

Методологии проектирования ПО

Владимир ДЗалбо. Использование паттернов с RUP



**ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Факультет Автоматики и Вычислительной Техники**

Методологии проектирования ПО

Использование паттернов с RUP.

Владимир Дзалбо

0. Содержание

| | |
|---|-----------|
| 0. СОДЕРЖАНИЕ | 2 |
| 1. ВВЕДЕНИЕ | 3 |
| 2. ОСНОВНАЯ ЧАСТЬ | 5 |
| 2.1. КЛАССИФИКАЦИЯ ПАТТЕРНОВ | 5 |
| 2.2. RATIONAL UNIFIED PROCESS | 9 |
| 2.3. ИСПОЛЬЗОВАНИЕ IBM ПАТТЕРНОВ | 12 |
| 2.3.1. Паттерны в отправной точке (<i>Inception</i>) | 12 |
| 2.3.2. Паттерны в выработке (<i>Elaboration</i>) | 18 |
| 2.3.3. Паттерны в конструировании (<i>Construction</i>) | 19 |
| 3. ЗАКЛЮЧЕНИЕ | 20 |
| 4. ЛИТЕРАТУРА | 21 |

1. Введение

Для начала необходимо познакомиться с термином "паттерн проектирования", для того чтобы можно было впоследствии говорить о тех преимуществах и достоинствах, которые открывает перед программистами методика шаблонного проектирования. Пожалуй, каждому разработчику ПО знакома ситуация, когда при проектировании дизайна сложной системы было трудно подобрать какое-то готовое и простое решение, вынуждая использовать громоздкие и порой малоэффективные схемы программирования.

Одним из основных принципов "хорошего" дизайна программы является простота. Так, анализируя процесс создания ПО, Гради Буч ввел такое понятие, как "отрицательная производительность" – точку во времени разработки проекта, после достижения которой число строк кода в продукте начинает уменьшаться, в то время как функциональность растет. На самом деле, существует огромное количество методологий и рекомендаций, направленных на упрощение проектируемых систем. Среди них можно выделить такие известные методики как RUP (Rational Unified Process), XP (eXtreme Programming), MSF (Microsoft Solution Framework) и др. Однако, в основе всех этих методологий лежит универсальная методика – принцип повторного использования, представленный техникой шаблонного проектирования.

В классической формулировке понятие "шаблон проектирования" – это описание взаимодействия объектов и классов, адаптированных для решения общей задачи проектирования в конкретном контексте. Кристофер Александер ввел свое определение паттернов: «Каждый шаблон представляет собой правило, состоящее из трех составляющих, которые выражают отношения контекста, проблемы и решения». Есть более общее определение, по Мартину Фаулеру: «Паттерн есть идея, которая однажды была практически реализована и, вероятно, может быть использована в будущем». Все эти определения довольно точно характеризуют смысл понятия паттерна проектирования, рассматривая его в контексте использования объектно-ориентированных принципов, таких как инкапсуляция, наследование и полиморфизм.

В реальном виде, паттерны представлены набором объектов и классов, которые определенным образом взаимосвязаны между собой. Конкретная реализация шаблона проектирования направлена на непосредственное решение поставленной задачи по упрощению дизайна и проектированию эффективной и доступной системы. В общем случае паттерн состоит из четырех основных элементов:

- **Название.** Название паттерна позволяет использовать паттерны на более высоком уровне абстракции, составляя специальные каталоги и словари паттернов.
- **Задача.** Паттерны используются только в строго разграниченном контексте определенной задачи. Такая изолированность помогает сконцентрироваться на поставленной проблеме.

- Решение. В данном случае это общее описание элементов дизайна, отношений между ними и функций каждого элемента.
- Результаты. Применение каждого паттерна ориентировано на решение определенной проблемы, в результате чего программист получает мощный инструмент многократного использования.

Рассматривая проблему в контексте паттернов проектирования, программисты не только упрощают дизайн будущей системы, но и решают такие дополнительные вопросы, как оптимизация программного кода, самодокументирование самого продукта и использование единого стиля программирования.

Паттерны проектирования позволяют решать разнообразные задачи. Спектр их применения велик и неограничен. В основе использования паттернов лежат принципы функционирования сложных систем. На основе этих принципов паттерны предоставляют программистам следующие возможности:

- - решать "реальные" проблемы и задачи;
- - накапливать и реализовывать полученный опыт;
- - принимать решение на основе логического обоснования;
- - передавать накопленный опыт другим;
- - формировать общий словарь программных решений;

Принимая во внимание специфику использования паттернов проектирования и рассматривая главные составляющие процесса разработки программного обеспечения, нельзя забывать и тот факт, что не всякое решение требует использования паттернов. Приведенные ниже ситуации относятся к их числу:

- - решения для однократного использования;
- - "провальные" проекты, рассматривающие паттерны в качестве "серебряной пули"[3];
- - системы, спроектированные без использования ООП;
- - системы, написанные на основе устаревших методик;

2. Основная часть

2.1. Классификация паттернов

При реализации проектов по разработке программных систем и моделированию бизнес-процессов встречаются ситуации, когда решение проблем в различных проектах имеют сходные структурные черты. Попытки выявить похожие схемы или структуры в рамках объектно-ориентированного анализа и проектирования привели к появлению понятия паттерна, которое из абстрактной категории превратилось в неперенный атрибут современных CASE-средств

Паттерны различаются степенью детализации и уровнем абстракции. Предлагается следующая общая классификация паттернов по категориям их применения:

- Архитектурные паттерны
- Паттерны проектирования
- Паттерны анализа
- Паттерны тестирования
- Паттерны реализации

Архитектурные паттерны (Architectural patterns) - множество предварительно определенных подсистем со спецификацией их ответственности, правил и базовых принципов установления отношений между ними.

Архитектурные паттерны предназначены для спецификации фундаментальных схем структуризации программных систем. Наиболее известными паттернами этой категории являются паттерны GRASP (General Responsibility Assignment Software Pattern). Эти паттерны относятся к уровню системы и подсистем, но не к уровню классов. Как правило, формулируются в обобщенной форме, используют обычную терминологию и не зависят от области приложения. Паттерны этой категории систематизировал и описал К. Ларман.

Паттерны проектирования (Design patterns) - специальные схемы для уточнения структуры подсистем или компонентов программной системы и отношений между ними.

Паттерны проектирования описывают общую структуру взаимодействия элементов программной системы, которые реализуют исходную проблему проектирования в конкретном контексте. Наиболее известными паттернами этой категории являются паттерны GoF (Gang of Four), названные в честь Э. Гаммы, Р. Хелма, Р. Джонсона и Дж. Влссидеса, которые систематизировали их и представили общее описание. Паттерны GoF включают в себя 23 паттерна. Эти паттерны не зависят от языка реализации, но их реализация зависит от области приложения.

Паттерны анализа (Analysis patterns) - специальные схемы для представления общей организации процесса моделирования.

Паттерны анализа относятся к одной или нескольким предметным областям и описываются в терминах предметной области. Наиболее известными паттернами этой группы являются паттерны бизнес-моделирования ARIS (Architecture of Integrated Information Systems), которые характеризуют абстрактный уровень представления бизнес-процессов. В дальнейшем паттерны анализа конкретизируются в типовых моделях с целью выполнения аналитических оценок или имитационного моделирования бизнес-процессов.

Паттерны тестирования (Test patterns) - специальные схемы для представления общей организации процесса тестирования программных систем.

К этой категории паттернов относятся такие паттерны, как тестирование черного ящика, белого ящика, отдельных классов, системы. Паттерны этой категории систематизировал и описал М. Гранд. Некоторые из них реализованы в инструментальных средствах, наиболее известными из которых является IBM Test Studio. В связи с этим паттерны тестирования иногда называют стратегиями или схемами тестирования.

Паттерны реализации (Implementation patterns) - совокупность компонентов и других элементов реализации, используемых в структуре модели при написании программного кода.

Эта категория паттернов делится на следующие подкатегории: паттерны организации программного кода, паттерны оптимизации программного кода, паттерны устойчивости кода, паттерны разработки графического интерфейса пользователя и др. Паттерны этой категории описаны в работах М. Гранда, К. Бека, Дж. Тидвелла и др. Некоторые из них реализованы в популярных интегрированных средах программирования в форме шаблонов создаваемых проектов. В этом случае выбор шаблона программного приложения позволяет получить некоторую заготовку программного кода.

Паттерны проектирования позволяют решать различные задачи, с которыми постоянно сталкиваются проектировщики объектно-ориентированных приложений. Ниже представлен полный список *паттернов проектирования* и краткое описание назначения каждого из них

| № | Название паттерна | Перевод | Назначение паттерна |
|---|----------------------------|---------------------|--|
| 1 | Abstract Factory | Абстрактная фабрика | Предоставляет интерфейс для создания множества связанных между собой или независимых объектов, конкретные классы которых неизвестны. |
| 2 | Adapter(синоним - Wrapper) | Адаптер (Обертка) | Преобразует существующий интерфейс класса в другой интерфейс, который понятен клиентам. При этом обеспечивает совместную работу классов, невозможную без данного паттерна из-за несовместимости интерфейсов. |
| 3 | Bridge | Мост | Отделяет абстракцию класса от его реализации, |

| | | | |
|----|-------------------------|----------------------|---|
| | | | благодаря чему появляется возможность независимо изменять то и другое. |
| 4 | Builder | Строитель | Отделяет создание сложного объекта от его представления, позволяя использовать один и тот же процесс разработки для создания различных представлений. |
| 5 | Chain of Responsibility | Цепочка обязанностей | Позволяет избежать жесткой зависимости отправителя запроса от его получателя, при этом объекты-получатели связываются в цепочку, а запрос передается по цепочке, пока какой-то объект его не обработает. |
| 6 | Command | Команда | Инкапсулирует запрос в виде объекта, обеспечивая параметризацию клиентов типом запроса, установление очередности запросов, протоколирование запросов и отмену выполнения операций. |
| 7 | Composite | Компоновщик | Группирует объекты в иерархические структуры для представления отношений типа "часть-целое", что позволяет клиентам работать с единичными объектами так же, как с группами объектов. |
| 8 | Decorator | Декоратор | Применяется для расширения имеющейся функциональности и является альтернативой порождению подклассов на основе динамического назначения объектам новых операций. |
| 9 | Facade | Фасад | Предоставляет единый интерфейс к множеству операций или интерфейсов в системе на основе унифицированного интерфейса для облегчения работы с системой. |
| 10 | Factory Method | Фабричный метод | Определяет интерфейс для разработки объектов, при этом объекты данного класса могут быть созданы его подклассами. |
| 11 | Flyweight | Приспособленец | Использует принцип разделения для эффективной поддержки большого числа мелких объектов. |
| 12 | Interpreter | Интерпретатор | Для заданного языка определяет представление его грамматики на основе интерпретатора предложений языка, использующего это представление. |
| 13 | Iterator | Итератор | Дает возможность последовательно перебрать все элементы составного объекта, не раскрывая его внутреннего представления. |
| 14 | Mediator | Посредник | Определяет объект, в котором инкапсулировано знание о том, как взаимодействуют объекты из некоторого множества. Способствует уменьшению числа связей между объектами, позволяя им работать без явных ссылок друг на друга и независимо изменять схему взаимодействия. |
| 15 | Memento | Хранитель | Дает возможность получить и сохранить во внешней памяти внутреннее состояние объекта, чтобы позже объект можно было восстановить |

| | | | |
|----|-----------------|-----------------|--|
| | | | точно в таком же состоянии, не нарушая принципа инкапсуляции. |
| 16 | Observer | Наблюдатель | Специфицирует зависимость типа "один ко многим" между различными объектами, так что при изменении состояния одного объекта все зависящие от него получают извещение и автоматически обновляются. |
| 17 | Prototype | Прототип | Описывает виды создаваемых объектов с помощью прототипа, что позволяет создавать новые объекты путем копирования этого прототипа. |
| 18 | Proxy | Заместитель | Подменяет выбранный объект другим объектом для управления контроля доступа к исходному объекту. |
| 19 | Singleton | Одиночка | Для выбранного класса обеспечивает выполнение требования единственности экземпляра и предоставления к нему полного доступа. |
| 20 | State | Состояние | Позволяет выбранному объекту варьировать свое поведение при изменении внутреннего состояния. При этом создается впечатление, что изменился класс объекта. |
| 21 | Strategy | Стратегия | Определяет множество алгоритмов, инкапсулируя их все и позволяя подставлять один вместо другого. При этом можно изменять алгоритм независимо от клиента, который им пользуется. |
| 22 | Template Method | Шаблонный метод | Определяет структуру алгоритма, перераспределяя ответственность за некоторые его шаги на подклассы. При этом подклассы могут переопределять шаги алгоритма, не меняя его общей структуры. |
| 23 | Visitor | Посетитель | Позволяет определить новую операцию, не меняя описаний классов, у объектов которых она вызывается. |

2.2. Rational Unified Process

Rational Unified Process (RUP) – фреймворк для разработки коммерческого программного обеспечения. RUP состоит из:

- Инструкции. RUP содержит набор отличных инструкций, покрывающий все: от управления проектом до детального руководства тестированием
- Инструменты для доставки процесса. RUP использует WEB-технологии для доставки, что позволяет быть интегрированным с другими инструментам разработки программного обеспечения и быть легкодоступным разработчикам
- Средства настройки. RUP состоит из компонентов и плагинов, которые могут быть выбраны и сконфигурированы в соответствии с конкретными требованиями проекта
- Разработка процесса. Организации могут расширять или модифицировать RUP путем создания своих собственных плагинов, используя Rational Process Workbench
- Сообщество. Rational Developer Network позволяет разработчиками программного обеспечения общаться и обмениваться своими расширениями для RUP.



(RUP) предоставляет очень упорядоченный и структурированный подход к разработке программного обеспечения. Его цель – гарантировать разработку программного обеспечения высокого качества, отвечающее требованиям конечного пользователя в пределах установленного расписания и бюджета.

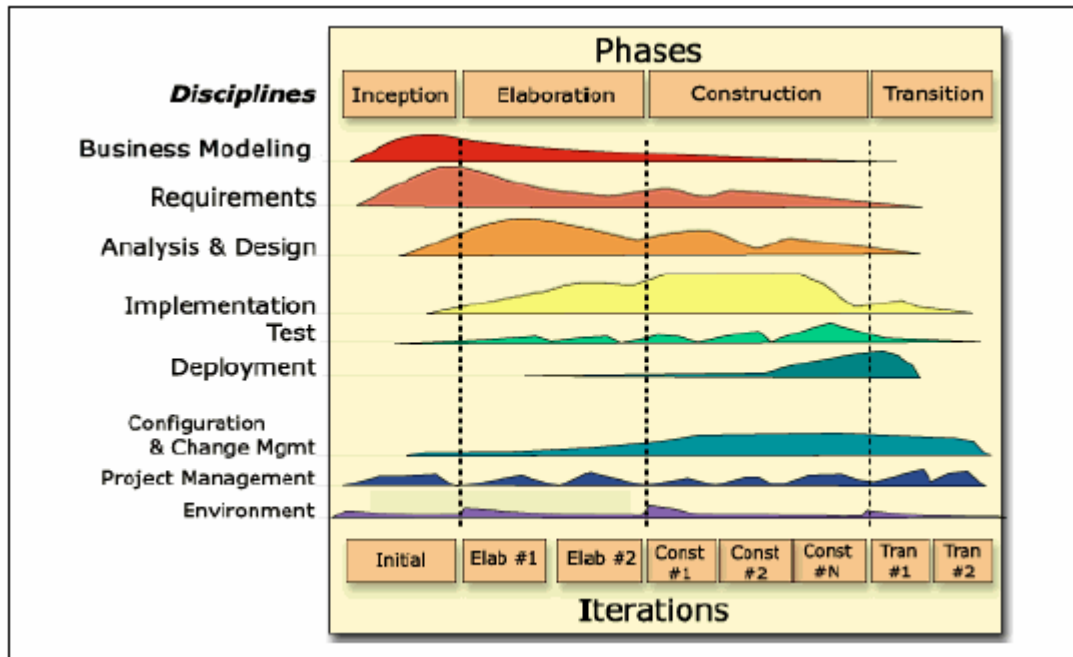
Средство достижения этой цели – использование так называемых коммерчески доказанных «лучших правил». Эти правила включают в себя:

- Пошаговая разработка
Функциональность системы должна быть выполнена в последовательной серии релизов, направленных на критические задачи, получая отклики, начиная с ранних релизов.
- Управление требованиями
Это систематический подход к организации, установке связей и управлению изменяющимися требованиями к программной системе или приложению.
- Использование компонентной архитектуры
RUP предоставляет методический, систематический подход для дизайна, разработки и верификации архитектуры. Архитектура, основанная на компонентах более гибка и понятна, нежели любая другая.
- Визуальное моделирование ПО
Модели – упрощения реальности. Они помогают нам понять проблемы, возможные пути их решения, а также осмыслить большие комплексные системы, которые в ином случае невозможно понять, как единое целое. RUP использует Unified Model Language (UML), стандартный графический язык для визуализации, спецификации, конструирования и документирования программных систем.
- Постоянная проверка качества
Качество – это обязанность каждого члена команды разработчиков. Управление качеством должно присутствовать во время всего жизненного цикла путем осуществления и оценки качества процесса и продукта.
- Контролирование изменения ПО

RUP использует так называемую мета-модель для описания процессов разработки. Она включает в себя следующее:

- Артефакты – то, что произведено;
- Действий – как сделать работу;
- Роли – кто выполняет работу.

Артефакты могут принимать различные формы: модели (например, Use-case model, Design model), базы данных или другие репозитории информации, исходный код и исполняемые файлы, документация.



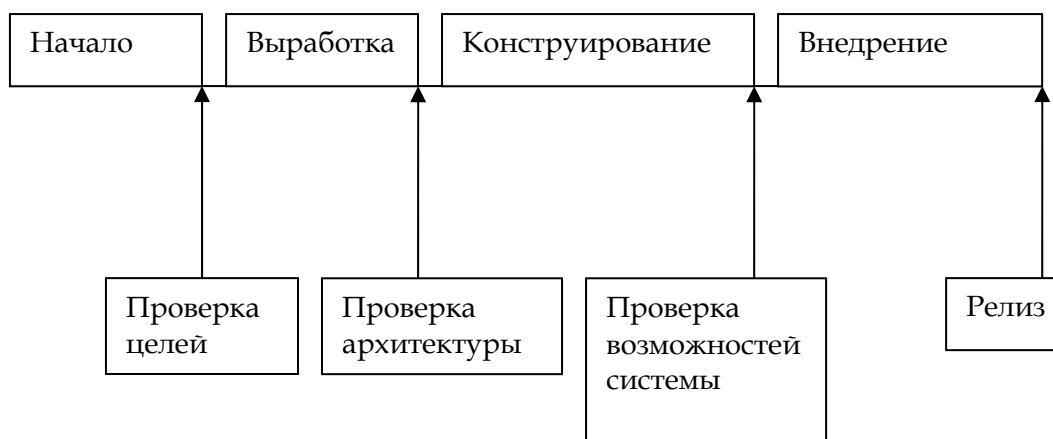
Роли определяют обязанности членов группы, работающей вместе над процессом разработки ПО в рамках одного проекта.

Действия – это работа, выполненная членами команды, которую можно определить как набор операций по созданию или изменению артефакта.

Еще одно ключевое понятие в RUP – фазы. Фазы обеспечивают «проверочные точки», «milestones», которые позволяют удостовериться, что итеративный процесс приводит к прогрессу и сходится к решению задачи, а не к неопределенности.

Фазы RUP:

- Начало (Inception)
- Выработка (Elaboration) – устанавливается архитектура системы
- Конструирование
- Внедрение



2.3. Использование IBM паттернов

2.3.1. Паттерны в отправной точке (Inception)

Ключевые цели отправной точки – выработка основных требований, определения инвестирования, бюджета и графика работы.

В следующей таблице попытаемся показать, как IBM паттерны могут быть использованы в начальной стадии RUP.

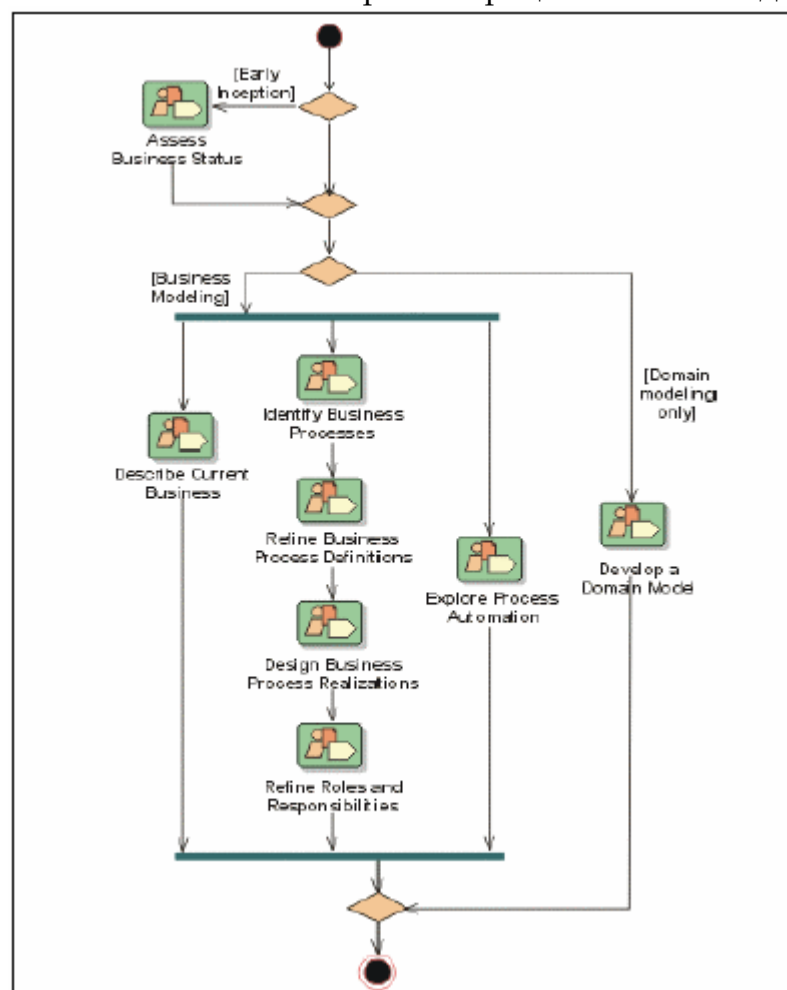
| Элементы RUP | Использование паттерна |
|---|--|
| <p>Окружение</p> <p>Задача: подготовить окружение проекта</p> <p>Действие: подготовить директивы к проекту</p> <p>Артефакт: Спецификация</p> | <p>Паттерны – один из основных интеллектуальных активов, который должен быть принят во внимание во время разработки начальных директив, указаний и спецификации.</p> |
| <p>Бизнес-моделирование</p> <p>Задача: изучить автоматизацию процесса</p> <p>Артефакт: Бизнес-образ</p> | <p>E-business паттерны могут быть использованы как отправные точки для определения и описания бизнес-процессов. Также анализ целей и процессов бизнеса даст ценную информацию о том что и как автоматизировать.</p> |
| <p>Требования</p> <p>Задача: Описать система</p> <p>Действие: Разработка образа</p> <p>Артефакт: Образ</p> | <p>Необходимость сбора информации для навигации по каталогу активов дает возможность сфокусированного анализа требований.</p> <p>Навигация по каталогу паттернов начинается с четкого осознания бизнес-требований.</p> |
| <p>Анализ и дизайн</p> <p>Задача: Выполнить архитектурный синтез</p> <p>Действие: архитектурный анализ</p> <p>Артефакт: Документ, описывающий архитектуру</p> | <p>Обзор архитектур, примерами которых являются Паттерны для e-business. Содействует выбору архитектуры.</p> |
| <p>Управление проектом</p> <p>Задача: Оценить границы и риски проекта</p> <p>Задача: Планирование</p> | <p>Описанная и выбранная ранее с помощью E-business паттернов архитектура помогает создать план разработки.</p> |

Артефакт: План разработки ПО

Рассмотрим теперь все пункты на примере выдуманной компании First Financial – крупной финансовой компании, ищущей способы эффективного предоставления услуг их клиентам.

Бизнес моделирование – опциональная часть RUP, которая используется для лучшего понимания бизнес процессов и определения пути их усовершенствования. Один из способов улучшения бизнес-процесса – определение возможностей автоматизации.

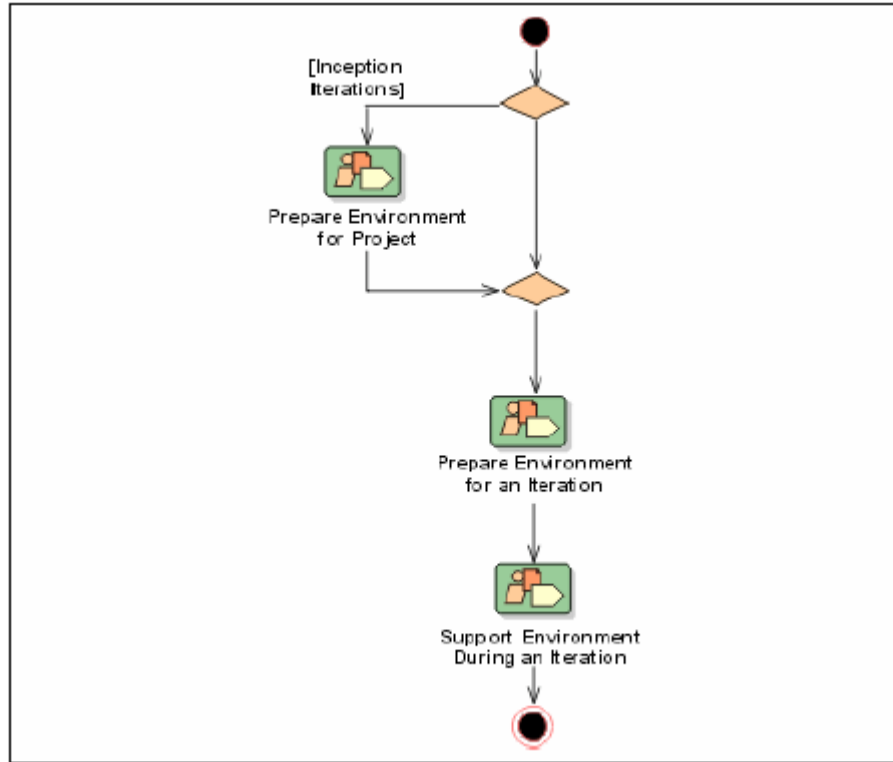
На рисунке ниже схематически изображен процесс Бизнес-моделирования:



В случае с First Financial на этом этапе мы можем описать, как сервисы предоставляются сейчас и как они могут быть улучшены. Один из способов – просмотр всех допустимых паттернов из репозитория. Например, паттерн для Self-Service вполне подходит для обеспечения работы клиентов с кредитными картами.

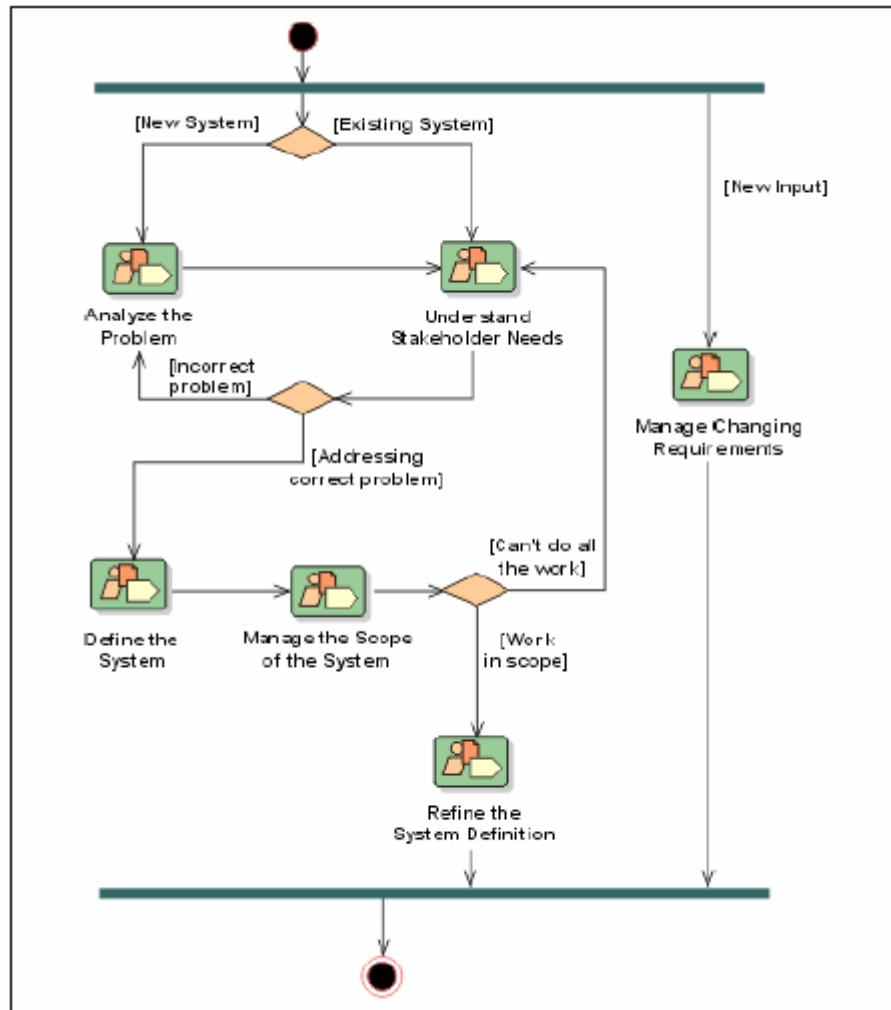
E-business паттерны – основные активы, которые должны быть рассмотрены во время начальной разработки процесса для e-business проекта.

На следующем этапе паттерны оцениваются в зависимости от их релевантности к проекту и процессу.



Так как First Financial заинтересован в e-business, то мы решаем использовать паттерны для быстрого определения пути к необходимому решению. Паттерны обычно не являются идеальным решением и должны быть изменены под нужды определенного проекта.

На следующем этапе мы анализируем проблему, вырабатываем требования к продукту и определяем систему:

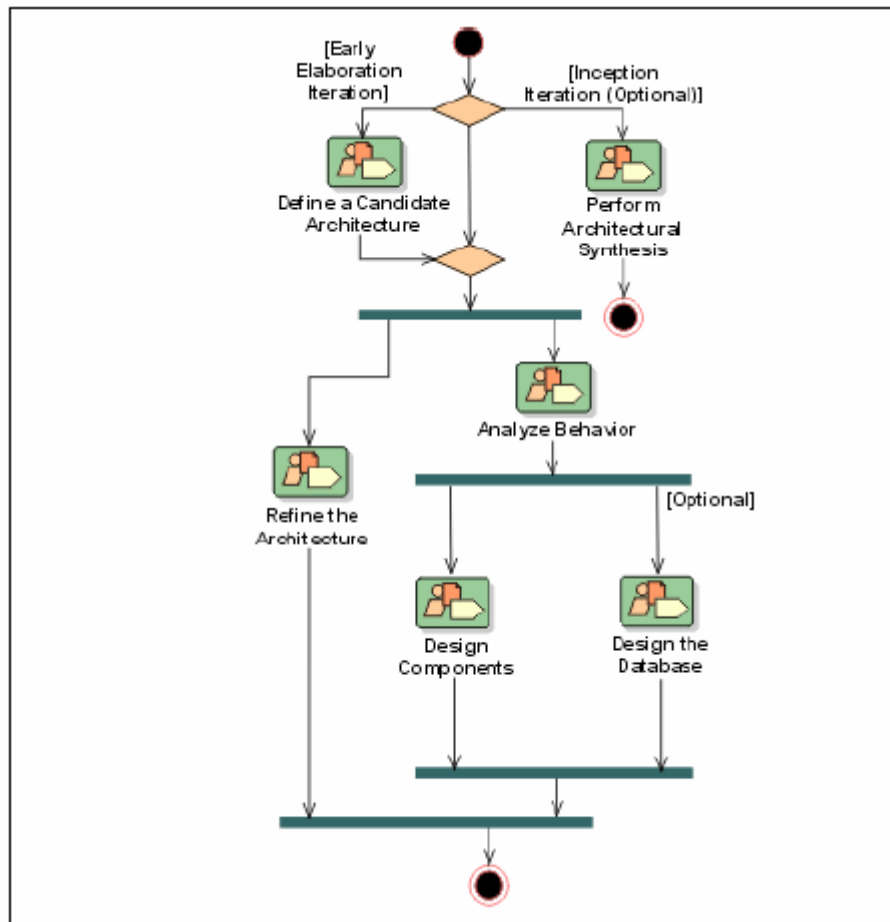


Цель First Financial – открыть доступ к существующим транзакциям и данным для «внешних клиентов». В частности, первым возможным решением тут может быть создание Веб-интерфейса для упавления кредитными картами.

Из раздела бизнес-моделирования мы можем выработать следующие цели First Financial:

- Уменьшение времени доступа к рынку
- Тактическое фокусирование на поддержке Веб-сервисов
- Стратегическое фокусирование на поддержке нескольких каналов доступа (кол-центры, телефонный доступ)

Анализ и дизайн обычно опционален и используется обычно для того, чтобы убедить инвесторов в необходимости финансирования. Паттерны из репозитория могут значительно помочь при составлении описания архитектуры. Выбор необходимых паттернов производится при помощи требований, выявленных раньше.

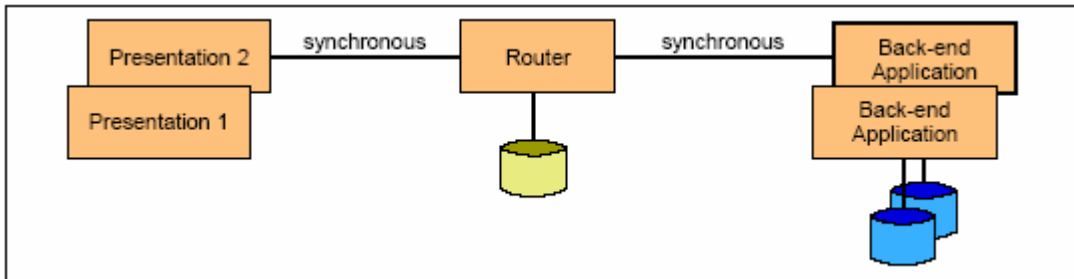


Во многих случаях инвесторам бывает достаточно лишь показать наличие нескольких паттернов, которые способны решить поставленные проблемы. Также могут быть представлены списки возможных к использованию технологий, набросок концептуальной модели, симуляция решения, прототип.

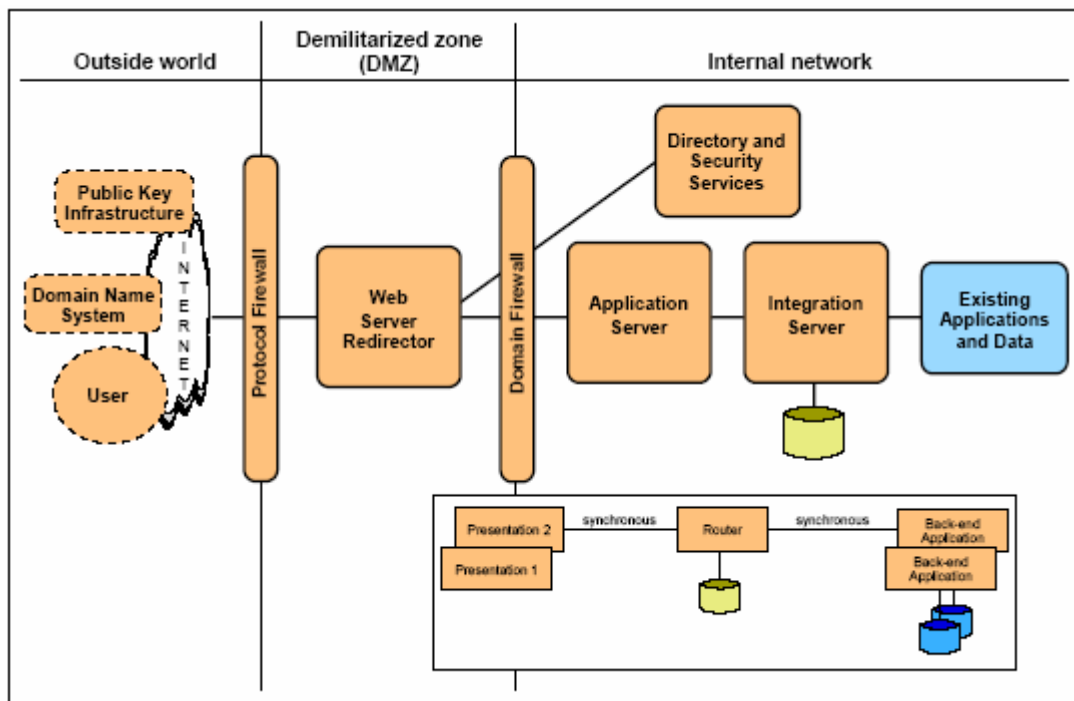
Очень часто случается, что для решения проблемы подходят несколько паттернов. В этом случае необходимо будет выполнить дополнительную работу по интеграции нескольких паттернов.

| Business Drivers | Stand-Alone | Directly-integrated | As-is Host | Customized Presentation to Host | Router | Decomposition | Agent | IT Drivers | Stand-Alone | Directly-integrated | As-is Host | Customized Presentation to Host | Router | Decomposition | Agent |
|---|----------------|---------------------|------------|---------------------------------|--------|---------------|-------|--|----------------|---------------------|------------|---------------------------------|--------|---------------|-------|
| | Single Channel | Single Channel | | | | | | | Single Channel | | | | | | |
| Time to market | ✓ | | ✓ | ✓ | | | | Minimize application complexity | ✓ | | | ✓ | | | |
| Improve the organizational efficiency | | ✓ | ✓ | ✓ | | ✓ | ✓ | Minimize total cost of ownership (TCO) | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Reduce the latency of business events | | ✓ | | | ✓ | ✓ | ✓ | Leverage existing skills | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Easy to adapt during mergers and acquisitions | | | | | ✓ | ✓ | ✓ | Leverage legacy investment | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Integration across multiple delivery channels | | | | | ✓ | ✓ | ✓ | Backend application integration | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| United customer view across lines of business (LOB) | | | | | | ✓ | ✓ | Minimize enterprise complexity | | | | | ✓ | ✓ | ✓ |
| Support effective cross-selling | | | | | | | ✓ | Maintainability | | | | | ✓ | ✓ | ✓ |
| Mass customization | | | | | | | ✓ | Scalability | | | | | ✓ | ✓ | ✓ |

Из набора Self-Service паттернов, представленного выше, для First Financial подходит Router pattern, так как он позволяет в будущем использовать несколько каналов для доступа к одному приложению (что входит в планы стратегического развития).

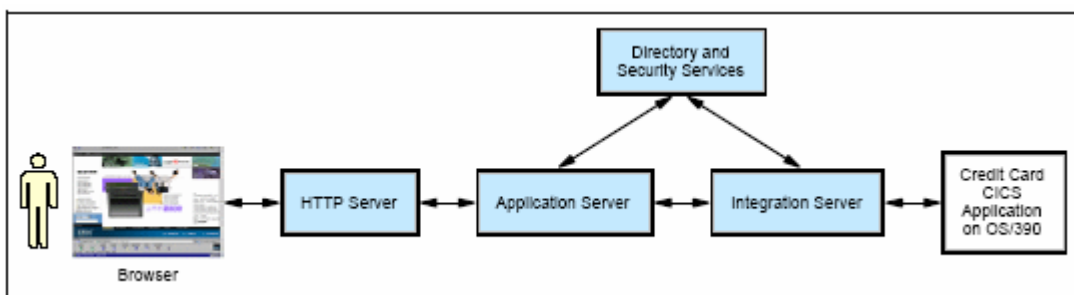


Выбор паттерна приводит нас к так называемому Runtime паттерну, которую описывает технические функциональные возможности выбранного паттерна.



Далее архитектор должен провести «продакт мэшинг» - подбор возможных продуктов, используемых для реализации этого паттерна.

В результате анализа мы можем представить следующую обзорную диаграмму:



На этом же этапе необходимо провести документирование, получив в качестве артефакта документ, описывающий архитектуру.

Анализ и дизайн были сделаны, а значит осуществимость проекта была доказана. Теперь можно переходить к управлению проектом, который включает в себя оценку бюджета, графика и необходимых для выполнения проекта ресурсов.

Надо не забывать, что паттерны – не есть законченное архитектурное решение, они только помогают архитектору более быстро составить, подобрать и модифицировать архитектуру системы. Использование паттернов также обычно увеличивает качество системы в целом.

2.3.2. Паттерны в выработке (Elaboration)

Если на начальном этапе анализ и архитектура системы были опциональны и могли включать в себя только часть документов и описаний системы, то на этом этапе окончательная выработка архитектуры является основной задачей.

| Элементы RUP | Использование паттерна |
|---|--|
| Требования Задача: полностью определить систему Артефакт: образ, use cases, дополнительные спецификации | Образ системы и требования к ней должны быть окончательными к завершению этапа Elaboration. Должно быть предоставлено достаточно деталей для окончательного определения системы и смягчения всех возможных рисков. Паттерны выбранные ранее рассматриваются еще раз, так как требования были уточнены. В зависимости от изменений внесенных в требования, паттерны подтверждаются или изменяются. |
| Анализ и дизайн Задача: определить конечную архитектуру Артефакт: Документ, описывающий архитектуру | Так как требования были изменены, проводится переоценка выбранных паттернов, чтобы удостовериться, что они все еще решают поставленные задачи. |

В нашем примере Self-Service остается единственным необходимым для использования паттерном. Следующие нефункциональные требования добавляются к спецификации: многоязыковая поддержка, способность работать с не менее чем 500 клиентами одновременно, масштабируемость, безопасность.

Работа по анализу и дизайну архитектуры, сделанная на первом этапе используется как отправная точка при создании конечной архитектуры. Смысл выполняемых действий остается тот же, что и раньше за той лишь разницей, что

теперь выбор архитектуры более не опционален и должен делаться с пониманием того, что является конечным, без возможности изменить его в будущем.

Дополнительно осуществляются следующие действия для тщательного анализа:

- Определение механизмов дизайна
- Определение элементов дизайна
- Описание функциональности системы и ее распределения, того, как система будет работать.
- Внедрение уже существующих элементов
- Структурирование модели реализации

В нашем примере все решения принятые на начальном этапе остаются верными и тут. Кроме этого вырабатывается дополнительный анализ о стоимости разработки и производительности системы.

2.3.3. Паттерны в конструировании (Construction)

На этом этапе мы реализуем ту архитектуру, которая была представлена в результате Elaboration этапа. Возможность изменения паттернов все еще есть, так как изменения к требованиям и целям еще возможны, хотя и по возможности должны быть сведены к минимуму.

Если ранее мы фокусировались только на ключевых требованиях к системе, то теперь необходимо сконцентрироваться на деталях.

3. Заключение

В этой работе было показано, каким образом паттерны могут быть эффективно использованы во время разработки RUP проекта, увеличив не только время исполнения, но и качество системы в целом.

И в заключение хочется дать несколько советов от проповедника идей “экстремального программирования” Мартина Фаулера, который посвятил немало своих статей проблемам применения шаблонов проектирования. Эти советы могут пригодиться тем программистам, которые еще только начинают осваивать тонкости методологических аспектов проектирования программного обеспечения, так и уже опытным разработчикам.

- Не бойтесь потратить время на изучение паттернов;
- Хорошо подумайте, когда лучше всего применить паттерн;
- Хорошо подумайте, как лучше всего реализовать паттерн в его simplestейшей форме, а уже потом вносите дополнения;
- Если вы применили паттерн, а потом поняли, что без него было бы лучше - убирайте, не сомневайтесь

4. Литература

1. Galls M., Macslaac B, Popescu D. Using a single Business Patter with RUP // IBM Corp. 2004, Redbooks Paper
2. Озеров А. Загадочный мир шаблонного проектирования
<http://lib.juga.ru/article/articleview/167/1/68>
3. Лекция: Паттерны проектирования и их представление в нотации UML:
версия для печати и PDA
http://www.intuit.ru/department/pl/umlbasics/14/umlbasics_14.html