клиента. Безусловно это хорошо. Проблема в том, что почти наверняка эта работа не заложена в бюджет на разработку приложения. Разработка по всем канонам DDD и с применением единого языка — дорогое и не быстрое удовольствие.

Проблема №2

«Единый язык» на самом деле не является «единым» для всего приложения и живет только в рамках контекста. И Эванс прямо об этом пишет. К сожалению, в книге сей факт формируется мягко, скорее как рекомендация, а не правило. Я бы набрал параграф красным КАПСОМ и поместил в рамку. Попытка создания единой модели предметной области обречена на провал. Один и тот же термин может значить разные вещи в разных контекстах. Я могу сходу назвать несколько примеров из разных предметных областей, но все они требуют специального объяснения, поэтому ограничусь синтетическим примером: салат рекурсивный — помидоры, огурцы, салат. Ну вы поняли…

Проблема №3

Никому не нужна модель предметной области целиком. Она скорее всего слишком сложна даже для крутых стратегов и аналитиков. Эти ребята попросят представить им данные в совершенно ином, понятном им, виде и разрезе. Контексты реализуют принцип «разделяй и властвуй», что помогает быстрее обучать пользователей и подключать новых членов в команду разработки.

Предметная область, предметная подобласть, смысловое ядро

Предметная область (Domain) – это то, что делает организация, и среда, в которой она это делает. Разработчик программного обеспечения для организации обязательно работает в ее предметной области. Следует понимать, что при разработке модели предметной области необходимо сосредоточиться в определенной подобласти, так как практически невозможно создать единственную, всеобъемлющую модель бизнеса даже умеренно сложной организации. Очень важно разделять модели на логические разделенные **предметные подобласти** (Subdomain) всей предметной области, согласно их фактической функциональности. Подобласти позволяют быстрее определить разные части предметной области, необходимые для решения конкретной задачи.

Также необходимо уметь определять **смысловое ядро** (Core domain). Это очень важный аспект подхода DDD. Смысловое ядро – это подобласть, имеющая первостепенное значение для организации. Со стратегической точки зрения бизнес должен выделяться своим смысловым ядром. Большинство DDD проектов сосредоточены именно на смысловом ядре. Лучшие разработчики и эксперты должны быть задействованы именно в этой подобласти. Большинство инвестиций должны быть направлены именно сюда для достижения преимущества для бизнеса и получения наибольшей прибыли.

Если моделируется определенный аспект бизнеса, который важен, но не является смысловым ядром, то он относится к **служебной подобласти** (Supporting subdomain). Бизнес создает служебную подобласть, потому что она имеет специализацию. Если она не имеет специального предназначения для бизнеса, а требуется для всего бизнеса в целом, то ее называют **неспециализированной подобластью** (Generic subdomain). Эти виды подобластей важны для успеха бизнеса, но не имеют первоочередного значения. Именно смысловое ядро должно быть реализовано идеально, поскольку оно обеспечивает преимущество для бизнеса.

Это и есть основа для стратегического проектирования при подходе DDD.

Пространство задач и пространство решений

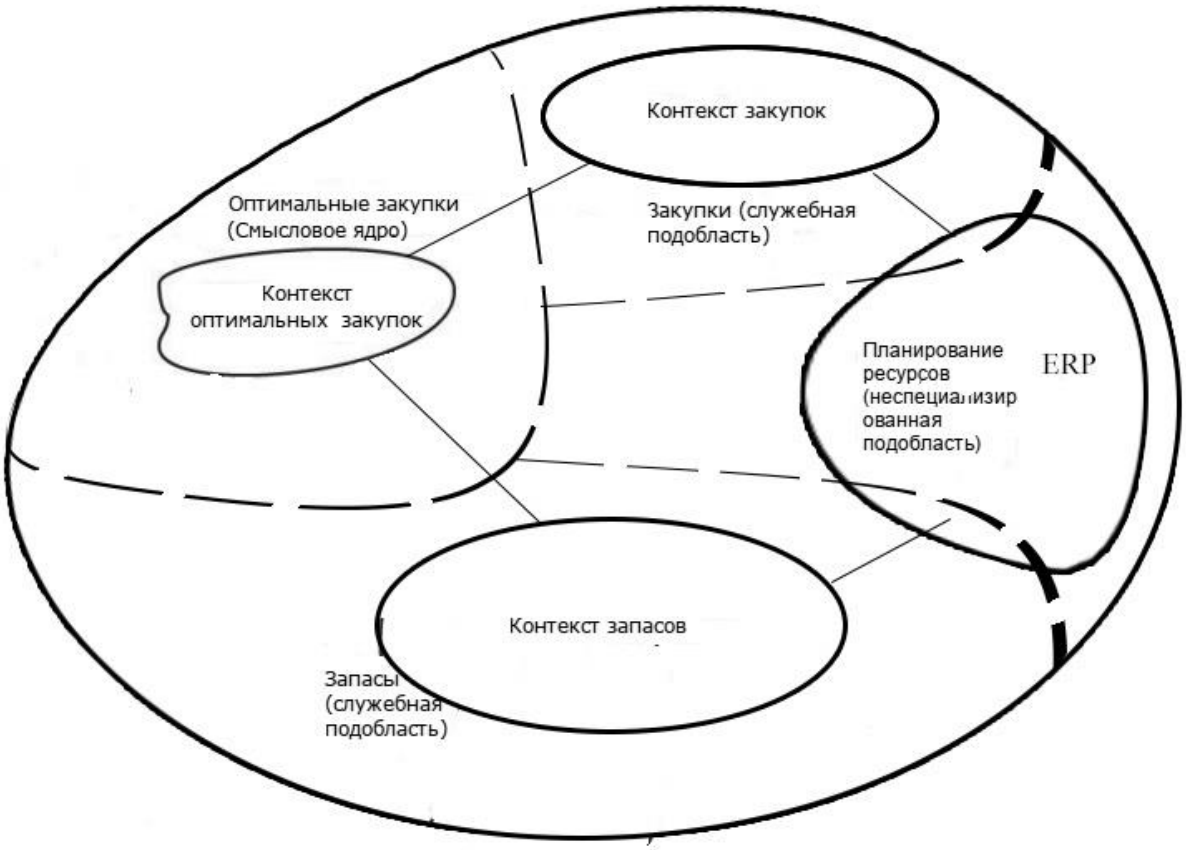
Предметные области состоят из пространства задач и пространства решений. Пространство задач позволяет думать о стратегической бизнес проблеме, которая должна быть решена, а пространство решений, сосредоточится на том, как реализуется программное обеспечение, чтобы решить бизнес проблему.

Пространство задач – части предметной области, которые необходимы, чтобы создать смысловое ядро. Это комбинация смыслового ядра и подобластей, которое это ядро должно использовать.

Пространство решений – один или несколько ограниченных контекстов, набор конкретных моделей программного обеспечения. Разработанный ограниченный контекст – это конкретное решение, представление реализации.

Идеальным вариантом является обеспечение однозначного соответствия между подобластями и ограниченными контекстами. Таким образом, объединяются пространство задач и пространство решений, выделяются модели предметной области в четко определенные области в зависимости от поставленных целей. Если система не разрабатывается с нуля, она часто представляет собой большой комок грязи, где подобласти пересекаются с ограниченными контекстами.

В качестве примера в книге Вона Вернона приводится смысловое ядро для небольшой компании, которая занимается розничной продажей в Интернете. Любой интернет-магазин может улучшить свое положение, если будет использовать механизм прогноза, который будет анализировать историю продаж и данные о запасах для получения прогноза спроса и конкретных объемов оптимальных запасов. (Компания не должна тратить деньги на товары, которые не имеют спроса и занимают дополнительную складскую площадь.) Именно это смысловое ядро делает организацию гораздо более конкурентоспособной, способной быстро идентифицировать выгодные сделки и гарантировать необходимый уровень запасов. Пространство задач состоит из смыслового ядро и подобластей, связанных с закупкой и запасами, и которые отображены на этом рисунке:



Это только часть предметной области, а не вся предметная область, в которой работает организация. Пространство задач (смысловое ядро и подобласти) необходимо проанализировать. Модель закупок, изображенная на рисунке, представляет собой решение для смыслового ядро. Модель предметной области будет реализована в явном ограниченном контексте: контексте оптимальных закупок. Этот ограниченный контекст однозначно соответствует одной подобласти под названием смысловое ядро оптимальных закупок. Еще один ограниченный контекст под названием контекст закупок будет разработан для уточнения технических аспектов процесса закупок и будет играть вспомогательную роль по отношению к контексту оптимальных закупок. Они обеспечивают взаимодействие с открытым интерфейсом системы ERP. Контекст закупок и модуль закупок ERP объединяются в служебную подобласть закупок. Сам модуль ERP является неспециализированной подобластью, поскольку ее можно заменить на любую другую коммерческую систему закупок. Однако, она становится служебной подобластью, если ее рассмотреть в сочетании с контекстом закупок в подобласти закупок. В соответствие с рисунком, контекст оптимальных закупок должен также взаимодействовать с контекстом товарных запасов, который управляет единицами хранения. Он использует модуль товарных запасов ERP, который находится в пределах служебной подобласти товарных запасов. ERP-приложение состоит из различных модулей, которые мы рассматриваем как логические подобласти, – это

подобласти товарных запасов и закупок. Эти модули и контексты запасов и закупок объединяются в служебные подобласти.

Итак, для оценки пространства задач в первую очередь необходимо:

- Определить, как выглядит стратегическое смысловое ядро
- Какие концепции должны рассматриваться как часть смыслового ядра
- Перечислить служебные и неспециализированные подобласти.

Оценка пространства задач влияет на оценку пространства решений. Здесь необходимо думать уже о существующих и новых системах и технологиях. Надо думать в терминах четко отделимых физических органических контекстов, для которых ищется единый язык. Следует:

- Проанализировать существующее программное обеспечение на предмет повторного использования
- Определить средства, которые следует приобрести или создать
- Изучить интеграцию этих средств
- Определить, как перекрываются концепции и данные разных ограниченных контекстов
- Определить, какой ограниченный контекст содержит концепции, относящиеся к смысловому ядру, и какие тактические шаблоны используются для его моделирования.

Ограниченный контекст состоит не только из модели предметной области. Хотя модель это ее основной компонент. Если создается схема баз данных из модели, она также участвует в этом ограниченном контексте. В то же время, если схема существовала ранее и в ней нарушен проект, эта схема не относится к ограниченному контексту.

Пользовательский интерфейс, который выражает модель, также относится к ограниченному контексту. Также могут быть реализованы сервис-ориентированные конечные точки, которые также находятся внутри границы. И компоненты пользовательского интерфейса, и сервис-ориентированные конечные точки делегируются прикладным службам, которые действуют как фасад по отношению к модели. Эти службы также находятся внутри границы ограниченного контекста.

Размеры ограниченных контекстов могут быть самыми разными. Главное, – чтобы модель демонстрировала богатство единого языка в контексте. В ней должно существовать столько концепций предметной области, сколько необходимо для моделирования в пределах ограниченного контекста – ни больше, ни меньше. Для того чтобы не пропустить ничего существенного, необходимо найти четкий и хороший критерий. Для этого можно использовать карту контекстов. Таким образом мы подходим к третьему важному шаблону стратегического проектирования.

Карта контекстов

Следуя подходу DDD, определенная команда должна создать собственную карту, которая отражает пространство решений, в которой находится эта команда. Эта карта состоит из ограниченных контекстов, а также интеграционных связей между ними. Пример:

Эта **карта контекстов** (Context map) отображает текущее положение дел, а не то, что будет в будущем. Необходимо избегать формальностей в процессе создания карты. Слишком большое количество деталей только мешает процессу создания.

Естественный ход событий – совпадение границ контекстов с организационным делением команды. Люди, работающие вместе, разделяют один общий контекст модели.

После создания предварительной карты контекстов, ее можно детализировать путем определения отношений между контекстами.

Существуют такие отношения между ограниченными контекстами и отдельными командами проекта:

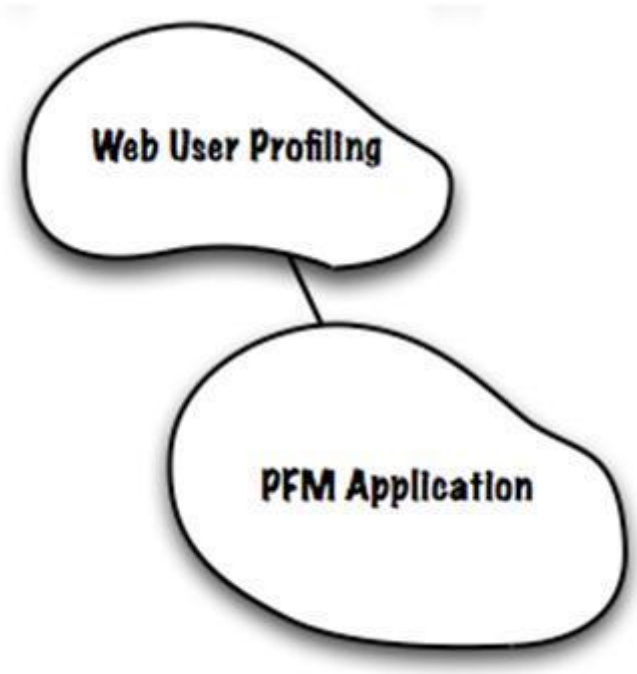
- партнерство (Partnership). Когда команды в двух контекстах достигают успеха и терпят неудачу вместе, возникает отношение сотрудничества. Они должны сотрудничать в процессе эволюции своих интерфейсов, чтобы учитывать потребности обеих систем.
- общее ядро (Shared kernel). Общая часть модели и кода образует тесную взаимосвязь. Обозначается четкая граница подмножества модели предметной области, которую команды согласны считать общей. Ядро должно быть маленьким. Оно не может изменяться без консультации с другой командой. Необходимо согласовывать единый язык команд.
- разработка заказчки-поставщик (Customer-supplier development). Когда две команды находятся в отношении «нижестоящий и вышестоящий», и команды вышестоящие учитывают приоритеты нижестоящих команд.
- конформист (Conformist). Когда две команды находятся в отношении «вышестоящий и нижестоящий», причем вышестоящая команда не имеет причин учитывать потребности нижестоящей команды. Нижестоящая команда учитывает сложность трансляции между ограниченными контекстами, беспрекословно подчиняясь модели вышестоящей команды.
- предохранительный уровень (Anticorruption layer). Если управление и коммуникация не соответствуют общему ядру, партнеру, или отношению «Заказчик-поставщик», то трансляция является сложной. Нижестоящий клиент должен создать изолирующий слой, чтобы обеспечить свою систему вышестоящей системы в терминах своей модели предметной области. Этот уровень общается с другой системой с помощью

существующего интерфейса, не требуя или почти не требуя модификаций другой системы.

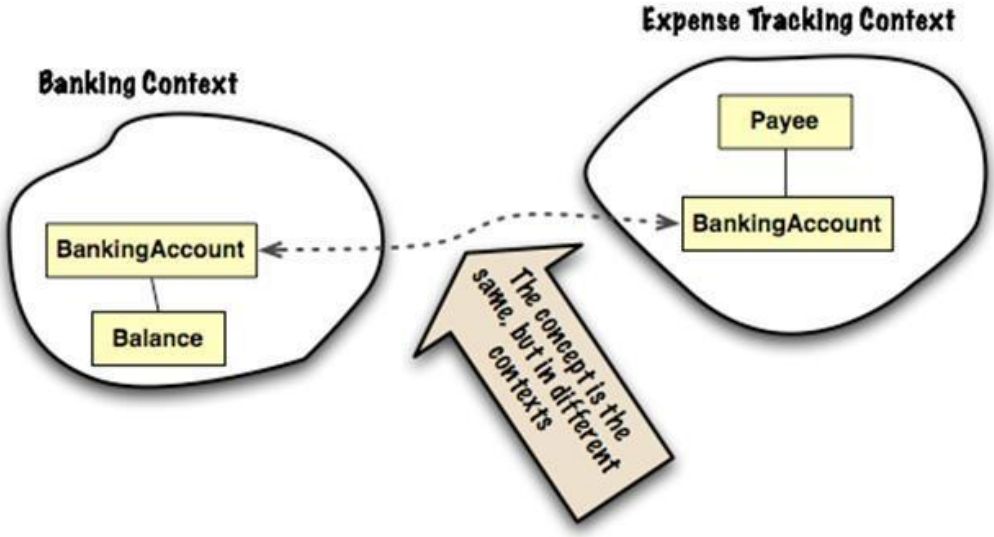
- служба с открытым протоколом (Open host service). Определяется протокол, который предоставляет доступ к системе как к набору служб. Для учета новых требований интеграции этот протокол расширяется и уточняется.
- общедоступный язык (Published language). Трансляция между моделями двух ограниченных контекстов требует общего языка. В качестве среды для коммуникации используется хорошо документированный общий язык, который может выразить необходимую информацию о предметной области, выполняя при необходимости перевод информации с другого языка на этот.
- отдельное существование (Separate ways). Если между двумя наборами функциональных возможностей нет важного отношения, их можно полностью отсоединить друг от друга. Интеграция всегда дорого стоит, а выгоды бывают незначительны.
- большой комок грязи (Big ball of mud). Существуют части системы, в которых модели перемешаны, а границы стерты. Необходимо нарисовать границу такой смеси и назвать ее большой комок грязи.

Пример разработки карты контекстов можно взять из статьи Альберто Брандолини Strategic Domain Driven Design with Context Mapping.

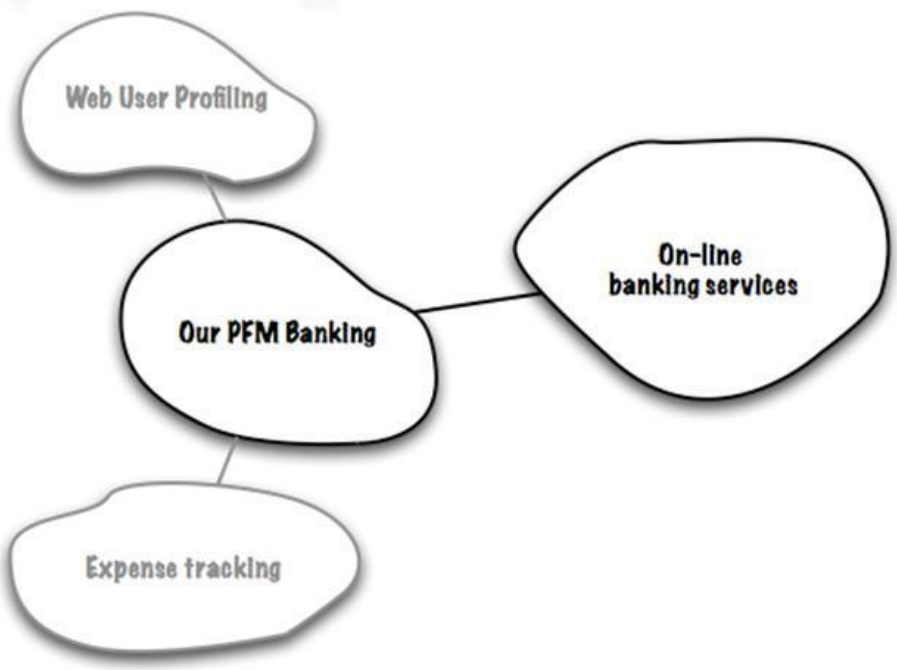
Сначала рисуется простая карта контекстов с границами и связью между ограниченными контекстами:



В этих двух контекстах есть различия в концепциях с одинаковым названием. Например, Account в Web User Profiling – это учетная запись пользователя (логин и пароль). В то же время, для PFM Application (персональное управление финансами) – это сводка, описывающая текущее состояние клиента с точки зрения банка. Иногда, как было указано выше, одна и та же концепция может использоваться в абсолютно разных контекстах, тем самым для них необходимо определить разные модели.

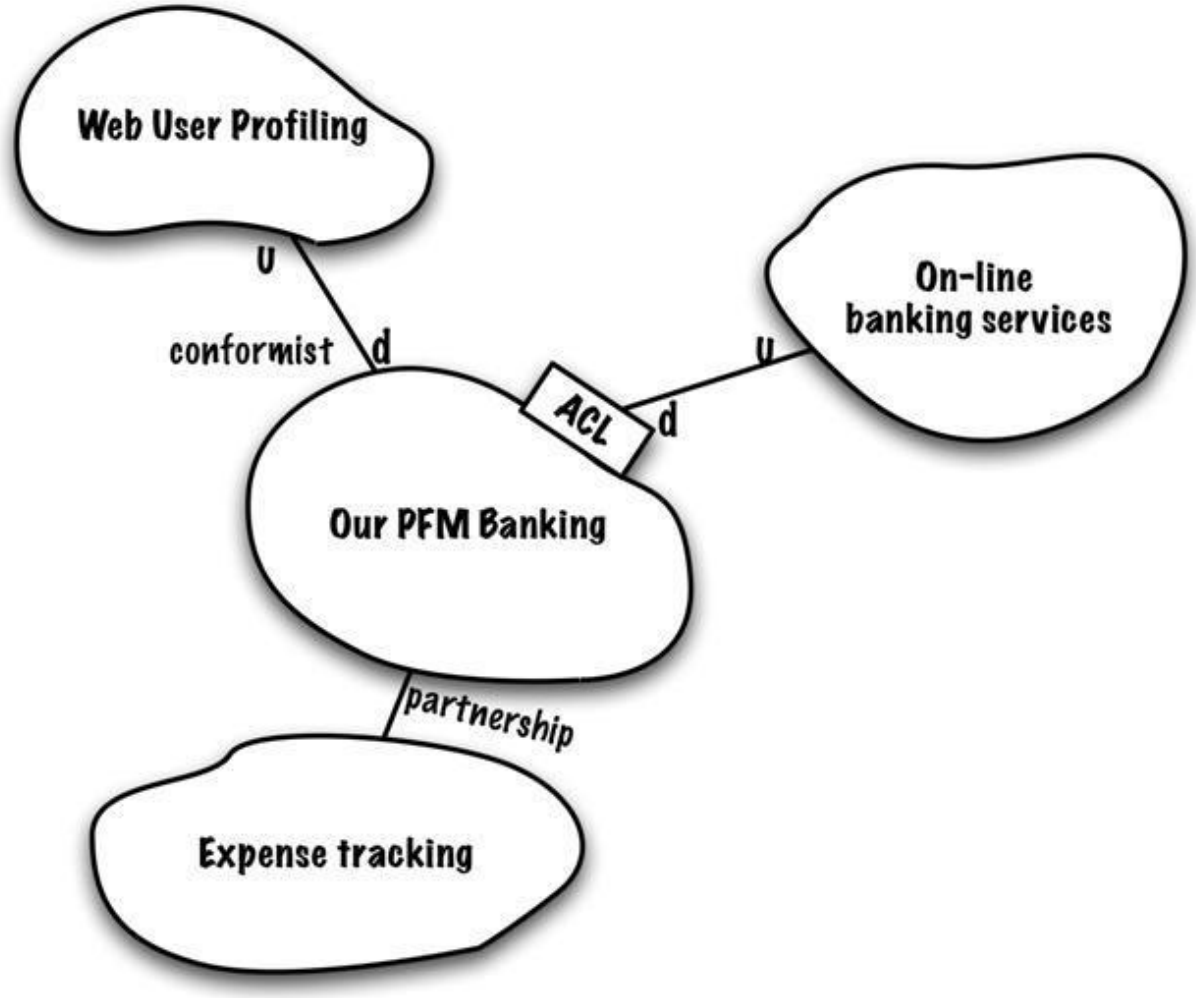


Например, PayeeAccount – это тот же BankingAccount, но с другим поведением (нельзя получить баланс). Таким образом будет создан отдельный контекст учета расходов (expense tracking). Также отдельно, в приведенном примере, создается контекст онлайн сервисов банка (on-line banking services) (такие сервисы, например, как распечатка выписок банка).



После первого шага создания карты, можно детализировать все отношения. У каждого отношения есть направление. Вышестоящие (upstream) влияют на нижестоящие (downstream), но не факт, что обратное верно.

Детализированная карта выглядит вот так:



Контекст банковских онлайн сервисов предоставляет API (это может быть служба с открытым протоколом и общедоступный язык). При изменении этого API, контекст персонального управления финансами должен тут же измениться, чтобы работать с новым API. При этом необходимо использовать предохранительный уровень, чтобы не дать понятиям из контекста онлайн сервисов просочиться в контекст персонального управления финансами (делается преобразование моделей предметной области).

Так как контекст профайлов веб-пользователей используется как готовый внешний модуль и он поставляется «как есть», здесь

устанавливается отношение конформист (нижестоящий подчиняется вышестоящему).

В случае с контекстом учетов расходов, то здесь лучше всего подходит отношение партнерства, так как существуют общие цели и концепции, но нет направления отношения.

Это простой пример карты контекстов, на деле они бывают намного сложней.

Список использованных источников

1. С.Грибняк. Domain-Driven Design: стратегическое проектирование. Часть [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/316438/. – Дата доступа: 03.02.2020.
2. М.Аршинов. Ubiquitous Language и Bounded Context в DDD [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/232881/. – Дата доступа: 03.02.2020.
3. Вернон, В. Реализация методов предметно-ориентированного проектирования. : Пер. с англ. – М. : ООО «И.Д. Вильямс», 2016. – 688 с. : ил. – Парал. тит. англ.