



**Представление знаний о
предметной области на основе
Эффективных юзкейсов А. Коберна
и IBM Rational RequisitePro**

**Хлебников Сергей Алексеевич,
независимый консультант,
sakhx@mail.ru**

RUP – это подход к разработке программного обеспечения, который основан на:

- приверженности итеративной схеме разработки (iterative approach)
- большом внимании к рискам (risk-driven, risk mitigation)
- управлении проектами на основе юзкейсов (use-case driven)
- сфокусированности на архитектуре (architecture-centric).

Фундаментальные принципы успешного применения RUP:

- принимайтесь за основные риски как можно раньше и продолжайте это делать постоянно ... иначе они займутся вами
- добивайтесь того, чтобы разработанное ПО представляло то, в чем нуждается заказчик (достигается с помощью юзкейсов)
- фокусируйте внимание, прежде всего, на исполнимых артефактах
- согласовывайте изменения на ранних этапах разработки
- стремитесь к построению базовой линии архитектуры на ранних этапах разработки
- выстраивайте систему на основе компонентного подхода
- работайте вместе как одна команда
- стремитесь к тому, чтобы качество стало образом жизни, а не было добавлено задним числом

- **Заказчики** должны быть уверены – Система делает то, что они ХОТЯТ.
- **Менеджеры** хотят эффективно планировать и осуществлять оперативное управление.
- **Аналитики** - документировать то, что Система должна делать.
- **Разработчики** – понимать потребности системы для её успешной разработки.
- **Тестеры** – понимать поведение Системы для её верификации.
- **Технические писатели** - понимать поведение Системы для её описания.
- **Разработчики интерфейса пользователя** – понимать цели пользователей, и как они будут использовать Систему для достижения этих целей

Термин анализ означает разбиение чего-то на составные части.

Термин синтез означает построение чего-то из составных частей. Юзкейсы фундаментально предлагают *синтетический подход*, нацеленный на то, чтобы построить такое понимание системы, с которым согласны различные группы стейкхолдеров.

1. ***Классический подход*** И. Якобсона (Ivar Jacobson), включённый в RUP.

Великолепно описан в книге

Bittner, K., Spence, I., 2002. **Use Case Modeling**, 2nd Edition, Addison Wesley.

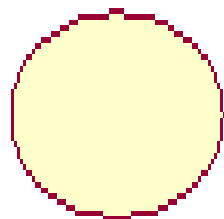
2. ***Эффективные юзкейсы*** А. Коберна (Alistair Cockburn)

Юзкейсная модель (Ивар Якобсон):

- факторы
- юзкейсы
- ассоциации юзкейсов
- экземпляры юзкейсов (сценарии)
- описания юзкейсов.

Классический подход базируется на этих концепциях и не считает нужным вводить какие-либо новые концепции.

Эффективные юзкейсы Алистера Коберна значительно расширяют этот набор.



Юзкейс

Юзкейс описывает, как эктор использует систему для достижения своей цели, и что система делает для эктора для того, чтобы достигнуть эту цель. Юзкейс текстуально описывает то, каким образом система и её экторы взаимодействуют для получения того, что представляет ценность по крайней мере для одного эктора.

Имя(Name)	Имя юзкейса. Каждый юзкейс должен иметь имя, которое указывает на то, чего удаётся достигнуть посредством взаимодействия с эктором. Имя может состоять из нескольких слов и должно начинаться с глагола в неопределённом времени.
Краткое описание (Brief description)	Краткое описание роли и цели юзкейса.
Поток событий (Flow of events)	Текстовое описание того, что Система делает по отношению к данному юзкейсу (не должно быть описанием решения специфической проблемы). Должно быть понятным стейкхолдерам. Поток событий может быть представлен в трёх формах: <ul style="list-style-type: none">▪ Базовый поток (basic flow)▪ Альтернативные потоки (alternative flows)▪ Вспомогательные потоки (subflows)
Специальные требования (Special requirements)	Текстовое описание всех остальных требований, включая нефункциональные требования, которые относятся к данному юзкейсу, однако не упоминаются ни в одном потоке событий, и которые необходимо принимать во внимание при дизайне и реализации.
Предусловия (Preconditions)	Текстовое описание ограничений в Системе при запуске юзкейса.
Послеусловия (Postconditions)	Текстовое описание ограничений в Системе при завершении юзкейса.
Точки расширения (Extension points)	Список мест внутри потоков событий юзкейса, для которых может быть определено дополнительное поведение.
Юзкейсные отношения (Relationships)	Юзкейсные отношения данного юзкейса.
Диаграммы (Diagrams)	Диаграммы, которые иллюстрируют некоторые аспекты данного юзкейса такие, как структура потока событий или юзкейсные отношения.

Юзкейс определяет соглашение между **стейкхолдерами** системы относительно ее поведения. Юзкейс описывает поведение системы при различных условиях в ответ на запрос со стороны одного из стейкхолдеров, который называется **Основным (Primary) эктором**. Основной эктор инициирует взаимодействие с системой для осуществления некоторой **цели**. Система отвечает, **защищая интересы** стейкхолдеров. При этом могут реализоваться различные **сценарии** поведения в зависимости от конкретного запроса и условий, при которых он был выполнен. Юзкейс объединяет все такие сценарии.

- Юзкейсы – это, прежде всего, **специальная текстовая нотация для записи алгоритмов**, в которой появляется возможность чётко идентифицировать каждую её отдельную ветвь (сценарий).
- Текст юзкейса в значительной степени представляет **проекцию ранее выделенных Интересов Стейкхолдеров** (например, с использованием Stakeholder Requests из RUP) **на данный юзкейс**.
- А. Коберн обобщает понятие Эктор системы. В его понимании **Эктор – это всякая сущность имеющая поведение**. Система с этой точки зрения также является Эктором. В результате, мы получаем симметричный взгляд на взаимодействие Эктора с Системой как на взаимодействие двух Экторов.
- **Любые человеко-машинные Системы** могут быть описаны, опираясь на понятия Эктора и Юзкейса в интерпретации А. Коберна.

А. Коберн вводит три фундаментальных уровня целей юзкейса:

- **user goal** (цель пользователя) – важнейший уровень; критерий **«за один присест»**
- **summary** (суммарный) – контекст *user goal*; критерий **«за несколько присестов»**
- **subfunction** (субфункциональный) – составная часть *user goal*

<score-иконка> <Имя юзкейса> <level-иконка>

Контекст использования (Context of use):

Границы рассмотрения (Scope):

Уровень цели юзкейса (Level):

Основной актер (Primary Actor):

Стейкхолдеры со своими интересами (Stakeholders & Interests):

Предусловие (Precondition):

Постусловия – минимальные гарантии (Minimal Guarantees):

Постусловия – успешные гарантии (Success Guarantees):

Триггер (Trigger):

Главный успешный сценарий (Main Success Scenario):

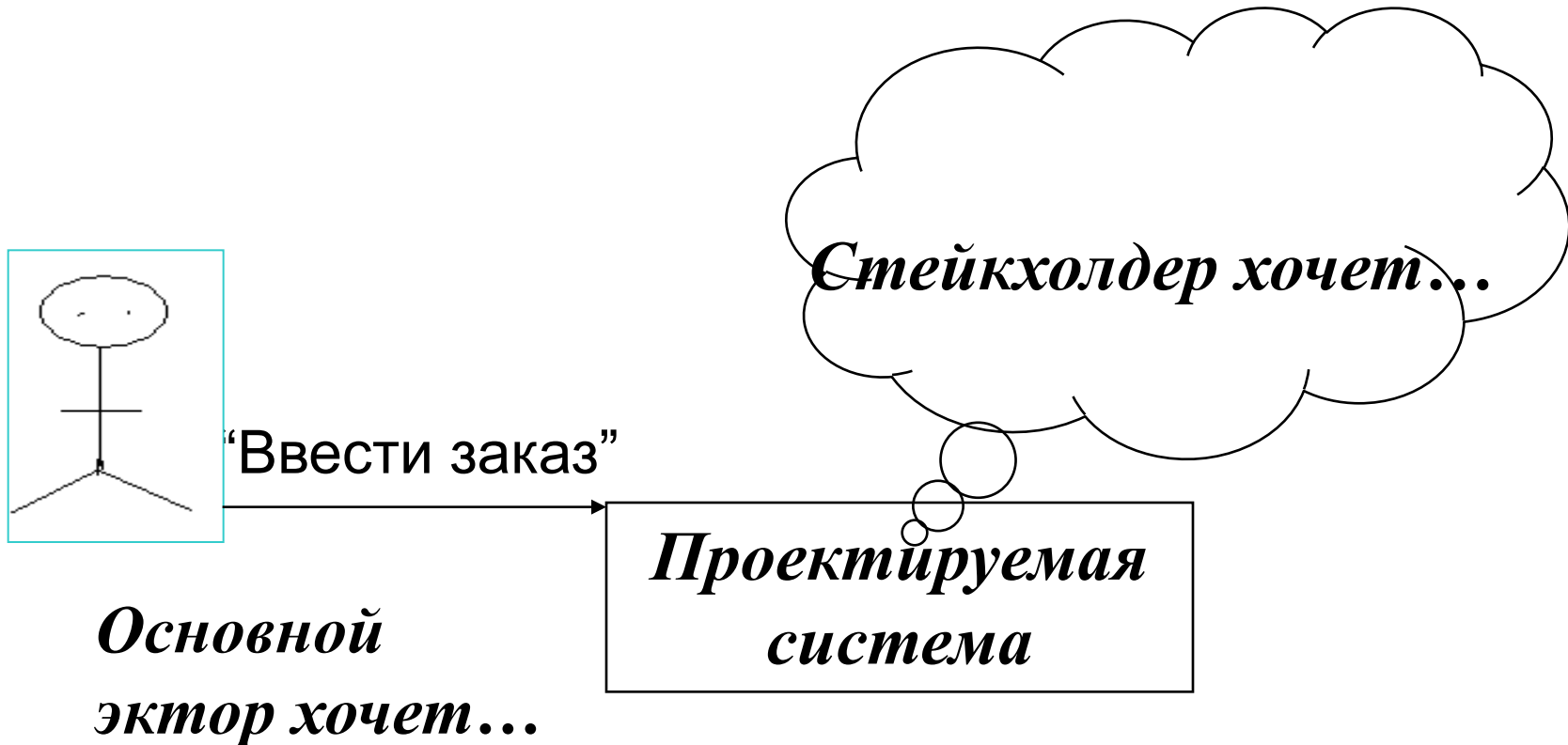
Расширения (Extensions):

Специальные требования (Special Requirements):

Список вариаций технологий и данных (Technology and Data Variations List.)

Дополнительная информация (Related Information)

Нерешенные вопросы (Open Issues)



Данный слот используется в тех случаях, когда есть желание выразить нечто, что может быть реализовано *несколькими способами*, но что не хотелось бы включать в слот «Расширения» (Extensions).

Концентрация разнородной информации в сценарной части юзкейсов может привести к тому, что её восприятие будет затруднено различными группами стейкхолдеров. В таких случаях желательно разнести эту информацию по различным слотам, связав её ссылками с шагами в сценарной части.

- Юзкейс «Обработать покупку» заимствован из книги [3]. Он описывает работу кассира, обслуживающего покупателя. Пример иллюстрирует использование Эффективных юзкейсов совместно с RequisitePro для выполнения **нарезки требований и установления трассировочных связей** с целью вычленения единиц работ, которые могут быть использованы при планировании **в условиях итеративной разработки**.
- Юзкейс «Обработать документ – стандартный алгоритм» взят из проекта «Система межфилиальных расчётов (МФР) в Сбербанке (СБ) РФ». Пример иллюстрирует использование Эффективных юзкейсов совместно с RequisitePro для выполнения **нарезки требований и установления трассировочных связей с целью восстановления документации Системы** для обеспечения возможности её дальнейшего развития. Данный пример также демонстрирует важную роль, которую играют **Отслеживаемые сущности проекта (ОСП)**.

Цитата из документа User's Guide:

RequisitePro позволяет выйти за пределы традиционного подхода к Управлению требованиями, уделяя в равной степени внимание как поддержке работе с документами, так и работе с базой данных. Благодаря глубокой интеграции MS Word с многопользовательской БД, RequisitePro обеспечивает возможность организовать требования, установить их приоритеты и отношения трассировки, а также легко отслеживать изменения в требованиях. Уникальная архитектура RequisitePro и динамические связи позволяют легко переходить от требований в БД к представлению в Word-документах и наоборот.

- Проект (Project)
- БД проекта (Project data base)
- Тип требования (Requirements Type)
- Требования (Requirements)
- Атрибуты требования (Requirement Attribute)
- Представления требований (Views)
 - Атрибутная матрица (Attribute Matrix)
 - Дерево трассировки (Traceability Tree)
- Типы документов (Document Type)
- Документы (Documents)
- Отношения трассировки (Traceability Relationship)

Концепция «Расслоение знаний» позволяет большую часть информации, связанной с предметной областью, организовать вокруг юзкейсов. Однако ссылочный механизм, используемый в формате «Fully-dressed» становится крайне неудобным при большом количестве различных ссылок, поскольку он имеет чисто **текстовую природу**.

RequisitePro совместно с концепцией «Расслоение знаний» позволяет получить качественно новый результат благодаря возможностям **«нарезки требований» и средствам трассировки**.

RequisitePro позволяет выполнить «нарезку требований». В качестве требования можно объявить любой фрагмент любого документа, либо построить требование непосредственно в БД. «Нарезанные» требования можно связать отношениями трассировки. Схема «нарезки» и трассировочные отношения зависят от поставленной цели и структуры документа. Независимо от этого важнейшая цель - построение сетевой структуры, которая позволяет наилучшим образом упорядочить информацию, находящуюся в различных слотах в формате «Fully-dressed».

На двух нижеприведённых слайдах показаны схемы в виде диаграммы классов UML, которые я использую для организации информации, расположенной в слотах Эффективного юзкейса в формате «Fully-dressed».

Первая схема демонстрирует возможности средств трассировки RequisitePro для **организации итеративной разработки**.

Вторая схема показывает, каким образом с помощью средств трассировки можно **представить знания**, относящиеся к любой предметной области. Например, построить описание ранее разработанной достаточно сложной Системы, для которой отсутствует необходимая для её развития документация.

Диаграмма классов UML для схемы трассировки в RequisitePro в расширенном формате "Fully-dressed" для итеративной разработки

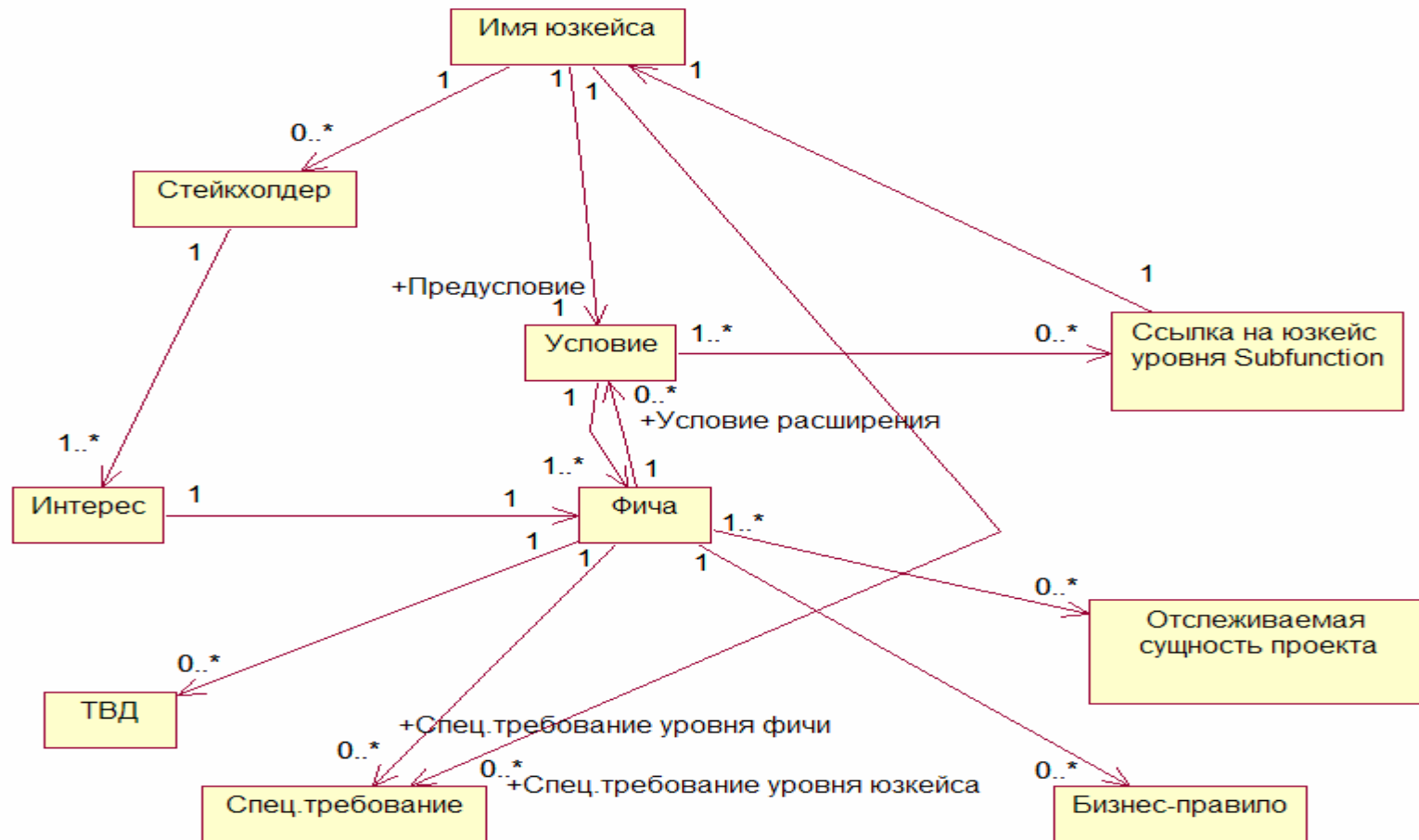
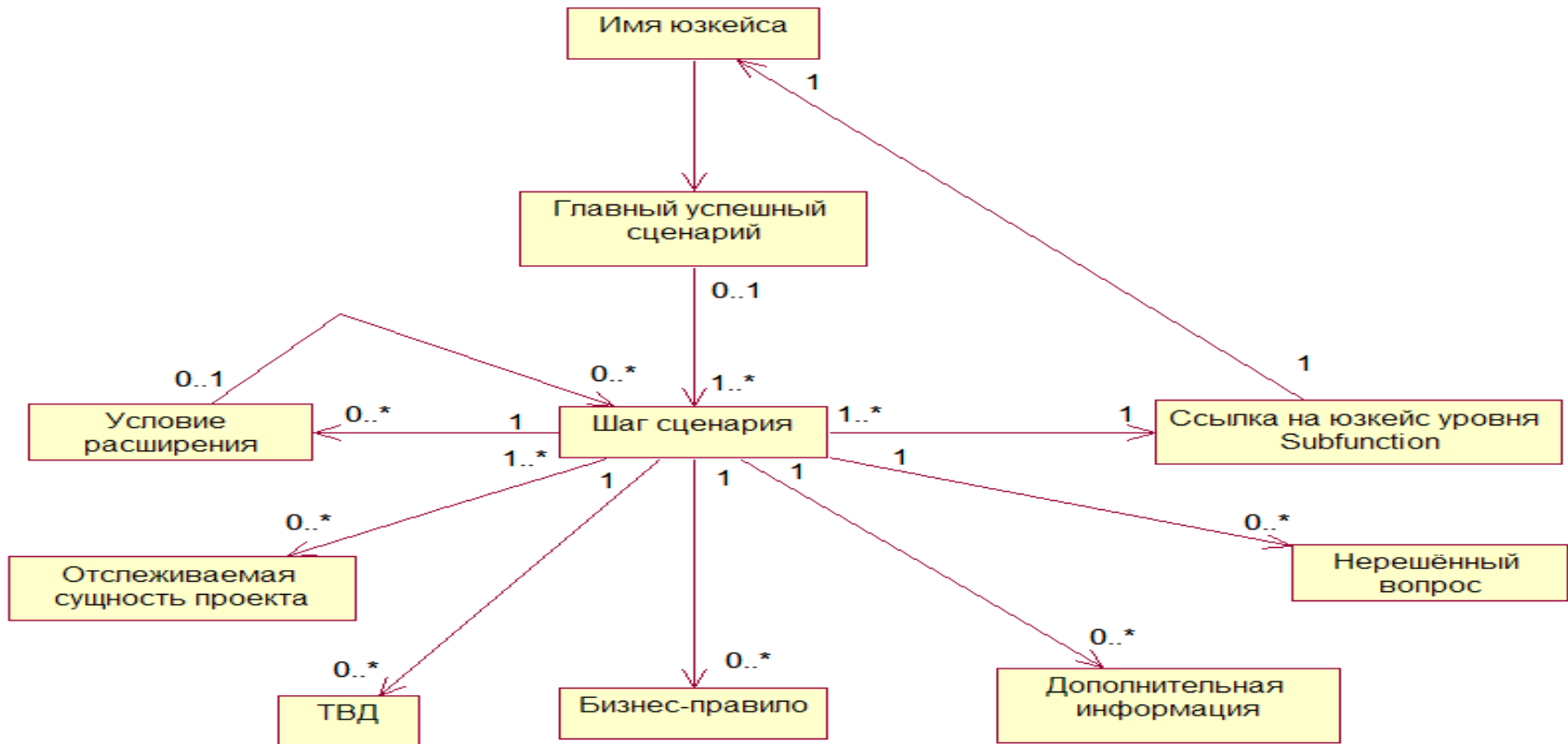


Диаграмма классов UML для схемы трассировки в RequisitePro в расширенном формате "Fully-dressed" для представления знаний о ПО.



RequisitePro позволяет организовать планирование и выполнение отдельной итерации.

- Планирование предполагает выделение работ небольшого объёма;
- Выполнение предполагает непродолжительное время для реализации одной работы (часы, реже дни).

Нарезка требований для Главного успешного сценария юзкейса

Главный успешный сценарий (Main Success Scenario):

1. [EUC10 Кассир начинает обслуживание покупателя.](#)
2. [EUC11 Кассир вводит идентификатор единицы покупки.](#)
3. [Система записывает информацию о единице покупки, отображает ее описание и отображает текущий итог.](#)

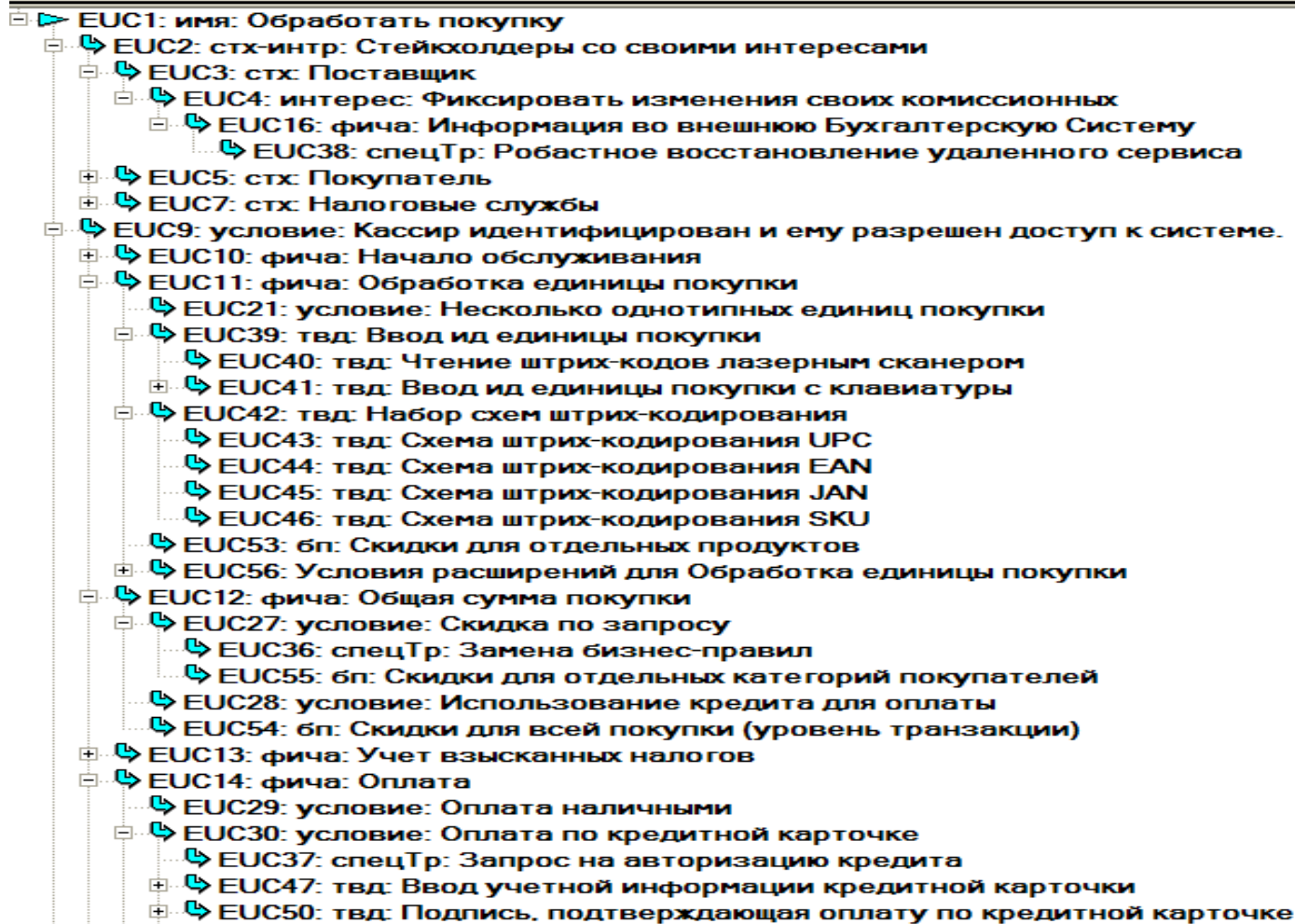
Кассир повторяет шаги 2-3 до завершения обслуживания покупателя.

4. [EUC12 Система представляет общую сумму покупки.](#) с [EUC13 учетом взысканных налогов.](#)
5. Кассир называет Покупателю сумму к оплате.
6. [EUC14 Покупатель платит. Система обрабатывает платеж.](#)
7. [EUC15 Система журналирует покупку](#) и посылает информацию о покупке и платеже [EUC16 во внешнюю Бухгалтерскую Систему](#) (для учета и вычисления комиссионных) и [EUC17 в Инвентарную Систему](#) (для изменения инвентарной ведомости).
8. [EUC18 Система предоставляет контрольный чек покупки.](#)
9. Покупатель покидает магазин с чеком и покупкой.

Фрагмент нарезки требований для различных слотов с показом трассировочных связей



Фрагмент дерева трассировки в RequisitePro



Фрагмент атрибутной матрицы в RequisitePro

Requirements:	Iteration	Property	Priority	Status
EUC38: спецТр: Робастное восстановление...	4	special	Medium	Approve
EUC40: твд: Чтение штрих-кодов лазерным...	3	tdv	Medium	Propose
▶ EUC41: твд: Ввод ид единицы покупки с...	1	tdv	Medium	Propose
EUC43: твд: Схема штрих-кодирования UPC	3	tdv	Low	Approve
EUC44: твд: Схема штрих-кодирования EAN	4	tdv	Low	Approve
EUC45: твд: Схема штрих-кодирования JAN	4	tdv	Medium	Approve
EUC46: твд: Схема штрих-кодирования SKU	4	tdv	Medium	Approve
EUC10: фича: Начало обслуживания	1	feat	Medium	Approve
EUC11: фича: Обработка единицы покупки	1	feat	Medium	Approve
EUC12: фича: Общая сумма покупки	1	feat	Medium	Approve
EUC13: фича: Учет взысканных налогов	3	feat	High	Propose
EUC14: фича: Оплата		feat	Medium	Approve
EUC15: фича: Журналирование покупки	2	feat	Medium	Approve
EUC16: фича: Информация во внешнюю...	3	feat	Medium	Approve
EUC17: фича: Информация в Инвентарную Систему	3	feat	Medium	Approve
EUC18: фича: Контрольный чек покупки	2	feat	Medium	Approve
EUC21: условие: Несколько однотипных единиц...		cond	Medium	Approve
EUC29: условие: Оплата наличными	1	cond	Medium	Approve

ОСП представляет некоторую информацию проекта, которая имеет форму таблицы. Она не требует отдельного документа, в RequisitePro представляется отдельным типом требования и сохраняется исключительно в его Базе данных.

В Системе Межфилиальных расчётов (МФР) использовались следующие ОСП:

- проверки правильности заполнения полей;
- типы банковских документов;
- действия (внутренняя кодировка использованная разработчиками;
- различная справочная информация из банковской сферы.

Такая форма представления, по моему мнению, является гораздо более удобной, чем традиционно используемая для этой цели Диаграмма классов UML. Это связано, в первую очередь, с единой формой представления знаний, которую предоставляет RequisitePro, а самое главное возможностью использовать его

Проблемы

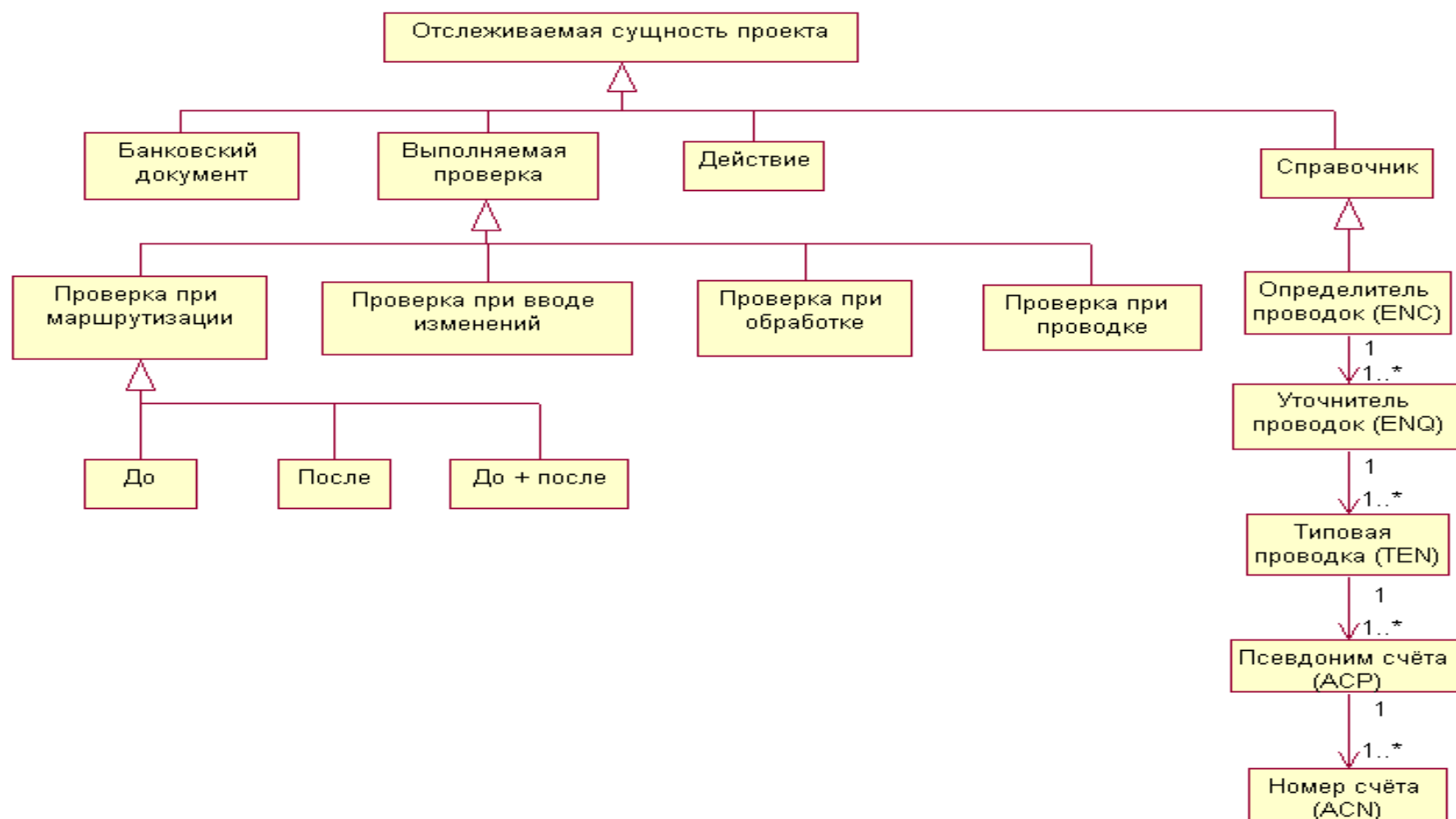
Проект предполагал восстановление документации ранее разработанной Системы с целью её дальнейшего развития. Сама предметная область отличалась **большой сложностью** во многом из-за большого количества **нетривиальным образом организованной различной справочной информации**, для которой не было удобного описания. Ситуация усугублялась ещё и **непростыми отношениями между различными подразделениями СБ**.

Система первоначально строилась путём разбиения её на отдельные компоненты, за каждую из которых отвечал один разработчик. Каждый из разработчиков использовал собственные средства для документирования разрабатываемой компоненты. В результате, цельной картины того, что должна делать Система не было ни у одного из членов команды разработчиков. Это порождало совершенно недопустимую зависимость от каждого из разработчиков.

Решение

С самого начала было принято решение строить документирование на основе **Эффективных юзкейсов с использованием RequisitePro**. В качестве формата юзкейса использовался расширенный формат “Fully-dressed”. Этот формат позволяет **найти слот для любой информации, полученной в процессе интервью**, что даёт возможность не потонуть в большом объёме информации. В процессе написания юзкейсов я старался следовать всем рекомендациям А. Коберна для **получения юзкейсов высокого качества** (в том числе, я стремился к тому, чтобы каждый юзкейс имел разумное число шагов, вычленяя юзкейсы уровня Subfunction). Используя концепцию «Расслоения знаний» **с каждым шагом связывается информация из других слотов**. Все вышеописанные действия выполняются непосредственно в RequisitePro. Там же размещаются все выделенные ОСП (отслеживаемые сущности проекта). **Нарезка требований выполняется только после получения текстов высокого качества**. Важнейший элемент в рассматриваемом подходе к документированию – дерево трассировки RequisitePro. **Дерево трассировки** даёт возможность из узла, который представляет юзкейс уровня User-goal, **получить доступ к любой информации**, который каким-либо образом связан с этим юзкейсом.

Примеры используемых ОСП в виде диаграммы классов UML

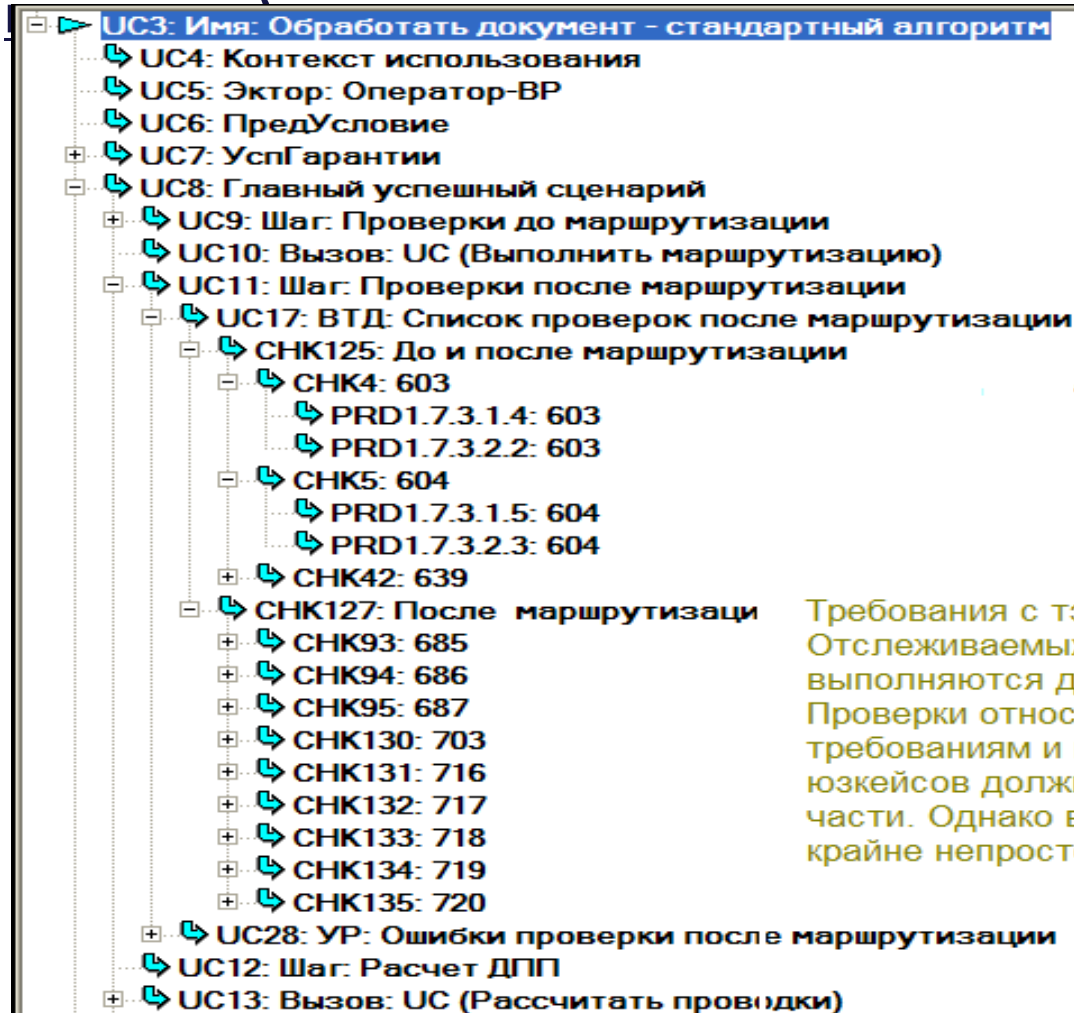


Нарезка требований для фрагмента одного из юзкейсов

[UC8 Главный успешный сценарий] (Main Success Scenario):

1. [UC9 Система подтверждает правильность документа, выполняя проверки перед маршрутизацией документа.]
2. [UC10 UC (Выполнить маршрутизацию).]
3. [UC11 Система подтверждает правильность документа, выполняя проверки после маршрутизации документа.]
4. [UC12 Система рассчитывает ДПП.]
5. [UC13 UC (Рассчитать проводки).]
6. [UC14 Система подтверждает, что проводки не были заданы при ручном вводе документа.]
7. [UC20 Система подтверждает возможность продолжения обработки.]
8. [UC21 Система отображает документ с ранее обнаруженными (в шагах 1 - 6) нефатальными ошибками и предоставляет возможность изменить проводки.]
9. [UC22 Оператор-ВР выбирает продолжение работы без изменения проводки.]
10. [UC23 Система не подтверждает необходимость дополнительного утверждения документа (из-за ранее обнаруженных нефатальных ошибок).]
11. [UC24 Система подтверждает корректность проводок документа, выполняя соответствующие проверки.]

Фрагмент дерева трассировки для одного из юзкейсов (ОСП –



Требования с тэгом CHK - пример Отслеживаемых сущностей проекта. Проверки выполняются до и после маршрутизации. Проверки относятся к функциональным требованиям и по всем правилам написания юзкейсов должны упоминаться в их сценарной части. Однако в данном случае сделать это крайне непросто. Число проверок около 150.

Для больших и сложных проектов работы, связанные с написанием юзкейсов, нарезкой требований и установлением трассировочных отношений являются **чрезвычайно трудоёмкими**, требуют много времени и большой аккуратности. Кроме того, нельзя не отметить большую трудоёмкость, связанную с поддержанием данной информации в актуальном состоянии.

Тем не менее, полученный результат может с лихвой перекрыть все усилия, потраченные на представление подобным образом знаний о предметной области. **Потратив часы, дни и даже недели, вы сможете пользоваться хорошо организованной и легко доступной информацией месяцы и годы.**

Следует отметить, что успех в значительной степени зависит от мастерства в части написания Эффективных юзкейсов и умения организовывать большой объём информации вокруг них.

- 1. Alistair Cockburn, 2001,
Writing Effective Use Cases, Reading, MA: Addison-Wesley.**
- 2. Karl Bittner, Ian Spence, 2002.,
Use Case Modeling. 2nd Edition, Addison Wesley.**
- 3. Craig Larman, 2004,
Applying UML and Patterns: An Introduction to Object-Oriented
Analysis and Design and Iterative Development, Third Edition,
Addison Wesley Professional.**
- 4. Per Kroll, Philippe Kruchten, 2003,
The Rational Unified Process Made Easy: A Practitioner's Guide to the
RUP, Reading, MA.: Addison-Wesley.**