Николай Киреев
info@webmax.by

Студия
WebMax.BY

Методическое пособие по бизнес- и системному анализу

ПРАКТИЧЕСКИЙ АНАЛИЗ С МОДЕЛИРОВАНИЕМ НА UML

Минск - 2014

www.webmax.by

ДЛЯ КОГО ПРЕДНАЗНАЧЕНО ДАННОЕ ПОСОБИЕ?

- для студентов высших учебных заведений, слушателей курсов переподготовки и повышения квалификации, изучающих технологии анализа и проектирования программных систем;
- для участников тренинга «Практический анализ с моделированием на UML»;
- для желающих получить практические навыки в области бизнес- и системного анализа и методик визуального моделирования на языке UML;
- для всех, кто хочет расширить свои представления о профессии бизнес- и системного аналитика.

ОСНОВНАЯ ЦЕЛЬ ДАННОГО ПОСОБИЯ

Имеется множество книг, пособий и информационных ресурсов, посвященных унифицированному языку моделирования UML, теории объектно-ориентированного подхода, моделям и методологиям разработки ПО, методам визуального моделирования, бизнес-анализу и анализу требований, архитектуре и конструированию программных систем.

Список некоторых приводится ниже на странице 31 в таблице 1 с краткими пояснениями и ссылками.

Проблема состоит в том, что каждое из них подробно описывает различные аспекты теории и дает необходимые знания в области анализа и моделирования, однако, из-за отсутствия у обучаемых практики, они не могут в достаточной мере их использовать в практической работе над проектом. Теория оказывается оторванной и существующей независимо от практики…

Основная цель данного пособия – изучение теоретических аспектов и методологии анализа с использованием визуального моделирования (на языке UML), в процессе работы над небольшим проектом по разработке прикладного промышленного ПО.

Весь проект разбивается на шесть практических работ, каждая из которых посвящена определенному этапу анализа и моделирования ПО и описана в соответствующем разделе данного пособия. Архитектура разрабатываемой программной системы создается путем последовательной трансформации визуальных моделей предметной области в модели системного анализа.

ГЛОССАРИЙ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Прежде, чем переходить к разработке реального проекта, уточним смысл некоторых базовых понятий, лежащих в основе ***программной инженерии*** – дисциплины, предусматривающей применение определенного систематического измеримого подхода при разработке, эксплуатации и поддержке программного обеспечения (ПО) [*Википедия*].

Программирование *– процесс отображения определенного множества целей (требований) на множество машинных команд и данных, интерпретация которых на компьютере или вычислительном комплексе обеспечивает достижение поставленных целей* [2].

Процесс разработки *ПО – совокупность процессов, обеспечивающих создание и развитие ПО.*

Модель разработки *ПО – формализованное представление процесса разработки.*

Жизненный цикл *ПО (ЖЦ ПО) – период времени, который начинается с момента принятия решения о необходимости создания программного продукта и заканчивается в момент его полного изъятия из эксплуатации.*

Цель разработки *ПО – разработать качественное ПО максимально полно удовлетворяющее реальным потребностям (целям) пользователей, уложившись в определенный срок и бюджет.*

Архитектура *(architecture)* **–** логическая и физическая структура системы, сформированная всеми стратегическими и тактическими проектными решениями.

Процесс графического представления объектной модели с помощью некоторого стандартного набора графических элементов называется ***визуальным моделированием***.

Бизнес-логика (*domain logic*) — совокупность правил, принципов, зависимостей поведения объектов предметной области (области человеческой деятельности, которую система поддерживает). Является синонимом термина «*логика предметной области*». Проще говоря, бизнес-логика — это реализация поведения объектов предметной области в программной системе. К ней относятся, например, формулы расчёта ежемесячных выплат по ссудам (в финансовой индустрии), автоматизированная отправка сообщений электронной почты руководителю проекта по окончании выполнения частей задания всеми подчиненными (в системах управления проектами), отказ от отеля при отмене рейса авиакомпанией (в туристическом бизнесе) и т. д. [*Википедия*]

ОБЪЕКТНО-ОРИЕНТИРОВАННЫЙ ПОДХОД

Как следует из приведенных выше определений, в основе любой разработки ПО лежат *цели пользователей* (их пожелания, потребности, требования, функциональные обязанности и т.д.), которые они планируют достичь при помощи программной системы, обладающей определенным поведением или *функциональностью*. С выявления, описания и моделирования требуемой функциональности программной системы начинается разработка конкретной архитектуры ПО, независимо от выбранной модели процесса разработки: водопадная, итерационная инкрементальная или гибкая.

Выбор типа элементов из которых будет создана архитектура системы и, которые в свою очередь, будут реализованы в программном коде, определяет как саму архитектуру, так и подходы к ее моделированию и разработке, а также выбор языка программирования. Современное ПО отражает реалии окружающего нас *объектного мира* и для него наиболее перспективным и популярным является объектно-ориентированный подход (ООП), поэтому архитектура, создаваемая в процессе работы над проектом, тоже является объектно-ориентированной. Теория ООП выходит за рамки данного пособия. Она широко описана в литературе [2, 4], но для дальнейшего понимания хода процесса разработки и логики построения моделей полезно учитывать следующие положения [2].

- *Объектно-ориентированный анализ* (OOA) — методология, при которой требования к системе воспринимаются с точки зрения классов и объектов, выявленных в предметной области.
- *Объектно-ориентированное проектирование* (OOD) — методология проектирования, объединяющая процесс объектной декомпозиции и приемы представления логической и физической, а также статической и динамической моделей проектируемой системы.
- *Объектно-ориентированное программирование* (OOP) — методология программирования, основанная на представлении программы в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.
- *Поведение* (функциональность) объектно-ориентированной ПС реализуется взаимодействующими объектами, которые относятся к связанным между собой классам ПС.
- Взаимодействие объектов описывается *клиент-серверной моделью*: объект «*клиент*» посылает *сообщение* объекту «*сервер*» с целью вызова соответствующей операции.

ТРЕБОВАНИЯ И ИХ КЛАССИФИКАЦИЯ

В основе любой программной системы, а также процесса её разработки лежат требования к ПО. К. Вигерс в [1] предложил следующую их классификацию.

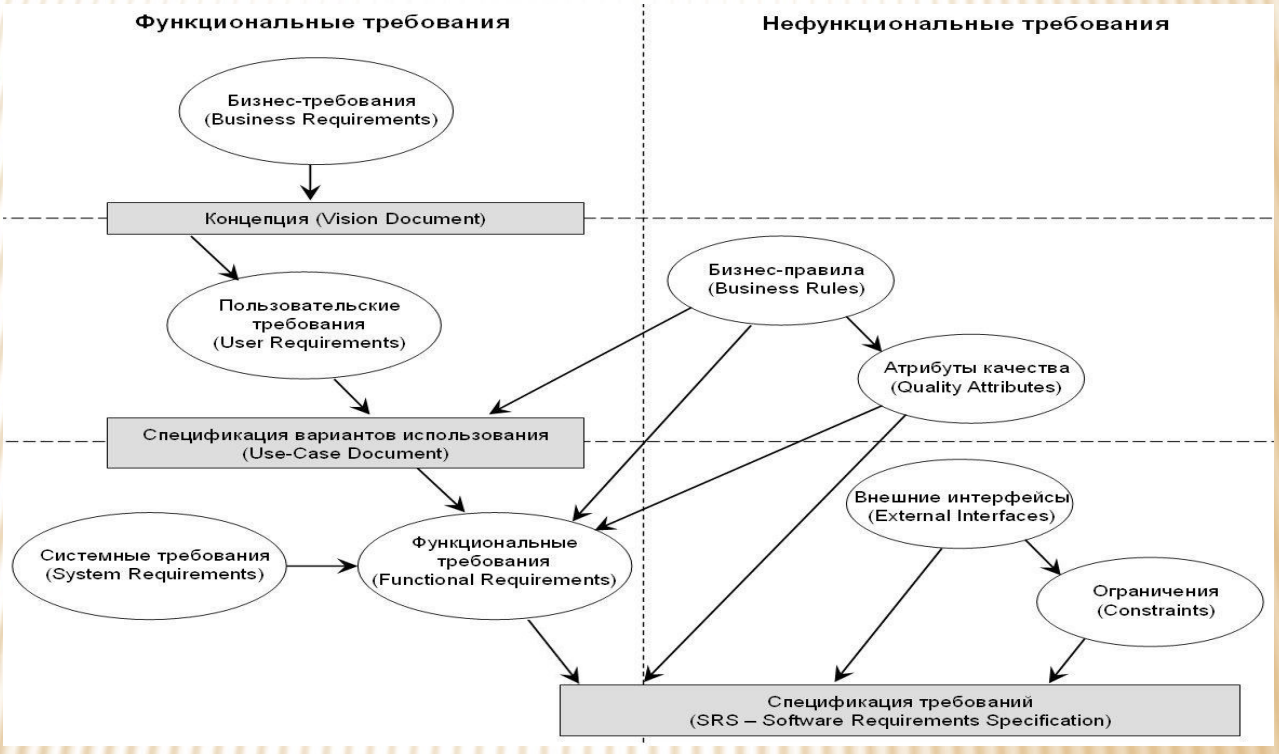

Рис.1. Классификация требований К. Вигерса

Группа функциональных требований является наиболее сложной с точки зрения их выявления, детализации и управления.

Бизнес-требования (Business Requirements) – определяют высокоуровневые цели организации/клиента (потребителя) – заказчика разрабатываемого ПО или его бизнеса.

Пользовательские требования (User Requirements) – описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями в рамках поддерживаемого бизнес-процесса.

Функциональные требования (Functional Requirements) – определяют функциональность (поведение) программной системы, которая должна обеспечить пользователям исполнение их обязанностей в контексте пользовательских требований.

ТРЕБОВАНИЯ И ИХ КЛАССИФИКАЦИЯ

Отличия пользовательских и функциональных требований проявляются уже на этапе их выявления. Пользовательские требования *(User Requirements)* соответствуют конструкции:

<действующее лицо> должно делать <действие>

Системные и/или функциональные требования (*System Requirements* &/or *Functional Requirements*) соответствуют конструкции:

<система> должна делать <действие>

Пользовательские требования описывают более общие сервисы с точки зрения самих пользователей, а функциональные требования или функции – это их конкретная реализация, рассматриваемая со стороны архитектуры системы.

Пользовательские требования должны быть очевидны и понятны пользователям, а функции системы зачастую скрыты от них. Функциональные требования используют разработчики, которые в процессе проектирования на каждую функцию создают соответствующую архитектуру, а затем, на этапе кодирования, реализуют её в программном коде.

В приведенной выше классификации (рис. 1) К. Вигерс предлагает создавать спецификацию и, соответственно, модель вариантов использования на основе пользовательских требований. Данный подход является классическим, но обладает своими недостатками. Часто такие варианты использования оказываются слишком абстрактными и с трудом могут быть использованы для дальнейшего проектирования архитектуры системы.

Другой подход – создавать модель вариантов использования на основе функциональных требований. При этом отдельные функции системы следует объединять в более общие функциональные сервисы, которые инициируются непосредственно пользователями. Проектировать архитектуру, реализующую конкретную функциональность проще чем ту, которая реализует более абстрактные пользовательские требования, однако, при таком подходе возникают другие крайности – излишняя детализация и скатывание к функциональной декомпозиции, а также сложность выявления функций системы могут свести на нет все его преимущества.

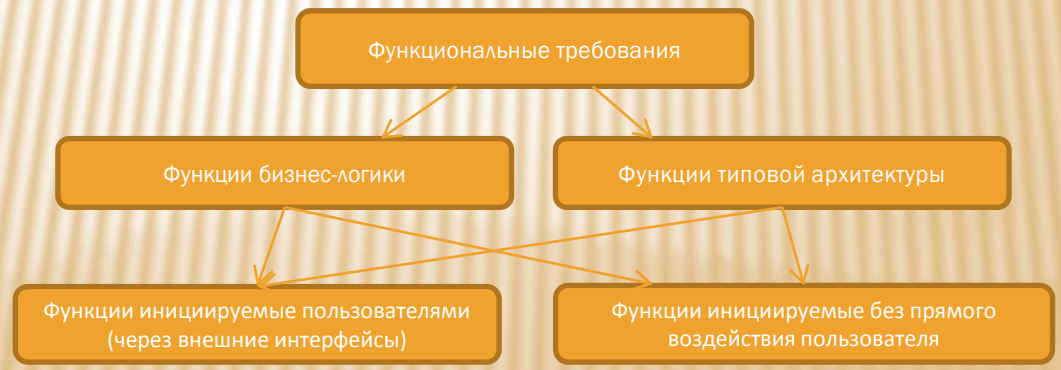

Рис.2. Функции бизнес-логики и типовой архитектуры

ОТ МОДЕЛИ К КОДУ

Основной целью разработчиков является *создание качественного ПО, максимально удовлетворяющего реальным требованиям пользователей, уложившись при этом в оговоренные сроки и бюджет* [3]. Главное – вовремя получить рабочий код, реализующий требуемую функциональность, а не формальное исполнение некоего процесса или построение набора моделей.

В идеале, быстрым решением может стать непосредственное написание кода, с минимальными затратами времени на моделирование и разработку архитектуры системы. Такая практика используется в гибких процессах разработки (XP, Scrum и т. д.), но обладает рядом существенных ограничений на использование в крупных проектах с большим количеством участников.

Альтернативой является так называемое MDA (*Model Driven Architecture*) – разработка ПО в процессе последовательной трансформации моделей. На рисунке ниже приведена модель MDA из литературы [3] с небольшими текстовыми дополнениями, поясняющими смысл последующих действий в реальном проекте.

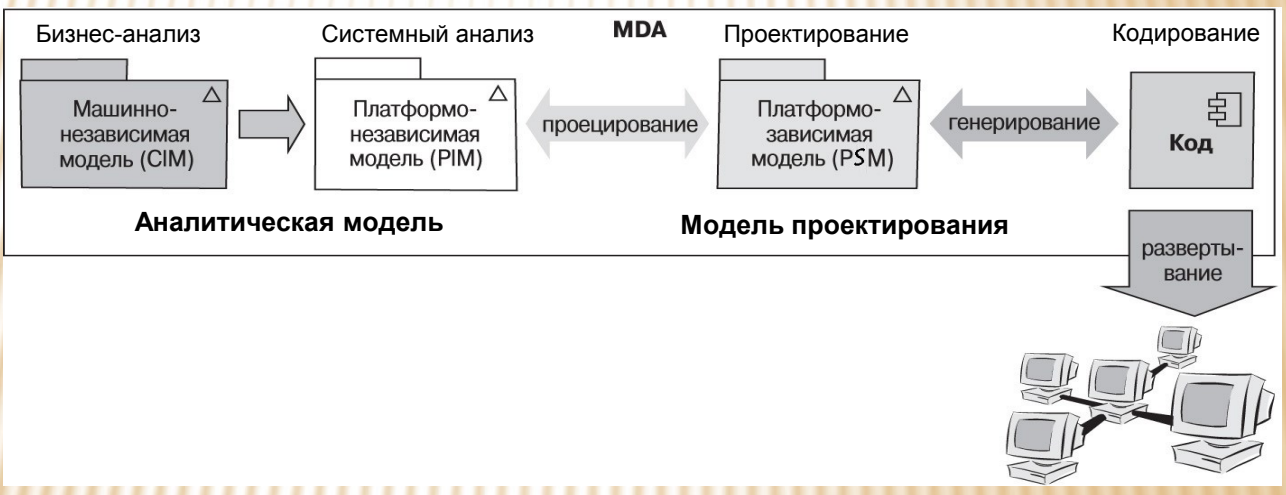

Рис.3. Модель MDA

АНАЛИТИЧЕСКАЯ МОДЕЛЬ

Согласно [3]: *«**Аналитическая модель** – это точное, четкое представление задачи, позволяющее отвечать на вопросы и строить решения.*
На этапе проектирования вы будете ссылаться именно на аналитическую модель, а не на исходную формулировку задачи.»
Аналитическая модель включает модель предметной области *(domain model)* и модель программной системы *(application model).*
Каждая модель (UML-диаграмма) отражает предметную область или программную систему с определенной точки зрения и помещается в предназначенное для нее представление.

Основные представления аналитической модели

1. **Представление классов (Logical View)**. Моделируем: сущности предметной области (*business entity*), классы анализа (*boundary, entity, controll*);
2. **Представление процессов & состояний (Process View).** Моделируем: бизнес-процессы, последовательности действий в вариантах использования, алгоритмы операций;
3. **Представление прецедентов (Use Case View).** Моделируем: варианты использования (*use-case*), пользователей (*actor*), объекты классов анализа, их связи и взаимодействие.
4. Представление размещения (**Deployment View**). Моделируем: пространственное распределение физических модулей системы.

ОБЩИЕ РЕКОМЕНДАЦИИ ДЛЯ МОДЕЛИРОВАНИЯ

Кроме принципов ООП следует принимать во внимание общие рекомендации по построению визуальных моделей.

- Создаваемые модели должны быть полезны заинтересованным сторонам: пользователям (заказчикам) и разработчикам (проектировщикам, архитекторам и т.д.). Перед построением каждой диаграммы необходимо уточнить её конкретные цели, а также для кого из участников проекта она предназначена. Не следует строить диаграммы «по аналогии» с другими проектами, так как каждый из них по сути уникален и требует своих решений. Применение шаблонов и паттернов оправдано при проектировании архитектуры, реализующей стандартные функции системы (прием-передача данных, сохранение данных и т. д.), но не при моделировании предметной области или функций бизнес-логики;
- Модели должны быть понятными, читабельными, исключающими неоднозначную интерпретацию представленной в них информации. Если на диаграмме много элементов (более 20 – 30) и сложные связи, то её следует упрощать, разбивая на несколько более простых диаграмм, либо жертвовать излишней детализацией;
- В моделях, в соответствующих представлениях, фиксируется всё, что непосредственно связано с самой программой и ее функционированием, а также с бизнесом или бизнес-процессом, который она обеспечивает;
- Модели, как и некоторые другие артефакты, например, глоссарий, создаются итерационно. Даже опытному аналитику с первого раза редко удается построить удачную модель, адекватно передающую пользователю всю необходимую информацию;
- Количество создаваемых артефактов должно быть достаточным, чтобы отразить все важные аспекты разрабатываемой ПС и минимальным, чтобы максимально сократить время, затрачиваемое на этапы анализа и проектирования.
- Качество работы аналитика должно определяться не по количеству создаваемых им артефактов, а по тому насколько быстро и адекватно воспринимается смысл требований бригадой разработчиков и как быстро можно перейти к написанию программного кода.

ПРОТОТИПИРОВАНИЕ ЭКРАННЫХ ФОРМ

Прототипы экранных форм позволяют эффективно выявлять и детализировать отдельные функции бизнес-логики системы, инициируемые пользователями через графический интерфейс. С их помощью можно представить и смоделировать сценарии взаимодействия пользователей и программной системы.

Они гораздо понятней заказчикам и пользователям при обсуждении функционала программы, чем, например, диаграммы вариантов использования.

В классических подходах, таких как RUP, рекомендуется создавать прототипы в виде рисунков или статических экранных копий на этапе описания потоков событий в сценариях вариантов использования. При этом они рассматриваются лишь как дополнительное средство, поясняющее процесс взаимодействия пользователя и системы, а не как средство выявления и документирования функциональности системы.

В современных реалиях, при использовании ***Agile***-методологии участники проекта стремятся сократить время разработки и как можно быстрее получить рабочий программный код, пренебрегая многими артефактами анализа и проектирования, уменьшая временные затраты на документацию и моделирование. В этом случае, построение и использование прототипов экранных форм оказывается едва ли не самой эффективной методологией для выявления и документирования функциональности системы.

Особую помощь в этом могут оказать популярные средства создания *динамических прототипов* ***Axure Pro*** и ***Prototyper***. В этих оболочках можно создавать прототипы, практически полностью имитирующие не только содержание экранных форм, но и всю динамику их взаимодействия с пользователями системы. Обычно, эффектные динамические прототипы приводят заказчика в восторг – он видит воплощение своих идей и пожеланий в «работающем» макете, его мало интересуют написанные вами ТЗ и построенные UML-диаграммы, зато он охотно готов обсуждать и участвовать в разработке дизайна каждой экранной формы: где и какого размера должно быть текстовое поле, комбо-бокс или радио-кнопка, как разбить прототипы форм на страницы и т. д. Многие команды, использующие методологию Agile (в частности Scrum) с удовольствием этим пользуются.

Но, к сожалению, здесь кроется основная опасность – чрезмерное увлечение детальными динамическими прототипами, полностью имитирующим логику работы приложения приводит к затягиванию сроков. Заказчику требуется больше и больше – он хочет, чтобы был реализован не только ввод данных, но и сопутствующие математические функции, а также была реализована логика, когда и какой элемент экранной формы (текстовое поле, кнопка, список и т. д.) доступен для воздействия пользователем (*enabled*) или нет (*disabled*). Ему нужно, чтобы всё было, как в рабочей программе! Но возможности любых прототипов ограничены, а излишняя детализация логики работы требует больших временных затрат.

ИНСТРУМЕНТАРИЙ ДЛЯ АНАЛИЗА

Важен ли инструментарий, который может быть использован аналитиками для моделирования?

Существует множество инструментов в которых, используя нотацию языка UML, можно строить визуальные модели как предметной области, так и программной системы. Их спектр – от простых графических редакторов (типа *MS Visio*) до сложных систем автоматизированного проектирования с генерацией программного кода на выбранном языке программирования. Какой инструментарий выбрать?

В принципе, аналитик может строить и обсуждать модели с заказчиком даже на салфетках, сидя за чашкой кофе в уютном кафе, поэтому один из четырёх принципов манифеста гибкой (Agile) разработки [*www.agilemanifesto.org*] гласит: «*Люди и взаимодействие* ***важнее*** *процессов и инструментов*». Не так важны процессы и инструменты, которые используются в разработке, как взаимопонимание между заказчиками и исполнителями и одинаковое представление о программном продукте, который создается совместными усилиями.

Однако, всё же будет лучше, если рабочий инструмент аналитика будет обладать следующим минимальным набором возможностей:

- иметь нотацию (стереотипы) для раздельного моделирования предметной области (машинно-независимых моделей CIM), архитектуры системного анализа (платформо-независимых моделей PIM) и для проектирования архитектуры программной системы (платформо-зависимой модели PSM);
- позволять вводить и сохранять спецификации элементов модели;
- модели, предназначенные для обсуждения с заказчиками и пользователями должны быть простыми интуитивно-понятными всем, кто никогда не сталкивался с языком UML;
- позволять строить модели с позиции, как минимум, трех основных представлений: логической структуры (Logical View сущности предметной области, классы программной системы), функциональности (Use Case View), а также процессов или состояний (Process View);
- обеспечивать принципы модульности и поддерживать иерархичность, как в самих моделях, так и в организационной структуре хранения и поиска информации. Каждый создаваемый в процессе разработки артефакт должен храниться в определенном месте, а участники проекта иметь возможность его легко найти;
- поддерживать выбранный процесс разработки.

Полностью всем перечисленным требованиям отвечает среда разработки *Rational Rose 2003 (RR). С*овременным её развитием является *Rational Software Architect (RSA)* версии 7.5 с поддержкой UML 2.1. Кроме всех остальных достоинств RR обладает интуитивно-понятным интерфейсом, практически не требующим времени на его изучение. Поэтому, дальнейший процесс разработки будет описан на примере RR, но при желании, может быть использован любой другой инструмент визуального моделирования, обладающий, по меньшей мере, теми же возможностями, что и предложенный. Подробное описание интерфейса RR представлено в [5].

РАБОТА НАД ПРОЕКТОМ

В качестве учебного предлагается небольшой проект, выполнявшийся по заказу немецкой фирмы, занимающейся переплавкой металлолома. Концепция системы представлена в приложении 1.
Используя описание, представленное в приложении, последовательно выполняются шесть лабораторных работ, каждая из которых относится к определенному разделу анализа, а ее результатом будет создание визуальных моделей в соответствии с методологией RUP.

I. Основы бизнес-анализа

Лабораторная работа № 1
- Глоссарий предметной области.
- Бизнес-цели программной системы.
- Определение действующих лиц и сущностей предметной области.
- Моделирование сущностей предметной области и их атрибутов.

Лабораторная работа № 2
- Построение модели действующих лиц и их функциональных обязанностей (модель штатной структуры).
- Моделирование бизнес-процессов.

II. Анализ требований

Лабораторная работа № 3
- Выявление и документирование пользовательских требований.
- Трассирование: *User Requirements — Functional Requirements — Use Cases*.

Лабораторная работа № 4
- Модель вариантов использования. Назначение и особенности построения use case diagram.

III. Основы системного анализа

Лабораторная работа № 5
- Сценарии, их написание и предназначение.
- Прототип графического интерфейса пользователя, его связь со сценариями.
- Моделирование потоков событий (activity diagram).

Лабораторная работа № 6
- Моделирование взаимодействия и структуры связей объектов бизнес-логики (sequence diagram и collaboration diagramm).
- Моделирование логической структуры (class diagram). Классы анализа (entity, control, boundary), их предназначение и отличие от классов программной системы.

ОРГАНИЗАЦИОННАЯ СТРУКТУРА ПРОЕКТА

Организационная структура проекта это своего рода «шкаф с полками», предназначенный для упорядоченного ввода и хранения артефактов. Без её создания и поддержки любой проект превратится в хаос.

В крупных проектах организационная структура поддерживается с помощью специального программного обеспечения, в данном небольшом учебном проекте её роль будет играть возможности браузера RR (Рис.4). Главное – используемая при разработке информация должна храниться в определенном месте, обеспечивающем её лёгкий интуитивный поиск.

Для создания организационной структуры, после входа в RR, следует загрузить шаблон *rational unified process*, или отказавшись от шаблонов (нажатием клавиши *Cancel* в окне выбора шаблонов), самостоятельно создать соответствующие папки в представлении Use-Case View (папки Business Use-Case Model, Use-Case Model) и в представлении Logical View (папки Business Object Model, Analysis Model и Design Model).

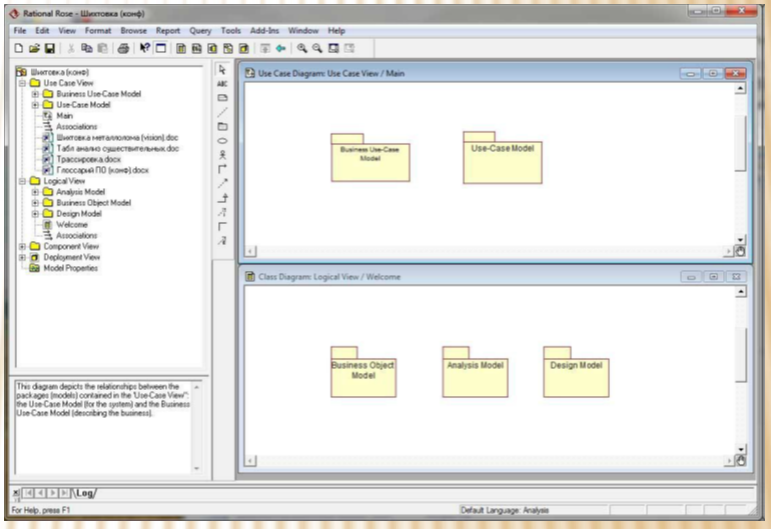

Рис.4. Организационная структура проекта в Rational Rose

Папка Business Use-Case Model предназначена для моделей предметной области, отражающих действующих лиц и их функциональные обязанности или требования. В этой папке моделируются пользовательские требования (User Requirements). К ней также можно будет подкреплять файлы исходных артефактов (концепция, интервью с инвесторами, пользователями и т.д.).

В папке Use-Case Model будут храниться диаграммы, отражающие пользователей и инициируемые ими функциональные сервисы программной системы. Таким образом, в ней моделируются функциональные требования (Functional Requirements).

В папке Business Object Model размещаются диаграммы, отражающие сущности предметной области (business entity) и их атрибуты.

Папка Analysis Model предназначена для диаграмм системного анализа, показывающих структуру классов программной системы (entity, control, boundary), отвечающих за функции бизнес-логики. Это папка системного аналитика.

В папке Design Model размещаются диаграммы классов, включая структуру их взаимосвязей для всей программной системы. Это папка проектировщика.

ЗАДАЧИ БИЗНЕС-АНАЛИЗА

Бизнес-анализ – это этап разработки, основными задачами которого являются: уточнение назначения, области применения и целей (*Business Requirements)* разрабатываемой программы, определение заинтересованных лиц (*stakeholders*), выявление и анализ пользовательских требований (*User Requirements*), изучение и моделирование предметной области. Моделирование предметной области делается опционально, по мере необходимости. Эти модели предназначены для команды разработчиков, так как заказчик, сам является экспертом в предметной области.

При этом, для получения достаточно полного и адекватного представления о предметной области, анализ и моделирование следует проводить как минимум по трем основным направлениям:

- изучение сущностей предметной области и их иерархий (в программной системе они найдут отражение в виде классов и логической структуры баз данных),
- изучение участников бизнеса (функционеров, в UML они называются *Business Actors*) и их функциональных обязанностей (по сути – это пользовательские требования, в UML они отражаются как *Business Use Cases*), на их основе разрабатываются функциональные требования к программной системе,
- изучение и моделирование бизнес-процессов или состояний и условий перехода из одного состояния в другое (так как не всё можно описать в виде процессов или алгоритмов).

Моделирование бизнес-архитектуры

Кроме основных моделей анализа на этапе бизнес-анализа часто определяется и моделируется наиболее общая структура модулей программной системы и связей между ними, так называемая ***бизнес-архитектура***, в которой оговариваются платформы, межмодульные интерфейсы и язык программирования на котором будет реализована система. Эти модели могут быть построены в представлении ***Deployment View*** с помощью одноименной диаграммы, но чаще, для их построения используют более наглядные средства, например, *MS Visio*. Примеры моделей бизнес-архитектуры, выполненные на UML и в MS Visio представлены на рисунках 5 и 6 ниже. Более подробное рассмотрение разработки и построения моделей бизнес-архитектуры выходит за рамки настоящего пособия.

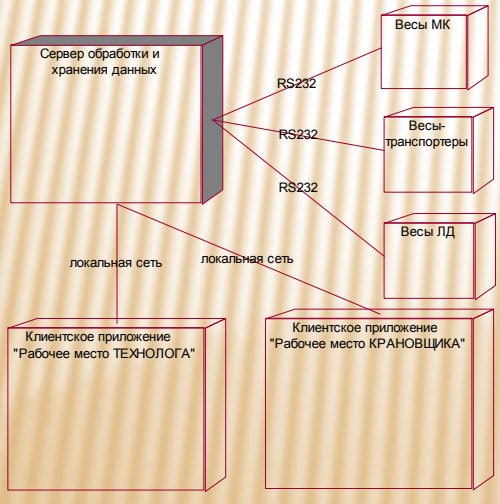

Рис. 5. Модель бизнес-архитектуры на UML

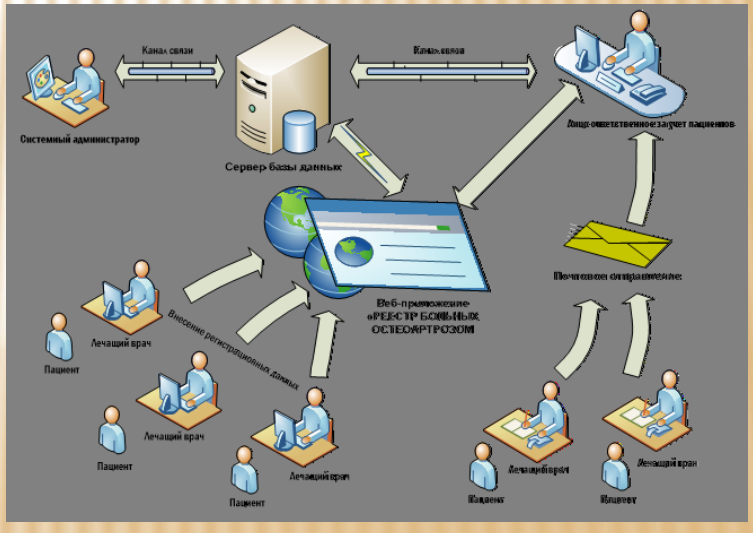

Рис. 6. Модель бизнес-архитектуры в MS Viso

ОСНОВЫ БИЗНЕС-АНАЛИЗА. ЛАБОРАТОРНАЯ № 1

Темы: *Бизнес-требования (business requirements)*
Глоссарий
Сущность предметной области (business entity)
Иерархии сущностей

Согласно [1, 6] **бизнес-требования** это высокоуровневые цели программной системы. Они должны быть уточнены и согласованы с заинтересованными лицами (заказчики, инвесторы), проверены на соответствие законодательству, нормам и правилам. В дальнейшем, все пользовательские требования *(user requirements)* должны проверяться на соответствие бизнес-требованиям, а выявленные противоречия устраняться.

> Так как бизнес-требования достаточно важные артефакты, в соответствие с которыми согласуется функциональность ПО, они требуют не формального подхода. Не следует отписываться общими фразами типа «*…с целью автоматизации*», «*…ускорения*» и т. д. Для выявления бизнес-требований следует задать простой вопрос: «За что конкретно заказчик готов заплатить деньги?».

Глоссарий – текстовой документ, разъясняющий разработчикам специфические термины (в том числе жаргонные), встречающиеся в данной предметной области, но с позиций разрабатываемой программной системы. В глоссарии документируются синонимы, омонимы и факты смены терминологии. Глоссарием пользуется в основном команда разработчиков. Заказчик сам является экспертом предметной области, но может им тоже воспользоваться для достижения единства понятий и терминологии.

Глоссарий не должен дублировать информацию введенную в других артефактах, например, в спецификациях на диаграмме сущностей предметной области. Информация вводится в систему всегда однократно.

> При написании глоссария следует помнить, что его цель это не повышение IQ участников проекта, и не какое-то формальное определение терминов, взятое из словарей, википедии или найденное через интернет-поисковики, а их разъяснение с точки зрения разрабатываемой программной системы.

Пример

Термин **шихтовка** и связанные с ним **шихтовать**, **шихта**, взятые из энциклопедических словарей [*enc-dic.com*]:

> ***Шихтовка*** *-и, ж. спец. Действие по знач. глаг. Шихтовать;* ***Шихтовать*** *-ту́ю, -ту́ешь; прич. страд. прош. шихто́ванный, -ван, -а, -о; несов., перех. спец. Составлять шихту;* ***Шихта*** *(от нем. Schicht) - смесь сырых материалов, а в некоторых случаях (напр., при выплавке чугуна в доменной печи) и топлива, подлежащая переработке в металлургич., хим. и др. агрегатах. Ш. загружают либо в виде равномерной смеси, подготовл. вне агрегата, либо порциями или слоями, состоящими из отд. её составных частей.*

В нашем случае, эти определения, возможно, повысят кругозор команды разработчиков, но они не отражают смысл, вложенный в этот термин заказчиком и не связаны с предметом разработки как, например, формальное определение типа: «*Шихтовка – это процесс составления плавильной смеси*». Поэтому будет предпочтительнее следующее: «*Шихтовка (шихтовать) – это процесс составления плавильной смеси, в котором обеспечивается заданная пропорция набираемых металлических компонентов (МК) и контролируется подача кокса, извести и спецкокса, в соответствии с набранным суммарным весом МК».*

Понятия плавильная смесь, кокс, известь, спецкокс, заданная пропорция МК, суммарный вес МК в глоссарий вноситься не будут, т.к. они должны быть введены и описаны в спецификациях на диаграмме сущностей предметной области.

ОСНОВЫ БИЗНЕС-АНАЛИЗА. ЛАБОРАТОРНАЯ № 1

Бизнес-сущность (сущность предметной области, business entity) — некий объект или концептуальное понятие предметной области, характеризуемое набором существенных признаков (данных), прямо или косвенно связанное с проектируемой программной системой.

> Здесь имеется ввиду, что одни бизнес-сущности найдут отражение непосредственно в классах программной системы, а другие, участвуя в бизнес-процессе, не будут отражены в виде класса, но могут влиять на работу программы. И те, и другие следует моделировать на диаграмме сущностей предметной области и дополнять соответствующими спецификациями.

Бизнес-сущности – это кандидаты в классы и объекты ПС, которые отвечают за ввод, изменение, удаление и хранение данных (атрибутов). Выявление бизнес-сущностей, а вместе с ними набора характеризующих их данных, которыми будет оперировать разрабатываемая программа – главная цель данной лабораторной работы. Для моделирования бизнес-сущностей в языке UML применяется стереотип *business entity.*

Методики выявления бизнес-сущностей могут быть разные (интервью с пользователями, наблюдение на рабочих местах, сбор бланков и листов и т.д.). В данной лабораторной работе предлагается выявлять сущности предметной области и их атрибуты путем чтения текста концепции и анализа имён существительных. Если некая сущность участвует в бизнес-процессе и инкапсулирует данные, которыми будет оперировать разрабатываемая программа, то она должна быть зафиксирована на диаграмме, а набор ее существенных характеристик (данных) должен быть показан в виде атрибутов.

> *Инструментом* для этого служит ***КРАН***, который забирает ***МЕТАЛЛИЧЕСКИЕ КОМПОНЕНТЫ*** (МК) своим *магнитом* и потом опускает в дозировочный **КОНТЕЙНЕР**. В соответствии с ***ВЕСОМ НАБРАННЫХ МК*** подаются ***КОКС, ИЗВЕСТЬ, СПЕЦКОКС***. По заполнении контейнера его содержание со всеми занесенными в него металлическими компонентами направляется в **ЧАН** для плавления, в который добавляются ***ЛЕГИРУЮЩИЕ ДОБАВКИ***. В конце рабочего дня чан направляется в **ПЛАВИЛЬНУЮ ПЕЧЬ**. Так как в процессе плавления должны использоваться определенные соотношения веса различных МК, программа должна обеспечить требуемый состав **ПЛАВИЛЬНОЙ СМЕСИ**. Этот контроль и составление смеси компонентов для плавки называется ***ШИХТОВКОЙ***.
>
> Программная система устанавливается на рабочем месте ***КРАНОВЩИКА*** и используется им для контроля над процессом шихтовки. Кроме того, она позволяет ***ТЕХНОЛОГУ*** задать определенную ***ПРОПОРЦИЮ МК*** в конечном ***СПЛАВЕ*** и осуществлять сохранение **информации** о составе и свойствах полученных сплавов в базе данных.

Не все имена существительные в тексте будут относиться к бизнес-сущностями: одни из них обозначают общие понятия, не имеющие значения в программной системе, другие будут её внешним пользователям (у них важны не их атрибуты, а взаимодействие с ПС), третьи будут обозначать процессы, четвертые будут данными, которые должны быть показаны в виде атрибута сущности.

> **Задание**: проанализируйте имена существительные выделенные красным цветом в тексте выше. Определите, какие из этих понятий можно моделировать в виде бизнес-сущностей, какие будут их атрибутами, а какие из них представляют внешних пользователей.

ОСНОВЫ БИЗНЕС-АНАЛИЗА. ЛАБОРАТОРНАЯ № 1

На основе сущностей предметной области создаётся логическая структура программной системы (классы, логическая структура базы данных), отвечающая за бизнес-логику, поэтому модель сущностей располагается в представлении Logical View. При этом, не каждая выявленная сущность будет в последствии отображена в виде класса программной системы, а только те из них, которые инкапсулируют данные, являющимися существенными, с точки зрения разрабатываемого ПО. Сущности, значения атрибутов которых не используются в процессе функционирования ПО, но участвующие в общем бизнесе или бизнес-процессе должны быть показаны в моделях предметной области. Это позволит команде разработчиков лучше понять предметную область и процессы, происходящие в ней.

Пример. В тексте выше имеются сущности «*Чан для плавления*» и «дозировочный *Контейнер*». Они участвуют в общем технологическом процессе и их следует отобразить на диаграмме сущностей, однако, атрибутов (данных), которые будет контролировать разрабатываемая система у них может и не быть. В этом случае они не будут отображаться в классах программной системы.

Модели сущностей предметной области располагаются в папке *Business Object Model*. При построении диаграмм следует учитывать, что сущности в реальном мире и, соответственно, существительные в исходном тексте, являются объектами, а на данной диаграмме они должны показываться в виде классов **business entity** (класс, используется для моделирования сущностей предметной области).

Как и любой другой класс *business entity* долен иметь уникальное имя, список атрибутов, определяющих состояние и набор операций, реализующий поведение, создаваемых классом объектов. Для моделирования бизнес-сущностей в языке UML используется одноименный стереотип (рис. 7).

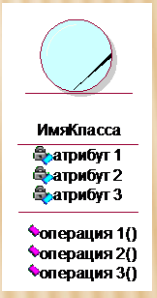

Рис.7. Класс business entity

Графическое отображение сущности и её атрибутов, необходимо дополнить текстовым описанием (документацией). В нем следует указать назначение и роль сущности в бизнес-процессе, а также будет ли она классом программной системы. Для атрибутов следует указывать их роль и ответственных за ввод/изменение и использование хранимых данных.

Связи между бизнес-сущностями. Целью данной диаграммы являются сами сущности и их атрибуты. Связи (*ассоциации*) между ними, а соответственно, и операции класса, в предметной области часто не определены и неоднозначны. Для их выявления используется методика моделирования обмена сообщениями между объектами в потоке событий (*sequence diagram*), которая описана в данном пособии ниже. Поэтому связи и, соответственно, операции классов на этих диаграммах не показываются.

Единственный тип связи, который очевиден и его можно отразить на диаграмме сущностей – это *агрегация (рис. 8)*. Она показывает отношение целого и составляющих его частей. При моделировании агрегации указывается множественность.

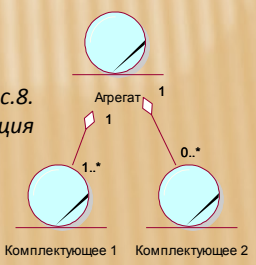

Рис.8. Агрегация

Задание для лабораторной работы

Используя описание концепции системы (Приложение 1):

- уточните бизнес-требования;
- составьте глоссарий предметной области;
- постройте диаграмму с соответствующими спецификациями.

ОСНОВЫ БИЗНЕС-АНАЛИЗА. ЛАБОРАТОРНАЯ № 2

Темы: *Заинтересованное лицо (stakeholder)*
Бизнес-актер (business actor) и бизнес-функция (business use-case);
Бизнес-процесс (business process)

Целью настоящей лабораторной работы является:

- выявление и моделирование участников бизнеса и их функциональных обязанностей (пользовательских функций);
- моделирование бизнес-процессов.

Приступая к разработке любого программного обеспечения, можно условно выделить три круга лиц, прямо или косвенно заинтересованных в результатах его функционирования:

- *Заинтересованное лицо* (***stakeholder***) – это индивидуум или группа внешних фигурантов, прямо или косвенно заинтересованных в результате создания системы. Это самый широкий круг лиц, связанных с разрабатываемой программой. В него входят как её непосредственные пользователи, так и спонсоры, финансирующие проект, а также лица и организации утверждающие нормативы, стандарты и требования, регламентирующие её функционирование. Обычно, не возникает необходимости в построении визуальных моделей для этого круга лиц, однако, с их интересами следует согласовывать высокоуровневые бизнес-требования (business requirements);
- *Участник бизнеса/бизнес-процесса* (***business actor***) – это штатная единица или группа, исполняющая свои функциональные обязанности в данном бизнесе или бизнес-процессе. Чаще всего ими могут быть штатные работники предприятий, занимающие определенные должности и исполняющие должностные обязанности (директор, главный инженер, контролер ОТК, мастер и т.д.), либо структурные подразделения (производственный отдел, сектор контроля качества и т.д.), но могут быть и внешние системы, исполняющие определенные функции (установка легирования, весы-транспортёры и т.д.). Этот круг лиц включает в себя пользователей программной системы и участников, не являющихся таковыми, но оказывающих на неё влияние. Так, например, курьер в интернет-магазине не является пользователем, но он участвует в бизнес-процессе и косвенно влияет на работу программной системы. Эти лица моделируются в представлении Use-Case View в папке Business Use-Case Model при помощи стереотипа Business Actor, а их функциональные обязанности с помощью стереотипа Business Use-Case. Построение этого типа диаграмм является одной из целей настоящей лабораторной работы.
- *Действующее лицо* (***actor***), синонимы актёр, актант – абстрактное понятие характеризует внешнего пользователя (или группу пользователей), непосредственно взаимодействующих с программной системой. Моделируются в представлении Use-Case View в папке Use-Case Model. Назначение и построение use-case-моделей рассматривается в лабораторной работе 4 настоящего пособия (см. стр. 23)

ОСНОВЫ БИЗНЕС-АНАЛИЗА. ЛАБОРАТОРНАЯ № 2

Примеры диаграмм, моделирующих бизнес-актёров и их функциональные обязанности (рис. 9)

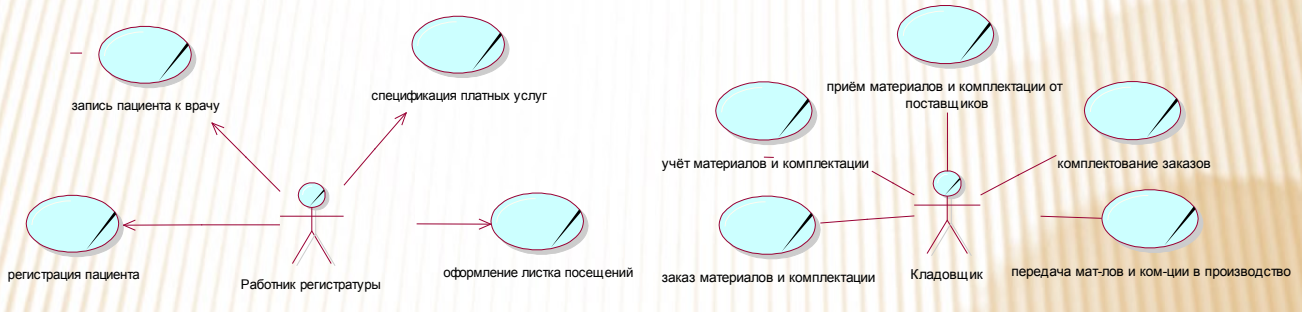

Рис.9. Примеры моделирования бизнес-актёров и их функциональных обязанностей

Бизнес-процесс – последовательность взаимосвязанных действий (в языке UML действие называется *action*) или деятельностей (деятельность – *activity*), направленная на достижение значимого бизнес-результата.

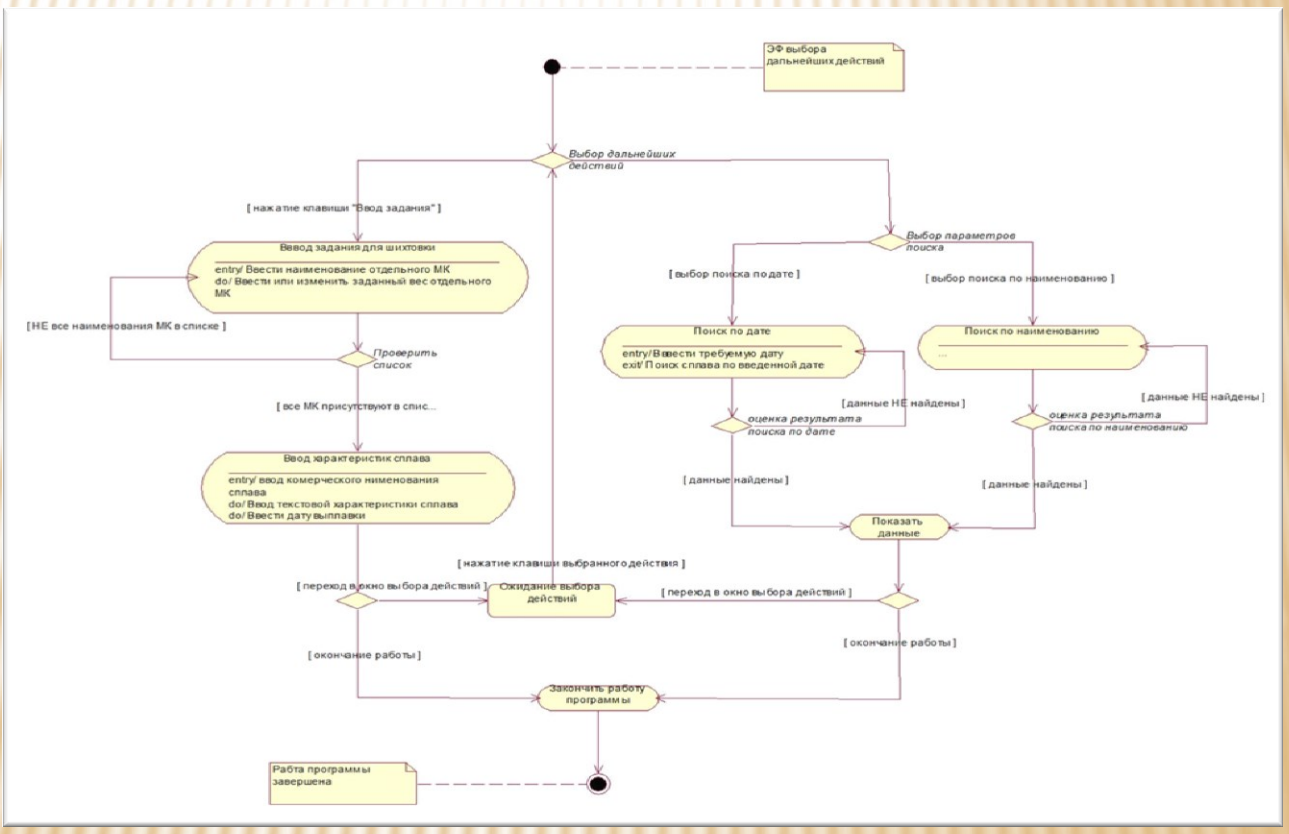

Рис.10. Пример моделирования бизнес-процесса

Задание для лабораторной

Используя текст концепции (Приложение 1):

- выявите на предметной области бизнес-актеров и определите их функциональные обязанности. В папке *Business Use Case Model* постройте модель бизнес-актеров и их функциональных обязанностей.
- представьте и смоделируйте с помощью *activity diagram* технологический процесс шихтовки металла.

АНАЛИЗ ТРЕБОВАНИЙ. ЛАБОРАТОРНАЯ № 3

Темы: *Анализ требований (requirements engineering)*
Пользовательские требования (user requirements)
Функциональные требования (functional requirements)
Вариант использования (use-case)
Трассирование (traceability)
Спецификация функциональных требований

Анализ требований *— это процесс сбора требований к ПО, их систематизации, документирования, анализа, выявления противоречий, неполноты, разрешения конфликтов в процессе разработки программного обеспечения. В англоязычной среде также говорят о дисциплине «инженерия требований»* (англ. *Requirements Engineering*) [*Википедия*].

Определения и отличия пользовательских и функциональных требований были приведены в настоящем пособии на стр. 6 и 7. Пользовательские требования связаны с обязанностями самих пользователей. Они могут извлекаться из разных источников при помощи различных аналитических методик: опросы пользователей и заинтересованных лиц (интервью, анкеты); обсуждения и семинары; наблюдение на рабочих местах; анализ сопутствующих текстовых документов (концепции, правила, стандарты, описания, прайс-листы и т.д.).

Вариант использования *(Use-Case, прецедент),* согласно Г. Бучу, – *внешняя спецификация последовательности действий, которые система может выполнять в процессе взаимодействия с актёрами (пользователями) для получения определённого значимого для них результата*.

Понятия функция системы и Use-Case разные, однако, если результат исполнения функциональности или функционального сервиса и результат выполнения варианта использования совпадают, то каждая выявленная функция или сервис системы будут соответствовать конкретному прецеденту в модели вариантов использования.

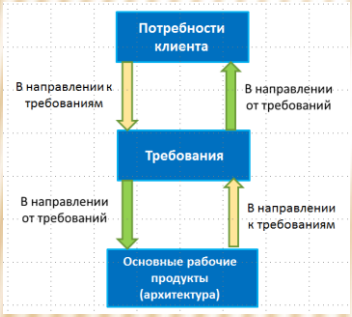

Рис.11. Прямая и обратная трассировка

Пользовательские требования, функциональные требования и варианты использования фиксируются в трассировочной таблице. Это позволяет сохранить сквозную трассировку от пользовательского требования и до структурных элементов разрабатываемой архитектуры. В соответствии с рекомендациями К. Вигерса важны как прямая, так и обратная трассировки (см. рис. 11). Трассировочную таблицу можно создать в текстовом редакторе или использовать специально предназначенную для этого программную оболочку (например, *Requisite Pro*).

Для выявления функций программной системы предлагается следующая методика. Каждое пользовательское требование, связанное с функциональностью, можно представить в виде процесса взаимодействия пользователя (актёра) и системы через соответствующий интерфейс. В случае, если пользователь физическое лицо, то взаимодействие будет осуществляться через графический интерфейс (GUI), т. е. посредством элементов экранных форм. Пользователь будет вносить данные в определённые поля, нажимать кнопки, выбирать элементы списков, инициируя таким образом функции системы, связанные с бизнес-логикой. Таким образом, пользовательскому требованию можно поставить в соответствие конкретные функции бизнес-логики.

АНАЛИЗ ТРЕБОВАНИЙ. ЛАБОРАТОРНАЯ № 3

Пример трассировочной таблицы

№	Требование Заказчика	№	Функция системы	Use-Case
1	…программа должна обеспечить требуемый состав плавильной смеси		НЕТ (абстрактное требование)	
2	Программная система устанавливается на рабочем месте крановщика и используется им для контроля над процессом шихтовки		НЕТ (абстрактное требование)	
3	она позволяет технологу задать определенную пропорцию МК в конечном сплаве	3.1 3.2 3.3	См. 5.1 См. 5.2 См. 5.3	
4	осуществлять сохранение информации о составе и свойствах полученных сплавов в базе данных	4.1	См. 23.1 – 23.7	
5	Технолог вводит в программу заданный вес по каждому МК, предназначенному для плавки, определяя, таким образом, их пропорцию в сплаве	5.1 5.2 5.3	Ввести наименование МК Ввести заданный вес Добавить в список	Ввод наименования МК Ввод заданного веса Добавить в список
6	…. коммерческое название сплава и характеристику сплава	6.1 6.2	Ввести название сплава Ввести характеристику сплава	Ввод названия сплава Ввод характеристики сплава
7	Вес поднятого краном МК определяется автоматически и вводится через последовательный порт (RS232) в компьютер	7.1	Принять протокол RS232 (вес МК)	Принять протокол RS232 (вес МК)
8	Вес кокса и извести обуславливается суммарным весом набранных в контейнер МК, который посылается на весы-транспортеры протоколом RS232	8.1 8.2	Рассчитать суммарный вес МК Послать протокол RS232 (суммарный вес МК)	Рассчитать суммарный вес МК Послать протокол RS232 (суммарный вес МК)

Рис.12. Пример трассировочной таблицы

Задание для лабораторной работы

В текстовом редакторе MS WORD создайте трассировочную таблицу, аналогичную таблице, представленной в примере выше. Файл с таблицей присоедините к папке *Use-Case Model* представления *Use-Case View*. Используя описание концепции (Приложение 1):

- выявите в тексте концепции и зафиксируйте в соответствующем столбце таблицы пользовательские требования;
- представляя сценарии использования поставьте в соответствие каждому пользовательскому требованию одно или несколько функциональных требований (функций системы), которые являются его реализацией;
- для каждой выявленной функции приведите в соответствие вариант использования и зафиксируйте его в таблице.

АНАЛИЗ ТРЕБОВАНИЙ. ЛАБОРАТОРНАЯ № 4

Темы: *Модель вариантов использования (use case model);*
Диаграмма вариантов использования (use case diagram);
Актер (actor), отношения (include, extend, generalization)

Модель вариантов использования (*use-case model*) — это модель, которая описывает функциональные требования к системе в терминах вариантов использования.

Диаграмма вариантов использования (*use-case diagram*) — это диаграмма, на которой изображены отношения, существующие между актерами (*actor*) и вариантами использования системы (*use-case*).

Актер (*actor*), синонимы актант, действующее лицо — это абстрактное понятие, которое характеризует внешнего пользователя (или нескольких пользователей), взаимодействующего с системой. Каждый актер соответствует одной роли в которой выступает пользователь, взаимодействуя с системой.

Обобщение (*generalization*) — это отношение связывает специализированный и более общий варианты использования.

Включение (*include use case*) — указывает на включение последовательности поведения варианта использования-поставщика в последовательность взаимодействия варианта использования-клиента.

Расширение (*extend use case*) — позволяет базовому варианту использования при определенном условии использовать функциональность варианта использования-клиента.

Рекомендации для построения модели ВИ:

- каждый ВИ должен быть инициирован актером или вступать в отношения «*include*», «*extend*» к другим ВИ;
- ВИ должны концентрироваться на том «ЧТО», а не «КАК» должна делать система;
- следует избегать функциональной декомпозиции (рис. 13). Она не подходит для моделей ВИ [3];
- предпочтительнее простая модель ВИ без отношений «*include*» и «*extend*»;
- распределяйте модели ВИ по папкам (Package), а не старайтесь показать всё на одной диаграмме.
- для каждого актёра следует создать отдельную папку и в ней показать инициируемые им функциональные сервисы.
- общие функциональные сервисы, предоставляемые нескольким актёрам лучше показать отдельно.

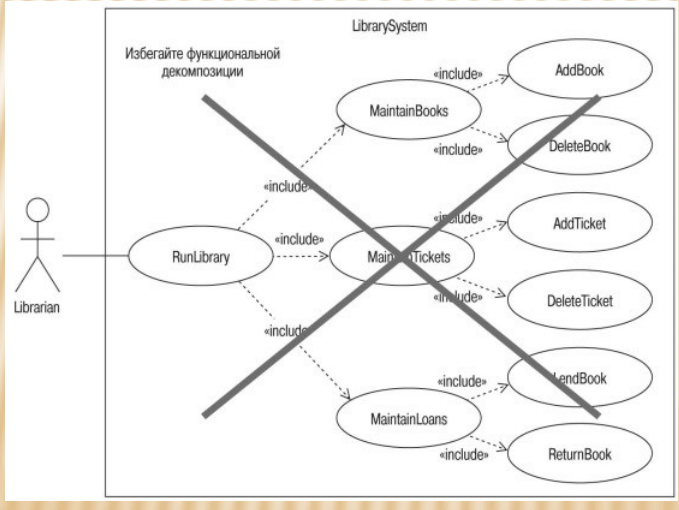

Рис.13. Избегайте функциональной декомпозиции [2]

АНАЛИЗ ТРЕБОВАНИЙ. ЛАБОРАТОРНАЯ № 4

Пример диаграммы вариантов использования

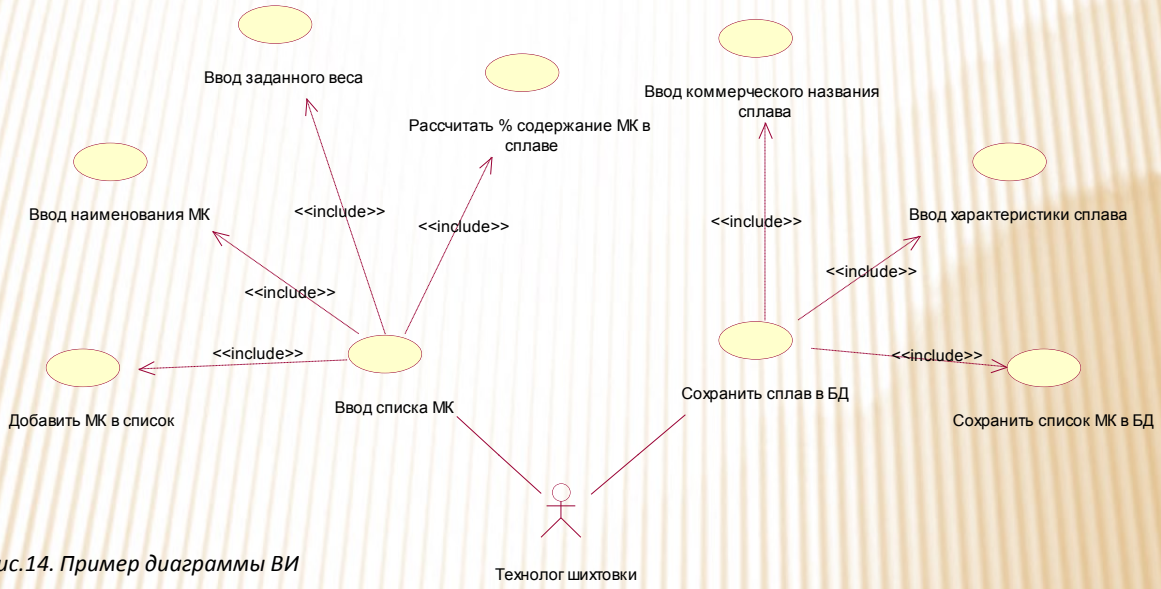

Рис.14. Пример диаграммы ВИ

Модель визуализации данных

Заказчик (пользователь) обычно хорошо представляет набор каких данных должен отображаться на мониторе, поэтому построение модели в которой, каждый вариант использования отражает одну единственную функцию: **показать [конкретный тип данных]**, не вызывает сложностей (рис. 15). Такие модели могут помочь при разработке и тестировании графического интерфейса пользователя (экранных форм), а также при тестировании полноты функционала системы.

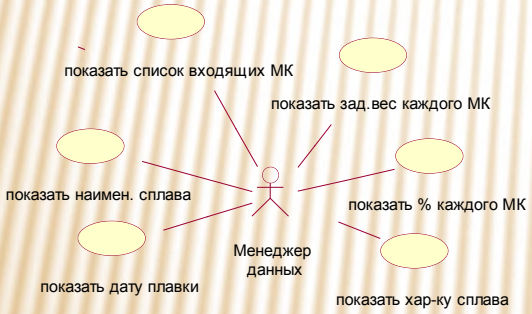

Рис.15. Модель визуализации данных

Однако, по определению (стр. XX) вариант использования это последовательность действий актёра и связанных с ними действий системы, направленные на достижение определённого, значимого для пользователя результата. Одиночную функцию **показать [конкретный тип данных]** можно лишь условно отождествить с вариантом использования. В классических подходах (например, в RUP и других) диаграммы, аналогичные рис. 15 отсутствуют. В данной работе построение модели визуализации данных предлагается опционально с целью облегчения последующей разработки прототипа экранных форм.

Задание для лабораторной работы

Используя трассировочную таблицу, модель бизнес-процессов и диаграмму бизнес-актёров и их обязанностей, выполненные в предыдущих лабораторных работах:

- определите и проанализируйте набор сервисов, предоставляемый программной системой, а также определите пользователей (актеров), которые будут их инициировать.
- распределите сервисы и актёров по папкам (модулям). Выявите общие сервисы (предоставляемые нескольким актёрам) и постройте для них модель в отдельной папке;
- конкретизируйте содержание базовых сервисов с помощью связанных с ними отношениями *include* и *extend* вариантов использования;
- (опционально) постройте модель визуализации данных для крановщика.

ОСНОВЫ СИСТЕМНОГО АНАЛИЗА. ЛАБОРАТОРНАЯ № 5

Темы: *Сценарии, их написание и предназначение.*
Прототип графического интерфейса, его связь со сценариями.
Моделирование потоков событий (activity diagram).

Сценарий — это текстовое описание потоков событий при выполнении конкретного варианта использования, выражающее некий аспект поведения системы.

Сценарии служат для перехода от вариантов использования к объектам программной системы. Анализируются имена-существительные в тексте сценария. Некоторые из них будут действующими лицами, другие — объектами, а третьи — атрибутами объекта.

Существует двухэтапный подход к написанию сценариев. На первом этапе сценарий пишут в произвольной форме с высоким уровнем абстракции. Целью такого сценария будет не выявление объектов, а поверхностное описание потока событий для общего понятия выполняемого функционала в данном варианте использования.

На втором этапе сценарии уточняют и подробно описывают в стандартной форме с указанием начальных и конечных условий, точек ветвления и привязывают к разработанному для этого сценария (или для группы сценариев) прототипу графического интерфейса пользователя GUI (Graphical User Interface).

Прототипы GUI строятся на основе Use-Case-моделей, они отображают функции бизнес-логики и, по возможности, содержат основные кнопки, поля и элементы меню.

Для подробного описания сценария в RUP имеется специальный шаблон. Каждый сценарий начинается главного раздела, далее описываются типовой и альтернативные потоки событий, для актеров – физических лиц должны быть предусмотрены ошибочные потоки событий.

Каждый поток событий заканчивается своим результатом. Типовой поток событий заканчивается желаемым для пользователя результатом.

Сценарии с прототипами экранных форм (особенно, с динамическими) являются первой действующей моделью разрабатываемой программы, которую можно представить для согласования Заказчику.

Пример сценария (главный раздел, типовой поток событий и прототип ЭФ) представлены на следующей странице.

ОСНОВЫ СИСТЕМНОГО АНАЛИЗА. ЛАБОРАТОРНАЯ № 5

Главный раздел

Наименование ВИ	Ввод списка МК
Актеры	Технолог шихтовки
Цель исполнения	Ввод списка МК, входящих в сплав и задание их пропорции
Краткое описание	Технолог шихтовки вводит последовательно наименование каждого МК и его заданный вес, затем помещает его в список МК, система по введенным заданным весам рассчитывает процентное содержание каждого МК в сплаве
Тип	Базовый
Начальные условия	На мониторе технолога шихтовки отображается окно ввода начальных данных

Типовой поток событий

Типовой поток событий сценария ВИ "Ввод списка МК"

Действия актеров	Отклик системы
1. Технолог шихтовки вводит в поле 1 экранной формы (ЭФ1, «Ввод данных») наименование МК;	
2. Технолог шихтовки вводит в поле 2 ЭФ1 ввода данных заданный вес данного МК;	
3. Технолог шихтовки нажимает клавишу 3 (Добавить МК в список) ЭФ ввода данных	
	4. ПС рассчитает % содержание данного МК и покажет в поле 6 списка МК;
	5. Наименование МК переместится в поле 4, заданный вес в поле 5 списка МК,
Примечание 1: Действия 1 – 6 повторяются по количеству МК	6. Поля 1 и 2 очистятся
Конечные условия:	На ЭФ1 заполнены поля списка МК 4, 5, 6. По каждому МК, предназначенному для шихтовки, показываются наименование, заданный вес и % содержание в сплаве

Рис. 16. Пример описания главного раздела и типового потока событий варианта использования

Прототип экранной формы

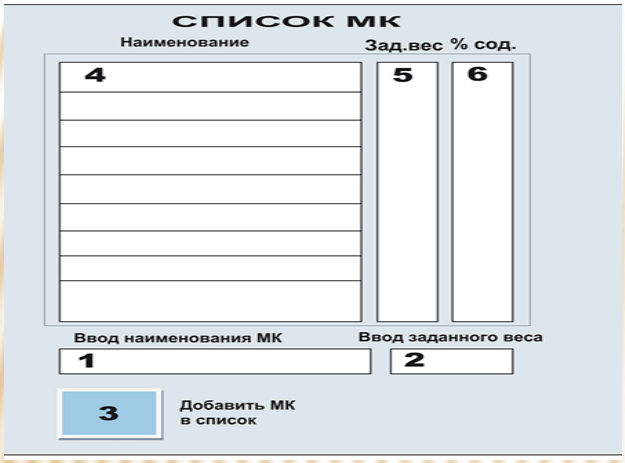

Рис.17. Пример прототипа (статического) ЭФ

Задание для лабораторной работы

Используя, созданную ранее модель вариантов использования:

- в любом графическом редакторе постройте прототип графического интерфейса пользователя, аналогичный рис. 17, для двух выбранных вариантов использования;
- используя стандартную форму рис.16 и макет графического интерфейса пользователя сделайте подробное описание всех потоков событий, ссылками указывая поля и клавиши на макете GUI.

ОСНОВЫ СИСТЕМНОГО АНАЛИЗА. ЛАБОРАТОРНАЯ № 6

Темы: *Классы анализа (entity, boundary, control);*
Диаграмма последовательности (sequence diagram);
Кооперативная диаграмма (collaboration diagram);
Диаграмма классов (class diagram).

В системном анализе все классы программной системы принято относить к трём видам (рис. 18), каждый из которых имеет определённое предназначение и обозначается своим стереотипом:

Рис. 18. Классы анализа

- **классы сущности** (*entity*) — описывают объекты программной системы, отвечающие за целостность и хранение данных (значений атрибутов). Объекты сущности не могут самостоятельно инициировать взаимодействия. Все присущие им операции, связаны с их атрибутами и относятся к следующим видам: ***модификатор*** (операция, изменяющая состояние объекта), ***селектор*** (операция, имеющая доступ к состоянию объекта, но не изменяющая его), ***итератор*** (операция, обеспечивающая доступ ко всем частям объекта в строго определенном порядке), ***конструктор*** (операция, создающая объект и/или инициализирующая его состояние) и ***деструктор*** (операция, стирающая состояние объекта и/или уничтожающая сам объект);
- **граничные классы** (*boundary*) — описывают объекты программной системы, которые являются интерфейсами связи с внешними пользователями (актёрами). В случае, если пользователь физическое лицо, то с помощью объектов-boundary моделируются экранные формы;
- **классы управления** (*control*) — описывают объекты программной системы, которые отвечают за выполнение методов (управляют взаимодействиями, реализуют математические методы, задают последовательности и так далее).

Диаграмма последовательности (*sequence diagram*) описывает временную последовательность обмена сообщениями между объектами программной системы в одном из потоков событий варианта использования.

Это объектная диаграмма, отражающая динамику взаимодействия объектов программной системы. Следует заметить, что в Rational Rose для того, чтобы назначить стереотип объекту, его необходимо ассоциировать с классом. Сообщения (Object Message) на этапе системного анализа могут быть написаны обычным текстом, а на этапе проектирования нужно ассоциировать с соответствующими операциями класса. Таким образом диаграмма последовательности отражает временную последовательность вызова методов объектов программной системы.

Пример диаграммы последовательности для типового потока событий варианта использования «Ввод списка металлических компонентов» представлена ниже (рис. 19).

Кооперативная диаграмма (*collaboration diagram*) показывает структуру взаимосвязей между объектами программной системы в одном из потоков событий варианта использования.

Пример диаграммы кооперации представлен ниже (рис. 20).

Диаграмма классов (*class diagram*) — это графическое представление статической модели, в которой отображены классы, их типы, содержимое и их отношения.

Диаграмма классов строится на основе анализа всех кооперативных диаграмм, поэтому в пример ниже (рис. 21) показывает лишь её фрагмент.

ОСНОВЫ СИСТЕМНОГО АНАЛИЗА. ЛАБОРАТОРНАЯ № 6

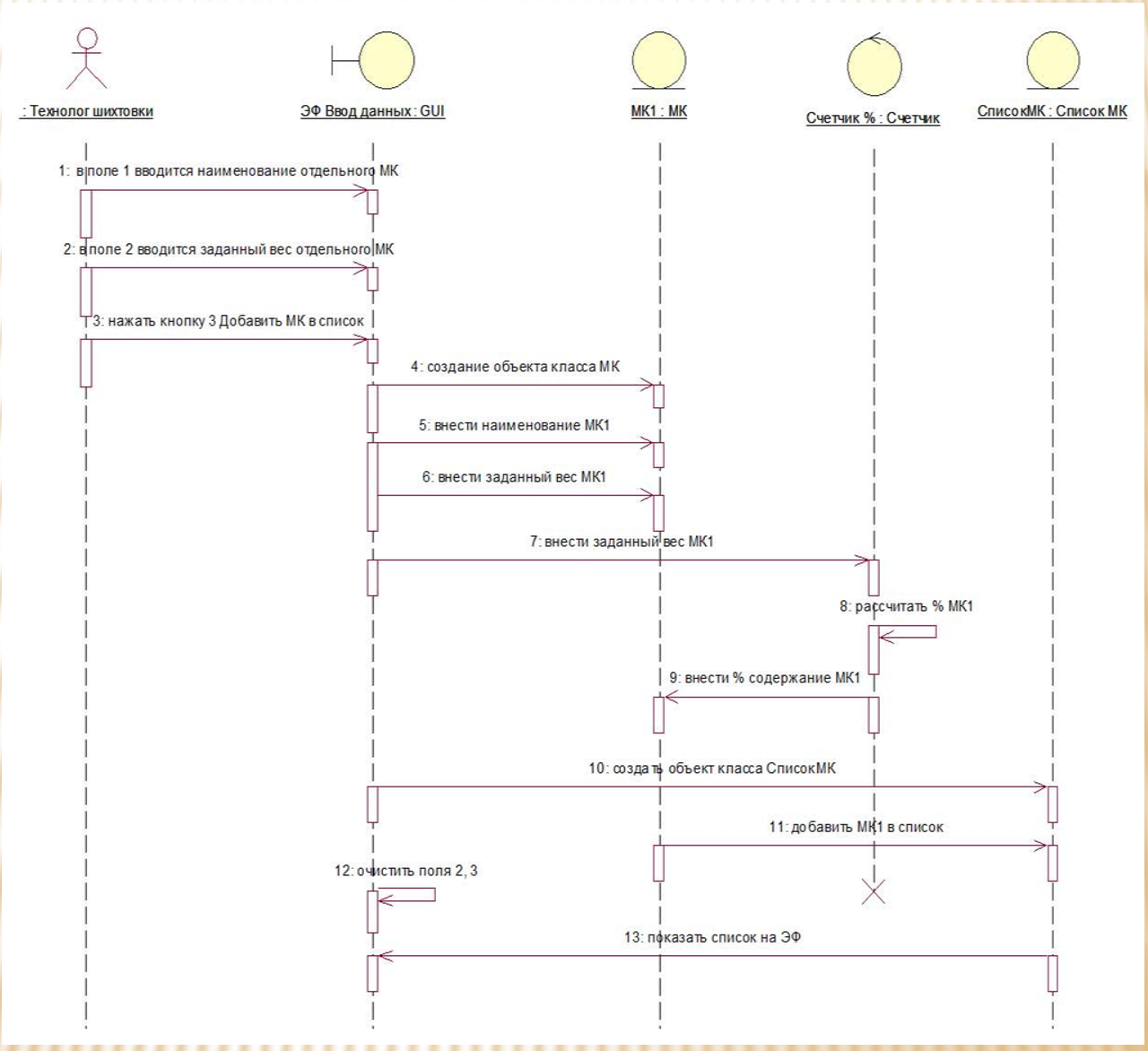

Рис. 19. Пример диаграммы последовательности (sequence diagram) для типового потока событий варианта использования «Ввод списка металлических компонентов»

ОСНОВЫ СИСТЕМНОГО АНАЛИЗА. ЛАБОРАТОРНАЯ № 6

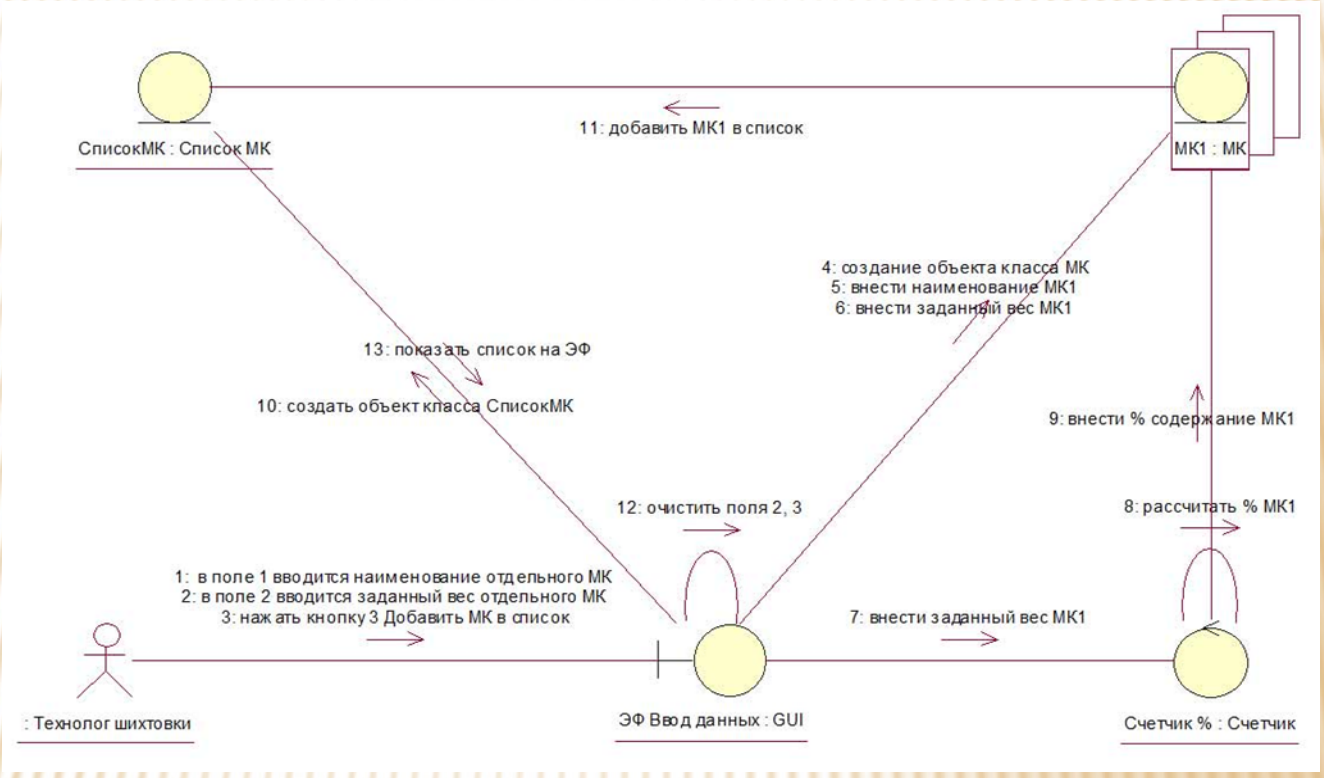

Рис. 20. Пример кооперативной диаграммы (collaboration diagram) для типового потока событий варианта использования «Ввод списка металлических компонентов»

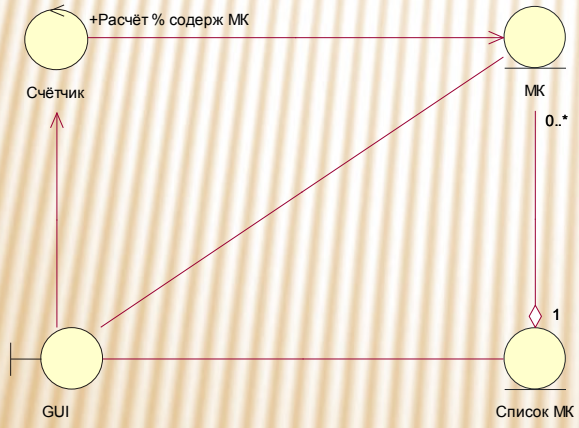

Рис. 21. Фрагмент диаграммы классов

Задание для лабораторной работы

Используя подготовленные сценарии:

- выявить из текста сценария классы анализа и их объекты;
- построить диаграммы последовательности, для объектов, описываемых классами анализа;
- нажав клавишу F5, преобразовать диаграммы последовательности в кооперативные диаграммы;
- построить диаграмму классов и их отношений (association, generalization, aggregation).

ЗАКЛЮЧЕНИЕ

В представленных выше шести лабораторных работах показан ход и артефакты анализа программной системы, выполненные в рамках стандартного унифицированного процесса разработки, включающего:

- анализ предметной области;
- анализ требований;
- системный анализ.

Строгое следование стандартному процессу не является самоцелью, важнее как можно быстрее получить работающую программу, поэтому процесс можно изменять и минимизировать количество создаваемых артефактов, оценивая при этом возрастающие риски.

ЛИТЕРАТУРА

1. Карл И. Вигерс Разработка требований к программному обеспечению «Русская Редакция». М., 2004.
2. Г. Буч и др. «Объектно-ориентированный анализ и проектирование с примерами приложений», Москва, «Вильямс», 2008 г., 3-е издание.
3. Дж. Арлоу, А. Нейштадт, «UML-2и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование». Санкт-Петербург, «Символ-Плюс», 2007 г., 2-е издание.
4. Дж. Рамбо, М. Блаха «UML-2 Объектно-ориентированное моделирование и разработка» Санкт-Петербург, «Питер», 2007 г., 2-е издание.
5. Уэнди Боггс, Майкл Боггс «UML и Rational Rose 2002» /Пер. с англ. — М. «Лори», 2004.
6. Основы программной инженерии (по SWEBOK, 2004 г.), перевод С. Орлик, 2004-2010 г.

ПРОДОЛЖЕНИЕ

АНАЛИЗ ТРЕБОВАНИЙ НА ОСНОВЕ UML-МОДЕЛЕЙ И ДИНАМИЧЕСКИХ HTML-ПРОТОТИПОВ

(тренинг на www.webmax.by)