ЛАБОРАТОРНЫЕ РАБОТЫ. ЧАСТЬ 1

по дисциплине ТехнологиЯ разработки программного обеспечения

Е.Н. Зернова, Л.Д. Челищева

**СОДЕРЖАНИЕ**

[Лабораторная работа №1 Анализ проблемы. Постановка задачи 3](#_Toc210560940)

[Краткие теоретические сведения 3](#_Toc210560941)

[Задание и порядок проведения работы 9](#_Toc210560942)

[Лабораторная работа №2 Разработка пользовательских историй при проектировании ИС 11](#_Toc210560943)

[Краткие теоретические сведения 11](#_Toc210560944)

[Задание и порядок проведения работы 13](#_Toc210560945)

[Лабораторная работа №3 Функциональные и нефункциональные требования 14](#_Toc210560946)

[Краткие теоретические сведения 14](#_Toc210560947)

[Задание и порядок проведения работы 21](#_Toc210560948)

[Лабораторная работа №4 Моделирование бизнес-процессов 22](#_Toc210560949)

[Краткие теоретические сведения 22](#_Toc210560950)

[Задание и порядок проведения работы 23](#_Toc210560951)

[Лабораторная работа №5 Построение диаграммы вариантов использования 23](#_Toc210560952)

[Краткие теоретические сведения 23](#_Toc210560953)

[Задание и порядок проведения работы 26](#_Toc210560954)

[Лабораторная работа №6 Построение диаграммы последовательности 26](#_Toc210560955)

[Краткие теоретические сведения 27](#_Toc210560956)

[Задание и порядок проведения работы 34](#_Toc210560957)

[Лабораторная работа №7 Построение диаграммы активности 35](#_Toc210560958)

[Краткие теоретические сведения 35](#_Toc210560959)

[Задание и порядок проведения работы 40](#_Toc210560960)

[Лабораторная работа №8 Построение диаграммы состояний 41](#_Toc210560961)

[Краткие теоретические сведения 41](#_Toc210560962)

[Задание и порядок проведения работы 42](#_Toc210560963)

# Лабораторная работа №1 Анализ проблемы. Постановка задачи

Цель работы: сформировать навыки:

* работы с реальными заказчиками программных систем;
* идентификации заинтересованных лиц и интервью с ними;
* анализа полученного материала;
* формулирования проблемы, ее актуальности и потребностей заинтересованных лиц.

## Краткие теоретические сведения

На этапе анализа проблемы проводится анализ предметной области, для которой разрабатывается ПО. Цели этапа:

1. определение границ, или контура, системы;
2. описание объектов автоматизации и/или формализации знаний об этих объектах;
3. выявление или определение потребностей заказчика ПО.

Анализ предметной области можно проводить, например, основываясь

на теории системного анализа и использовать предложенные в ней методы.

Исходными данными для этапа системного анализа являются:

1. регламенты работы отделов и должностные инструкции сотрудников этих отделов;
2. анкеты опроса заинтересованных лиц;
3. записи интервью с заинтересованными лицами;
4. другие документы, имеющие отношение к исследуемому объекту.

Выходными данными, или результатом, этапа системного анализа являются:

1. перечень заинтересованных лиц;
2. список потребностей заинтересованных лиц в разрабатываемом ПО;
3. описание объектов автоматизации;
4. модель объектов автоматизации или предметной области.

Описание примера

Здесь формулируется задача, решением которой является разработка программного обеспечения.

Итак, в результате вступления России в Болонский процесс в РФ была инициирована реформа высшего профессионального образования, в соответствии с которой Министерством образования и науки РФ была разработана программа перевода традиционной системы оценки успеваемости студентов в систему зачетных единиц (кредитов). Это объясняется необходимостью унификации систем высшего образования с целью создания единого образовательного пространства в тех странах, которые уже вступили в Болонский процесс. В рамках этой программы все вузы страны должны к установленной дате перейти на новую систему. Красноярский государственный политехнический университет (КГПУ) начал решать поставленную перед ним задачу поэтапно. Одной из задач перехода на новую систему в КГПУ являлась автоматизация учета текущей успеваемости и промежуточных аттестаций студентов в целях унификации этого процесса на всех кафедрах и факультетах вуза, реализации возможности автоматизированного формирования отчетов, публикации на сайте вуза рейтингов успеваемости студентов.

Составление списка заинтересованных лиц

Заинтересованные лица – это все те, кто имеет прямое или косвенное отношение к процессу, автоматизация которого производится.

Для выявления заинтересованных лиц необходимо ответить на следующие вопросы:

* кто является пользователем системы?
* кто является заказчиком (покупателем) системы?
* на кого еще окажут влияние результаты работы системы?
* кто будет оценивать и принимать систему, когда она будет представлена и развернута?
* существуют ли другие внутренние или внешние пользователи системы, чьи потребности необходимо учесть?
* кто будет заниматься сопровождением новой системы?
* не забыли ли мы кого-нибудь?

В нашем примере определим будущих пользователей системы – это преподаватели, секретари кафедр и деканатов, заведующие кафедрами, системный администратор и сотрудники Учебного управления. Заказчиком нашей системы является вуз в лице первого проректора.

Теперь попытаемся выяснить, на кого еще будут оказывать влияние результаты работы нашей системы. Во-первых, на студентов, ведь рейтинги успеваемости всех студентов будут доступны на сайте вуза. Во-вторых, по этой же причине, на родителей. В-третьих, на деканов факультетов и заместителей деканов по учебной работе, поскольку любые изменения в учебном процессе касаются их профессиональной деятельности.

Итак, рассмотрев первые три вопроса, мы практически охватили всех заинтересованных лиц. Поскольку сопровождать систему будет разработчик этой системы, то в список заинтересованных лиц мы его не включаем. Таким образом, получаем следующий список заинтересованных лиц для нашей системы:

* преподаватели;
* секретари кафедр;
* секретари деканатов;
* заведующие кафедрами;
* Учебное управление (в лице начальника);
* первый проректор;
* системный администратор (тот, кто будет администрировать нашу систему);
* студенты;
* родители студентов;
* заместители деканов по учебной работе;
* деканы факультетов.

Анкетирование и проведение интервью

Для выявления потребностей заказчика и описания объектов автоматизации можно проводить как анкетирование, так и интервью. Но наибольший эффект возможен только при проведении и того и другого.

Примеры анкеты и перечня вопросов для интервью приведены ниже.

Анкета для опроса заинтересованных лиц

1. Имя.

2. Наименование организации.

3. Наименование структурного подразделения.

4. Должность.

5. Кому Вы непосредственно подчиняетесь?

6. Каковы Ваши основные обязанности?

7. Что Вы в основном производите?

8. Для кого?

9. Какие документы или какую информацию можно считать входящими, или необходимыми, для Вашей деятельности?

10. Какие документы или какую информацию можно считать исходящими, или результатом Вашей деятельности?

11. Как измеряется успех Вашей деятельности?

12. Какие проблемы влияют на успешность Вашей деятельности?

13. Какие тенденции, если такие существуют, делают Вашу работу

проще или сложнее?

14. Какой интерес или какие потребности у Вас есть относительно будущего решения (разрабатываемого ПО)?

Перечень вопросов для интервью

1. Оценка проблемы
   1. Для каких проблем (прикладного типа) Вы ощущаете нехватку хороших решений? Назовите их. (Не забывайте спрашивать: «А еще?»)
2. По каждой проблеме выясняйте следующее:
   1. почему существует эта проблема?
   2. как она решается в настоящее время?
   3. как заказчик (пользователь) хотел бы ее решать?
3. Понимание пользовательской среды
   1. Каковы Ваши навыки в компьютерной области?
   2. С какими типами приложений Вы имеете опыт работы?
   3. Какая платформа используется?
   4. Каковы Ваши планы относительно будущих платформ?
   5. Используется ли ПО, которое имеет отношение к данной проблеме? (Если да, то пусть о нем немного расскажут.)
   6. Каковы Ваши ожидания относительно практичности продукта?
   7. В каком виде должна быть представлена справочная информация для пользователя (в интерактивном или печатном)?
4. Резюме (перечисляются основные пункты, чтобы проверить, все ли правильно вы поняли)
   1. Итак, Вы сказали мне… (перечислите описанные заказчиком проблемы своими словами)
   2. Адекватно ли этот список представляет проблемы, которые имеются при существующем решении?
   3. Какие еще проблемы Вы испытываете?

Заключение аналитика.

После интервью, пока его данные еще свежи в вашей памяти, зафиксируйте не менее трех потребностей или проблем с наивысшими приоритетами, выявленных вами в беседе с данным заказчиком (пользователем).

После проведения анкетирования и интервьюирования необходимо обработать собранную информацию. На основе этих данных нужно сформулировать перечень потребностей заказчиков, построить модель предметной области и описать объект/объекты автоматизации. Все эти результаты в дальнейшем будут использованы при написании технического задания (ТЗ) на разрабатываемую систему.

Интервью с начальником учебного управления

|  |  |
| --- | --- |
| Вопрос | Ответ |
| 1. Имя | Иванов Петр Владимирович |
| 2. Наименование структурного подразделения | Учебное управление |
| 3. Должность | Начальник Учебного управления |
| 4. Кому Вы непосредственно подчиняетесь? | Первому проректору |
| 5. Каковы Ваши основные обязанности? | Обеспечение общего управления учебной деятельностью. Обеспечение качества и своевременного выполнения возложенных на Учебное управление задач и функций. Обеспечение выполнения мероприятий по защите конфиденциальной и информационной безопасности в подразделениях управления |
| 6. Что Вы в основном производите? | Функции, выполняемые Учебным управлением:  1. Создание единого нормативно-справочного поля учебного процесса.  2. Организационное сопровождение учебного процесса.  3. Планирование ресурсов учебного процесса (объемов работ кафедр, площадей, численности контингента, нагрузки на условиях почасовой оплаты, бланочной документации и др.).  4. Контроль за ходом организации и анализ показателей учебного процесса (как внешних, так и внутренних).  5. Отчетность на различные уровни управления.  6. Анализ выполнения учебной нагрузки, трудоемкости отдельных параметров учебного процесса, результатов сессии, результатов государственных аттестационных комиссий. Разработка графика учебного процесса.  7. Составление расписания учебных занятий, экзаменов, зачетов и графика защит.  8. Контроль занятости аудиторного фонда, составления семестровых планов, проведения занятий, заседаний государственных аттестационных комиссий, хода сессий, делопроизводства факультетов, оформления приложений к дипломам.  9. Учет движения студентов по всем формам обучения (очное, очно-заочное и заочное, экстернат, второе высшее образование, обучение по сокращенным программам) |
| 7. Какие документы или какую информацию можно считать входящими, или необходимыми, для Вашей деятельности? | 1. Приказы и инструктивные письма Минобразования России по учебно-методическим вопросам.  2. Положение об Учебном управлении.  3. Должностные инструкции сотрудников.  4. Приказы ректора по контингенту студентов (первые экземпляры).  5. Годовые планы приема абитуриентов.  6. Планы работы факультетов.  7. Отчеты председателей государственных аттестационных комиссий по всем специальностям и направлениям |
| 8. Какие документы или какую информацию можно считать исходящими, или результатом Вашей деятельности? | 1. Семестровые планы занятий.  2. Расписание учебных занятий.  3. Расписание экзаменов.  4. Отчеты вуза по учебно-методической работе за учебный год.  5. Сводные статистические отчеты вуза о движении контингента студентов на начало и конец учебного года |
| 9. Как измеряется успех Вашей деятельности? | В настоящее время отсутствуют количественные показатели оценки деятельности управления |
| 10. Какие проблемы влияют на успешность Вашей деятельности? | Проблемы, связанные с организацией учебного процесса. В настоящее время основной проблемой является перевод учебного процесса в систему зачетных единиц |
| 11. Какой интерес или какие потребности у Вас есть относительно будущего решения (разрабатываемого ПО)? | Разрабатываемая система должна быть максимально эргономичной, работать стабильно (без сбоев); отклик системы не должен вызывать у пользователей раздражения; реализуемая функциональность должна полностью удовлетворить потребности пользователя |

В качестве примера проведем анкетирование и интервью с начальником Учебного управления. Для этого воспользуемся одним перечнем вопросов, совместив две эти процедуры воедино. По сути, мы проведем только интервью. Пример результата интервью (вопросы и ответы) приведен в таблице.

Список потребностей заинтересованных лиц

В результате анкетирования и интервьюирования всех заинтересованных лиц были сформулированы потребности заказчика относительно разрабатываемого ПО. Далее необходимо провести аналогию между выявленными потребностями и структурой и требованиями ТЗ в соответствии с ГОСТ 34.602–89. Таким образом, потребности заказчика в ТЗ могут быть описаны в разделе «Назначение и цели создания системы».

В нашем примере были выявлены следующие потребности:

1) унифицировать процесс оценивания знаний в системе кредитов на всех кафедрах и факультетах вуза;

2) минимизировать субъективность при оценивании студентов в промежуточных аттестациях;

3) реализовать возможность автоматического формирования рейтингов студентов по разным параметрам в системе кредитов;

4) реализовать возможность формирования единой отчетности на кафедрах и факультетах.

## Задание и порядок проведения работы

1. Составить перечень заинтересованных лиц.
2. Провести интервью и/или анкетирование с каждым заинтересованным лицом.
3. Проанализировать полученную информацию и сформулировать актуальность проблемы и потребности заинтересованных лиц.

**Подсказки к работе:**

* Вы разрабатываете ПО для некоторой воображаемой компании-заказчика, а не для вывода на рынок в целом.
* Вопросы для каждого заинтересованного лица составляются индивидуально!
* Спрашивайте только то, что может помочь вам при дальнейшей разработке.
* Вы должны узнать весь ожидаемый функционал в ходе опроса, причем достаточно детально, чтобы можно было составить ТЗ и начать распределять задачи (например, ответ «хочу отчеты» ничего не означает. Какие отчеты?).
* Фактически у любого заинтересованного лица вы должны выяснить 3 вещи:
  + Что он делает на своей должности?
  + Чего ему не хватает/что не удобно/что выполняется неэффективно и др. проблемы?
  + Что из вышеперечисленного он бы хотел автоматизировать? (что не хочет – то не автоматизируем! Так что только выяснить задачи/проблемы недостаточно)
* У заказчика нужно также узнать эти 3 пункта, но про компанию в целом – что делает, что не так, зачем вообще ИС?
* Заказчика нужно опросить первым, в интервью. Он может вообще при этом не пользоваться системой, что и ответит на вопросы про личные потребности (их ему тоже нужно задать).
* Остальные вопросы в интервью: уточняющие/наводящие.
* Вы как разработчик не должны быть знакомы с предметной областью. Для знакомства и проводится опрос. А значит вопросы должны быть либо максимально общие, без предположений о рабочих задачах и потребностях опрашиваемого, либо ссылаться на ответы на предшествующие вопросы (в т.ч. из других интервью).
* Отвечающий, за исключением сис.админа – не IT-специалист, а зачастую даже не инженер. Он не может отвечать в профессиональных терминах или понимать их без разъяснений.
* Вопросы для анкет должны либо быть максимально общие, либо быть составлены на основе обсуждения их с кем-то из заинтересованных лиц в интервью ранее.
* Ответы на анкеты можно приводить как от лица одного отвечающего, так и как будто в виде результата анализа сборной статистики.
* В ходе обсуждения предметной области обязательно кем-то да должны быть упомянуты все основные сущности будущей системы (задайте себе вопросы по типу: «Кто добавляет новых сотрудников и выдает им роли? Кто редактирует список товаров? Кто может удалить неиспользуемую категорию или закрытый филиал?»).
* В ходе обсуждения отвечающий может использовать качественные прилагательные, однако в список потребностей они не входят, т.к. не несут однозначного смысла. Пример таких слов: красивый, медленный, быстрый, удобный и т.п.
* Актуальность формулируется следующим образом: «Актуальность данной работы состоит в том, что на данный момент в компании … (все очень плохо, все делается вручную/на бумаге/в эксель). Следовательно, введение нового ПО позволит/улучшит … (сделать хорошо)».
* Список потребностей должен быть максимально подробным, фактически резюме всех интервью. Из этого списка напрямую переформулируются требования в ТЗ.

**Контрольные вопросы**

1. Что является исходными данными для анализа проблемы (предметной области)?
2. Что является результатом этапа системного анализа предметной области?
3. Как определить заинтересованных лиц?
4. Какой метод сбора информации наиболее эффективен?
5. Для чего проводятся интервьюирование и анкетирование?

# Лабораторная работа №2 Разработка пользовательских историй при проектировании ИС

Цель работы: освоение технологии разработки и управления требованиями

## Краткие теоретические сведения

В практике подготовки требований, которые в дальнейшем будут составлять каркас разрабатываемого программного продукта, принято использовать один из стандартизированных подходов для их описания – User Stories ("пользовательские истории").

Пользовательские истории формулируются как одно или более предложений на "повседневном" языке пользователя. Они получаются относительно небольшие по объему, что удобно как для составления, так и для их обсуждения, приоритизации, планирования, оценки и дальнейшей работы с ними. Пользовательские истории получаются в виде алгоритма действий пользователя с реализуемым программным продуктом. Адекватная оценка трудоемкости историй позволяет планировать сроки ее реализации, тем самым управляя содержимым спринтов. Разработать оптимальную пользовательскую историю не так просто, нужен определенный навык, который позволит создать качественное описание требований пользователей к реализуемому программному продукту, но в награду за это получаются следующие преимущества:

* Краткость. Пользовательская история описывает небольшую часть бизнес-ценности, которую возможно реализовать за период спринта.
* "Незатратность" создания и сопровождения. За счет своей "компактности" пользовательские требования достаточно просто создать и сопровождать их изменения на всем протяжении жизненного цикла продукта.
* Вовлечение ключевых пользователей в процесс создания продукта. За счет своей доступности бизнес-требования смогут стать реальным "мостиком" между пользователями и командой разработчиков. Это позволит более адекватно управлять ожиданиями пользователей и вовлечь их на нужную степень погружения в процесс разработки продукта.
* Облегчают оценку заданий. Формат пользовательских историй способствует более точной оценке необходимых системных разработок/доработок.

Чем более компактный объем имеет пользовательская история, тем проще выполнять ее оценку. Это приводит к более верным оценкам сроков реализации программного продукта и к более понятному планированию выполняемых работ. При этом важно знать меру в уменьшении и дроблении Use cases. Если сделать их слишком много, то у вас получится огромный список задач в бэклоге продукта, и процесс управления ими и фиксации на карте историй усложнится.

Для пользователей требования, выраженные в виде пользовательских историй, являются основным инструментом влияния на разрабатываемый программный продукт. Пользовательские истории определяют формат, в котором у пользователей есть возможность отразить все те важные факторы, которые, по их мнению, должны быть учтены в процессе автоматизации.

Важно отдавать себе отчет в том, что пользовательские истории не обеспечивают полноту всех функциональных требований и имеют ряд недостатков:

* Не масштабируются для больших программных продуктов. Пользовательские истории хорошо себя зарекомендовали, когда речь идет о создании небольших или средних по объемам и сложности программным продуктам. За счет того, что этот вид работы с требованиями ориентирован на непосредственную работу с пользователями и поддерживается "бизнес-лексиконом", он неприменим в работе над крупными информационными системами, когда на первый план выходит организационная структура проекта/процесса, и важны формальные признаки сдачи/приемки работ.
* Требовательны к квалификации разработчиков. Разработчики, работающие с пользовательскими историями, должны обладать высокой квалификацией и неплохими коммуникативными навыками, которые позволят им получить необходимые уточнения уже изложенных требований. Как правило, таких разработчиков немного. И это еще один неоспоримый недостаток.
* Не являются средством документирования. Пользовательские истории – это небольшое и удобное представление информации. Они сформулированы на ежедневном языке пользователя и содержат небольшие детали, оставаясь открытыми для интерпретации. Они помогают понимать, что должна делать система, но при этом пользовательских требований недостаточно, чтобы понять, как будет организована логика системы. Пользовательские требования являются необходимой "верхушкой" для понимания назначения информационной системы, но для реализации системы разработчику приходится додумывать множество значимых деталей.

Инженерия требований – важный этап в создании каждого программного продукта. Функциональные обязанности по извлечению, разработке и управлению требованиями ложатся на всех членов команды, именно поэтому в процессе работы с ними задействованы такие артефакты, как пользовательские истории, назначение которых – облегчить получение информации, необходимой для качественной реализации программного продукта.

Формат пользовательской истории как правило следующий:

Я как тип пользователя, хочу действие, для того чтобы цель.

Пример:

Я как покупатель, хочу сформировать отчет по покупкам, для того чтобы проверить свои покупки.

Придерживаться такой структуры необязательно, но она помогает определить критерии готовности работы. История выполнена, когда упомянутый тип пользователя получает требуемую ценность. В идеале, команды формулируют свою собственную структуру и придерживаются ее.

## Задание и порядок проведения работы

Ознакомиться с теоретическим материалом, разработать не менее **16** пользовательских историй от различных типов пользователей.

# Лабораторная работа №3 Функциональные и нефункциональные требования

## Краткие теоретические сведения

Требования к программной системе часто классифицируются как функциональные, нефункциональные и требования предметной области.

**Функциональные требования** задают “что” система должна делать; **нефункциональные** – с соблюдением “каких условий” (например, скорость отклика при выполнении заданной операции); часто функциональные требования представляют в виде сценариев (вариантов использования) Use Сase.

1. Функциональные требования. Это перечень сервисов, которые должна выполнять система, причем должно быть указано, как система реагирует на те или иные вход­ные данные, как она ведет себя в определенных ситуациях и т.д. В некоторых слу­чаях указывается, что система не должна делать.
2. Нефункциональные требования. Описывают характеристики системы и ее окружения, а не поведение системы. Здесь также может быть приведен перечень ограничений, накладываемых на действия и функции, выполняемые системой. Они включают временные ограничения, ограничения на процесс разработки системы, стандарты и тд.
3. Требования предметной области. Характеризуют ту предметную область, где будет эксплуатироваться система. Эти требования могут быть функциональными и не­функциональными.

В действительности четкой границы между этими типами требований не существует. Например, пользовательские требования, касающиеся безопасности системы, можно отнести к нефункциональным. Однако при более детальном рассмотрении такое требование можно отнести к функциональным, поскольку оно порождает необходимость включения в систему средства авторизации пользователя. Поэтому, рассматривая далее эти виды требований, мы должны всегда помнить, что данная классификация в значительной степени искусственна.

Классический пример (см. рисунок 4.3) высокоуровневого структурирования групп требований как требований к продукту описан в работах одного из классиков дисциплины управления требованиями – Карла Вигерса.

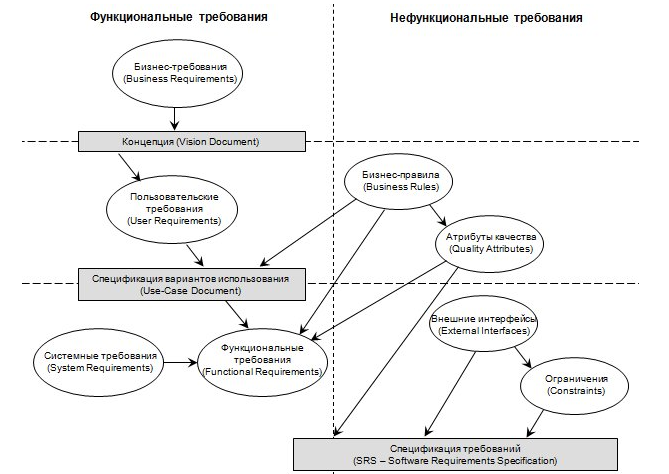


Рисунок 4.3. Уровни требований по Вигерсу

* Группа функциональных требований
  + Бизнес-требования (Business Requirements) – определяют высокоуровневые цели организации или клиента (потребителя) – заказчика разрабатываемого программного обеспечения.
  + Пользовательские требования (User Requirements) – описывают цели/задачи пользователей системы, которые должны достигаться/выполняться пользователями при помощи создаваемой программной системы. Эти требования часто представляют в виде вариантов использования (Use Cases).
  + Функциональные требования (Functional Requirements) – определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований.
* Группа нефункциональных требований (Non-Functional Requirements)
  + Бизнес-правила (Business Rules) – включают или связаны с корпоративными регламентами, политиками, стандартами, законодательными актами, внутрикорпоративными инициативами (например, стремление достичь зрелости процессов по CMMI 4-го уровня), учетными практиками, алгоритмами вычислений и т.д. На самом деле, достаточно часто можно видеть недостаточное внимание такого рода требованиям со стороны сотрудников ИТ-департаментов и, в частности, технических специалистов, вовлеченных в проект. Business Rules Group дает понимание бизнес-правила, как “положения, которые определяют или ограничивают некоторые аспекты бизнеса. Они подразумевают организацию структуры бизнеса, контролируют или влияют на поведение бизнеса”. Бизнес-правила часто определяют распределение ответственности в системе, отвечая на вопрос “кто будет осуществлять конкретный вариант, сценарий использования” или диктуют появление некоторых функциональных требований. В контексте дисциплины управления проектами (уже вне проекта разработки программного обеспечения, но выполнения бизнес-проектов и бизнес-процессов) такие правила обладают высокой значимостью и, именно они, часто определяют ограничения бизнес-проектов, для автоматизации которых создается соответствующее программное обеспечение.
  + Внешние интерфейсы (External Interfaces) – часто подменяются “пользовательским интерфейсом”. На самом деле вопросы организации пользовательского интерфейса безусловно важны в данной категории требований, однако, конкретизация аспектов взаимодействия с другими системами, операционной средой (например, запись в журнал событий операционной системы), возможностями мониторинга при эксплуатации – все это не столько функциональные требования (к которым ошибочно приписывают иногда такие характеристики), сколько вопросы интерфейсов, так как функциональные требования связаны непосредственно с функциональностью системы, направленной на решение бизнес-потребностей.
  + Атрибуты качества (Quality Attributes) – описывают дополнительные характеристики продукта в различных “измерениях”, важных для пользователей и/или разработчиков. Атрибуты касаются вопросов портируемости, интероперабельности (прозрачности взаимодействия с другими системами), целостности, устойчивости и т.п.
  + Ограничения (Constraints) – формулировки условий, модифицирующих требования или наборы требований, сужая выбор возможных решений по их реализации. В частности, к ним могут относиться параметры производительности, влияющие на выбор платформы реализации и/или развертывания (протоколы, серверы приложений, баз данных, ...), которые, в свою очередь, могут относиться, например, к внешним интерфейсам.
* Системные требования (System Requirements) – иногда классифицируются как составная часть группы функциональных требований (не путайте с как таковыми “функциональными требованиями”). Описывают высокоуровневые требования к программному обеспечению, содержащему несколько или много взаимосвязанных подсистем и приложений. При этом, система может быть как целиком программной, так и состоять из программной и аппаратной частей. В общем случае, частью системы может быть персонал, выполняющий определенные функции системы, например, авторизация выполнения определенных операций с использованием программно-аппаратных подсистем.

**Функциональные требования**

Эти требования описывают поведение системы и сервисы (функции), которые она выполняет, и зависят от типа разрабатываемой системы и от потребностей пользовате­лей. Если функциональные требования оформлены как пользовательские, они, как прави­ло, описывают системы в обобщенном виде. В противоположность этому функциональ­ные требования, оформленные как системные, описывают систему максимально подроб­но, включая ее входные и выходные данные, исключения и т.д.

Функциональные требования для программных систем могут быть описаны разными способами. Рассмотрим для примера функциональные требования к библиотечной системе университета, предназначенной для заказа книг и документов из других библиотек.

1. Пользователь должен иметь возможность проводить поиск необходимых ему книг и документов или по всему множеству доступных каталожных баз данных или по определенному их подмножеству.
2. Система должна предоставлять пользователю подходящее средство просмотра библиотечных документов.
3. Каждый заказ должен быть снабжен уникальным идентификатором (ORDERJD), который копируется в формуляр пользователя для постоянного хранения.

Эти функциональные пользовательские требования определяют свойства, которыми должна обладать система. Они взяты из документа, содержащего пользовательские требования, и показывают, что функциональные требования могут быть описаны с разным уровнем детализации (сравните первое и третье требования).

Многие проблемы, возникающие при разработке систем, связаны с неточностью и "размытостью" спецификации требований. Естественно, разработчики интерпретируют требования, допускающие двоякое толкование, так, чтобы систему было проще реализо­вать. Но это толкование может не совпадать с ожиданиями заказчика. Такая ситуация приводит к разработке новых требований и внесению изменений в систему. Это, в свою очередь, ведет к задержке сдачи готовой системы и ее удорожанию.

Рассмотрим второе требование к библиотечной системе из приведенного выше списка и обратим внимание на выражение "подходящее средство просмотра документов". Библио­течная система может предоставлять документы в широком спектре форматов. В требова­нии подразумевается, что система должна предоставить средства для просмотра документов в любом формате. Но поскольку это условие четко не выписано, разработчики в случае де­фицита времени могут использовать простое средство для просмотра текстовых документов и настаивать на том, что именно такое решение следует из данного требования.

В принципе спецификация функциональных требований должна быть комплексной и непротиворечивой. Комплексность подразумевает описание (определение) всех систем­ных сервисов. Непротиворечивость означает отсутствие несовместимых и взаимоисклю­чающих определений сервисов. На практике для больших и сложных систем крайне труд­но разработать комплексную и непротиворечивую спецификацию функциональных тре­бований. Причина кроется частично в сложности самой разрабатываемой системы, а частично – в несогласованных опорных точках зрения на то, что должна де­лать система. Эта несогласованность может не проявиться на этапе первоначального формулирования требований – для ее выявления необходим более глубокий анализ спе­цификации. Когда несогласованность системных функций проявится на каком-либо этапе жизненного цикла программы, в системную спецификацию придется внести соответст­вующие изменения.

**Нефункциональные требования**

Как следует из названия, нефункциональные требования не связаны непосредственно с функциями, выполняемыми системой. Они связаны с такими интеграционными свойствами системы, как надежность, время ответа или размер системы. Кроме того, нефункциональные требования могут определять ограничения на систему, например на пропускную способность устройств ввода-вывода, или форматы данных, используемых в системном интерфейсе.

Многие нефункциональные требования относятся к системе в целом, а не к отдельным ее средствам. Это означает, что они более значимы и критичны, чем отдельные функцио­нальные требования. Ошибка, допущенная в функциональном требовании, может снизить качество системы, ошибка в нефункциональных требованиях может сделать систему не­работоспособной.

Вместе с тем нефункциональные требования могут относиться не только к самой программной системе: одни могут относиться к технологическому процессу создания ПО, другие – содержать перечень стандартов качества, накладываемых на процесс разработки. Кроме того, в спецификации нефункциональных требований может быть указано, что проектирование системы должно выполняться только определенными CASE-средствами, и приведено описание процесса проектирования, которому необходимо следовать.

Нефункциональные требования отображают пользовательские потребности; при этом они основываются на бюджетных ограничениях, учитывают организационные возможности компании-разработчика и возможность взаимодействия разрабатываемой системы с другими программными и вычислительными системами, а также такие внешние факторы, как правила техники безопасности, законодательство о защите интеллектуальной собственности и т.п. На рис. 4.4 показана классификация нефункциональных требований.

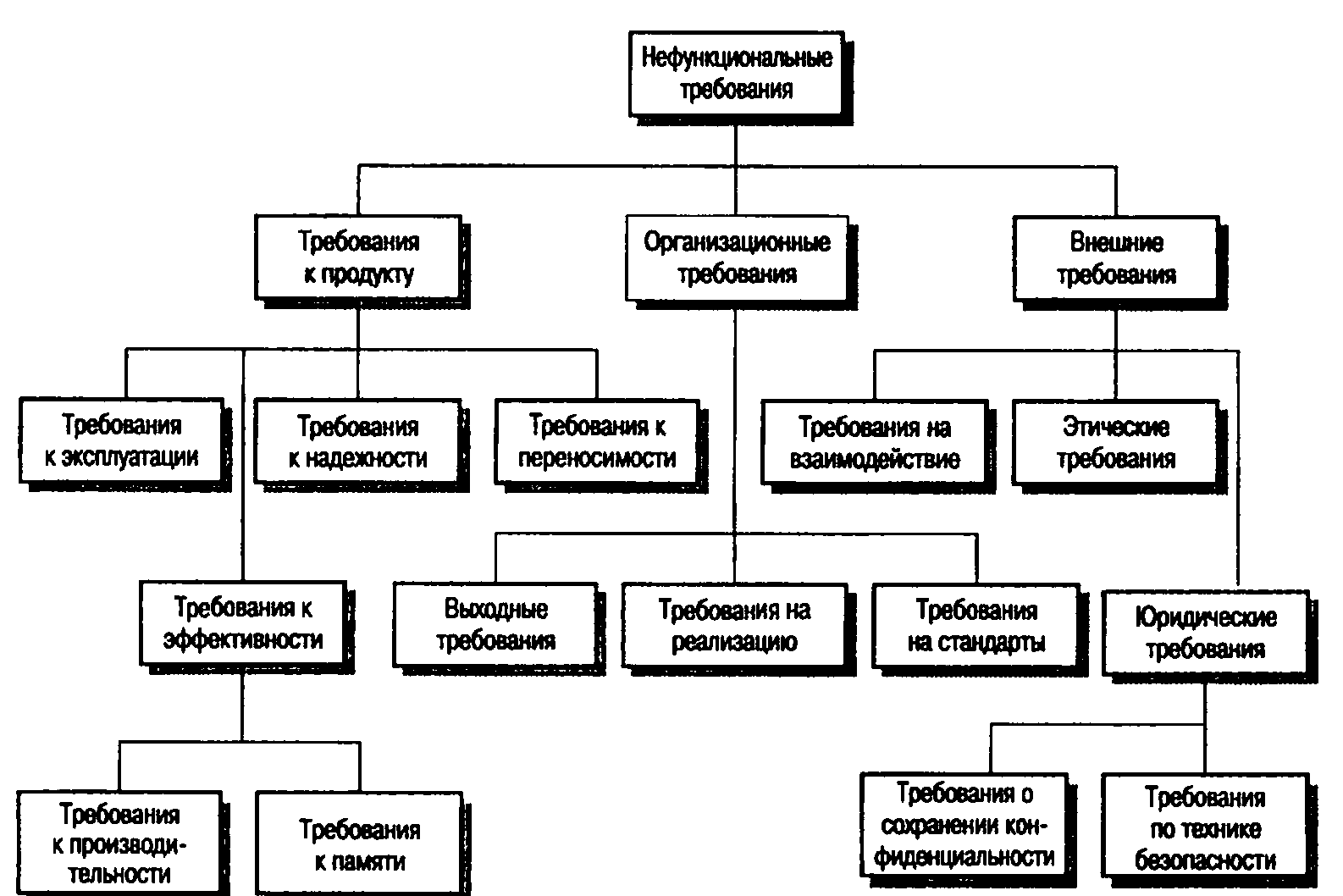


Рис. 4.4. Типы нефункциональных требований

Все нефункциональные требования, показанные на рис.4.4. разбиты на три большие группы.

1. Требования к продукту. Описывают эксплуатационные свойства программного продукта. Сюда относятся требования к производительности системы, объему необходимой памяти, надежности (определяет частоту возможных сбоев в системе), переносимости системы на разные компьютерные платформы и удобству эксплуатации.
2. Организационные требования. Отображают политику и организационные процедуры заказчика и разработчика ПО. Они включают стандарты разработки программного продукта, требования к реализации ПО (т.е. к языку программирования и методам проектирования), выходные требования, которые определяют сроки изготовления программного продукта, и сопутствующую документацию.
3. Внешние требования. Учитывают факторы, внешние по отношению к разрабатываемой системе и процессу ее разработки. Они включают требования, определяющие взаимодействие данной системы с другими системами, юридические требования, следование которым гарантирует, что система будет разрабатываться и функционировать в рамках существующего законодательства, а также этические требования. Последние должны гарантировать, что система будет приемлемой для пользователей или заказчика.

Основная проблема нефункциональных требований состоит в том, что их выполнение трудно проверить. Часто они пишутся для того, чтобы отобразить общие цели заказчика системы, такие, как простота эксплуатации, возможность восстановления после сбоев или быстрый ответ на запросы пользователя. Реализация подобных требований может оказаться сложной для системных разработчиков, поскольку они нечетко сформулированы и открывают простор для различных толкований. Подобную ситуацию иллюстрирует пример 2. Здесь одним из основных показателей (целей) системы указана простота эксплуатации, что в виде нефункциональных требований можно выразить различными способами. В данном случае требование сформулировано так, что его можно проверить.

Пример 2. Системные цели и проверка требований.

**Системная цель.**

Система должна быть простой, в эксплуатации для опытного оператора и сводить количество его ошибок к минимуму.

**Проверяемое нефункциональное требование.**

Опытному оператору должны быть доступны все системные функции после двух часов обучения работе с данной системой. После такого обучения число ошибок оператора не должно превышать двух за рабочий день.

В идеале нефункциональные требования должны выражаться через количественные показатели, которые можно объективно измерить. В табл. 4.1 приведены показатели, с помощью которых можно специфицировать нефункциональные системные свойства. На практике выразить нефункциональные требования с помощью количественных по­казателей весьма затруднительно. Часто заказчик ПО не может оформить свое видение будущей системы посредством требований, выраженных количественными показателями. Либо некоторые системные требования, например удобство сопровождения, вообще нельзя выразить через количественные показатели. Кроме того, затраты на объективное измерение количественных нефункциональных требований могут оказаться крайне высокими.

Нефункциональные требования часто вступают в конфликт с другими требованиями, предъявляемыми системе. Например, в соответствии с одним из системных требований размер системы не должен превышать 4 Мбайт, поскольку она должна полностью поместиться в постоянное запоминающее устройство ограниченной емкости. Другое требование обязывает использовать для написания системы язык программирования Ada, который часто применяется для создания критических систем реального времени. Но, допустим, откомпилированная системная программа, написанная на языке Ada, занимает более 4 Мбайт. Итак, одновременное выполнение этих требований невозможно. В этой ситуации следует отказаться от одного из требований. Можно или применить другой язык программирования, или увеличить объем памяти, выделяемый для системы.

Таблица 4.1. Количественные показатели нефункциональных требований

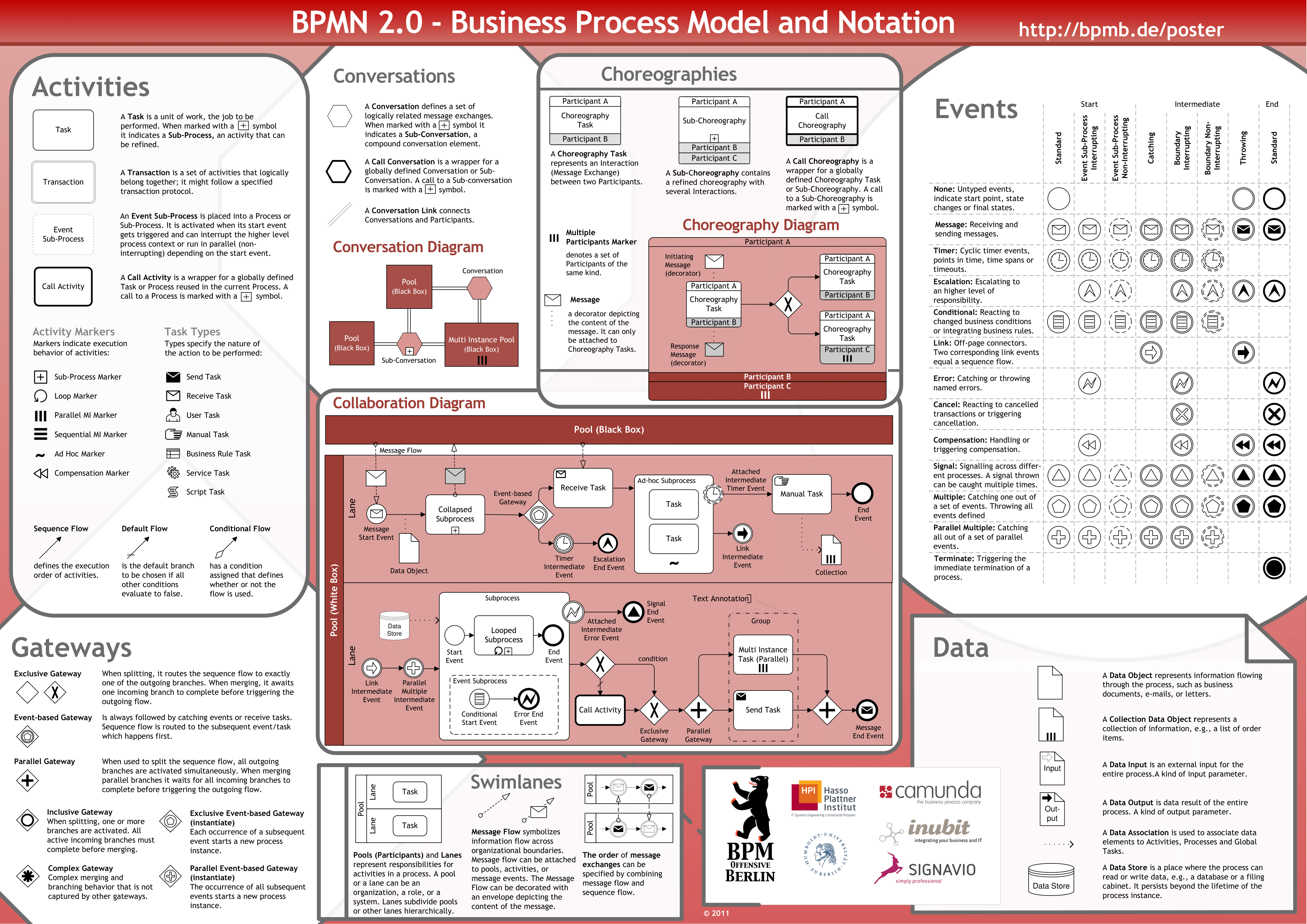
|  |  |
| --- | --- |
| Показатель | Единицы измерения |
| Скорость | Количество выполненных транзакций в секунду; |
|  | время реакции на действия пользователя; |
|  | время обновления экрана |
| Размер | Килобайты; |
|  | количество модулей памяти |
| Простота эксплуатации | Время обучения персонала; |
|  | количество статей в справочной системе |
| Надежность | Средняя продолжительность времени между двумя последовательными проявлениями ошибок в системе;  вероятность выхода системы из строя;  коэффициент готовности системы |
| Устойчивость к сбоям | Время восстановления системы после сбоя; процент событий, приводящих к сбоям; вероятность порчи данных при сбоях |
| Переносимость | Процент машинно-зависимых операторов; |
|  | количество машинно-зависимых подсистем |

## Задание и порядок проведения работы

Составить полные списки функциональных и нефункциональных требований к вашей системе.

# Лабораторная работа №4 Моделирование бизнес-процессов

## Краткие теоретические сведения



Хорошие и плохие примеры использования различных элементов: <https://stormbpmn.com/bpmn/elements>

Подробные уроки по нотации: <https://elma365.com/ru/articles/urok-1-vvod-v-notaciyu-bpmn/>

## Задание и порядок проведения работы

Ознакомиться с теоретическим материалом, разработать диаграмму BPMN для одного из главных бизнес-процессов в вашей системе в любом графическом редакторе (пример: draw.io, Visio).

Изобразить можно как процесс as is (как он работает до введения вашей ИС), так и to be (как будет происходить после введения ИС) на выбор, однако это нужно указать в описании схемы.

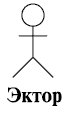
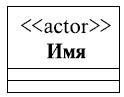
Главным бизнес-процессом любого магазина будет продажа чего-либо или предоставление услуг клиенту, системы для внутренней работы магазина – закупка заканчивающихся товаров, игры – игровой процесс от и до и т.п.

# Лабораторная работа №5 Построение диаграммы вариантов использования

Цель работы: освоение технологии проектирования ИС с помощью UML диаграмм

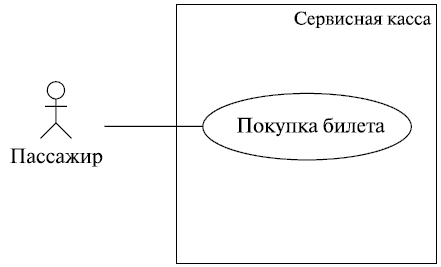
## Краткие теоретические сведения

Эктор – это набор ролей, которые исполняет пользователь в ходе взаимодействия с некоторой сущностью (системой, подсистемой, классом). Эктор может быть человеком, другой системой, подсистемой или классом, которые представляют нечто за пределами рассматриваемой сущности.

Прецедент – это описание набора последовательных событий (включая возможные варианты), выполняемых системой, которые приводят к наблюдаемому эктором результату. Прецеденты описывают сервисы, предоставляемые системой экторам, с которыми она взаимодействует. Причем прецедент никогда не объясняет, "как" работает сервис, а только описывает, "что" делается.

Изображаются прецеденты в виде эллипса, внутрь контура которого помещается имя (описание) прецедента. Имя прецедента обычно намного длиннее имен других элементов модели. Почему это так, в принципе, понятно: имя прецедента описывает взаимодействие эктора с системой, говорит о том, какими сообщениями они обмениваются между собой. В нашем примере с заказом обедов мы видели несколько прецедентов и наверняка читатель заметил, что имя прецедента – это, скорее, название сценария, воспроизводящегося в ходе взаимодействия эктора с системой. Причем это всегда описание с точки зрения эктора, описание услуг, предоставляемых системой пользователю. Приведем пример простейшей диаграммы, иллюстрирующей сказанное нами об обозначениях прецедента



Сценарий – это конкретная последовательность действий, иллюстрирующая поведение.

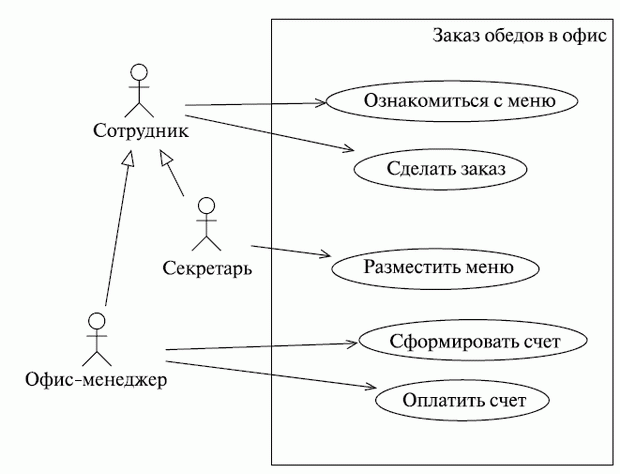
Сценарий – это повествовательный рассказ о совершаемых эктором действиях, история, эпизод, происходящий в данных временных рамках и данном контексте взаимодействия. Сценарии (в различных формах представления) широко применяются в процессе разработки программного обеспечения. Как мы уже только что отметили, написание сценария напоминает написание художественного рассказа, и этим объясняется тот факт, что использование сценариев широко распространено среди аналитиков, которые часто обладают художественными или литературными способностями. Несмотря на непрерывный повествовательный характер, сценарии можно рассматривать как последовательности действий (делать раскадровку ).

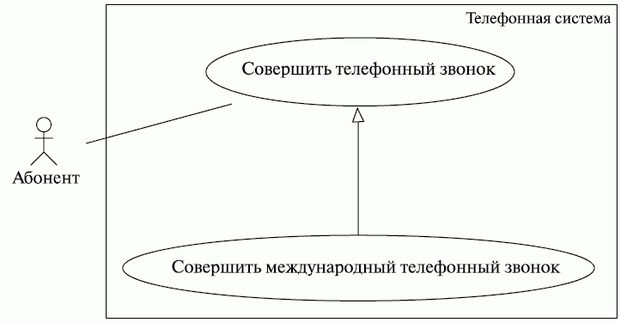


Как мы уже говорили выше, обобщение (наследование) чаще всего используют между классами и интерфейсами. Однако другие элементы модели также могут находиться между собой в отношении наследования – например, пакеты (о которых мы тут не говорим), экторы, прецеденты...

Изображается обобщение, как, конечно, помнит внимательный читатель, линией с "незакрашенной" треугольной стрелкой на конце. Обобщение – это отношение между предком и потомком, и стрелка всегда указывает на предка. Если вспомнить, что потомки наследуют (используют) свойства предка, то вполне логично вспоминается наше утверждение о том, что стрелки в UML всегда направлены в сторону того, от кого что-то требуют, чьими сервисами пользуются.

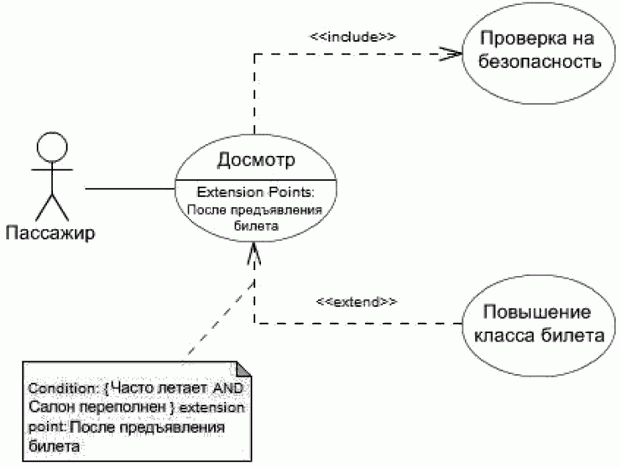
|  |  |
| --- | --- |
| **Прецедент** | **Действующее лицо** |
| разместить меню | секретарь |
| ознакомиться с меню | сотрудник, секретарь, офис-менеджер |
| сделать заказ | сотрудник, секретарь, офис-менеджер |
| сформировать счет | офис-менеджер |
| оплатить счет | офис-менеджер |





Отношение включения (include) означает, что в некоторой точке базового прецедента содержится поведение другого прецедента. Включаемый прецедент не существует сам по себе, а является всего лишь частью объемлющего прецедента. Таким образом, базовый прецедент как бы заимствует поведение включаемых, раскладываясь на более простые прецеденты.

Расширение (extend) дополняет прецедент другими прецедентами, "срабатывающими" при некоторых условиях, – просто добавляет в исходный прецедент последовательность действий, содержащуюся в другом прецеденте. Отношение расширения прецедента А к прецеденту В означает, что экземпляр прецедента В может включать в себя поведение, описанное в прецеденте А.Точка расширения описывается в дополнительном разделе прецедента, отделенном от его названия горизонтальной линией – точно так же, как в отдельных разделах перечисляются атрибуты класса и его операции.



## Задание и порядок проведения работы

Ознакомиться с теоретическим материалом, разработать диаграмму вариантов использования в любом редакторе.

# Лабораторная работа №6 Построение диаграммы последовательности

Цель работы: освоение технологии проектирования ИС с помощью UML диаграмм

## Краткие теоретические сведения

Диаграммы последовательностей используются для уточнения диаграмм прецедентов, более детального описания логики сценариев использования. Это отличное средство документирования проекта с точки зрения сценариев использования.

Диаграммы последовательностей обычно содержат объекты, которые взаимодействуют в рамках сценария, сообщения, которыми они обмениваются, и возвращаемые результаты, связанные с сообщениями. Впрочем, часто возвращаемые результаты обозначают лишь в том случае, если это не очевидно из контекста.

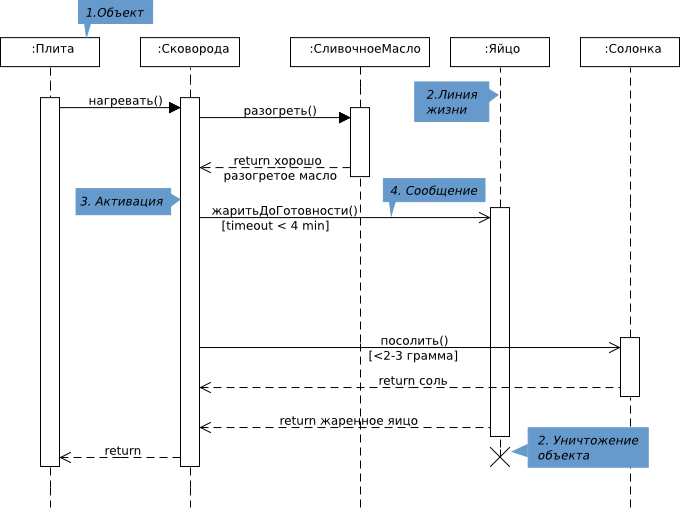
Объекты обозначаются прямоугольниками с подчеркнутыми именами (чтобы отличить их от классов).

Сообщения (вызовы методов) – линиями со стрелками.

Возвращаемые результаты – пунктирными линиями со стрелками.

Прямоугольники на вертикальных линиях под каждым из объектов показывают “время жизни” (фокус) объектов.

Диаграмма последовательности является временной диаграммой. Время в данной диаграмме течет сверху в низ

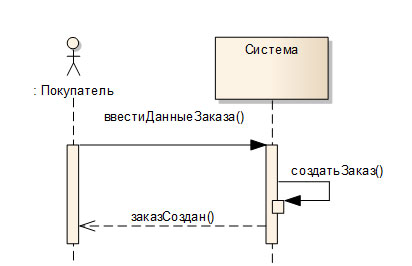


Данный вид диаграмм отражает следующие аспекты проектируемой Системы:

* обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними Системами)
* ограничения, накладываемые на взаимодействие объектов
* события, инициирующие взаимодействия объектов.

В отличие от диаграммы деятельности, которая показывает только последовательность (алгоритм) работы Системы, диаграммы взаимодействия акцентируют внимание разработчиков на сообщениях, инициирующих вызов определенных операций объекта (класса) или являющихся результатом выполнения операции.

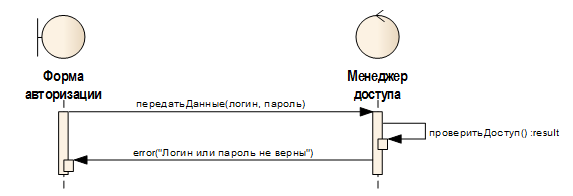
Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов Системы во времени, а также обмена сообщениями между ними.

Одним из основных принципов ООП является способ информационного обмена между элементами Системы, выражающийся в отправке и получении сообщений друг от друга. Таким образом, основные понятия диаграммы последовательности связаны с понятием Объект и Сообщение.  
  
На диаграмме последовательности объекты в основном представляю экземпляры класса или сущности, обладающие поведением. В качестве объектов могут выступать пользователи, инициирующие взаимодействие, классы, обладающие поведением в Системе или программные компоненты, а иногда и Системы в целом.

Объекты располагаются с лева на права таким образом, чтобы крайним с лева был тот объект, который инициирует взаимодействие.  
Неотъемлемой частью объекта на диаграмме последовательности является линия жизни объекта. Линия жизни показывает время, в течение которого объект существует в Системе. Периоды активности объекта в момент взаимодействия показываются с помощью фокуса управления. Временная шкала на диаграмме направлена сверху вниз.

На диаграммах последовательности допустимо использование стандартных стереотипов класса:

|  |  |
| --- | --- |
| http://it-gost.ru/images/articles/uml/seq_2.gif | Actor – экземпляр участника процесса (роль на диаграмме прецедентов) |
| http://it-gost.ru/images/articles/uml/seq_3.gif | Boundary – Класс-Разграничитель – используется для классов, отделяющих внутреннюю структуру системы от внешней среды (экранная форма, пользовательский интерфейс, устройство ввода-вывода). Объект со стереотипом <<bondary>> отличается от, привычного нам, класса <<Интерфейс>>, который по большей части предназначен для вызова методов класса, с которым он связан. Объект boundary показывает именно экранную форму, которая принимает и передает данные обработчику.</bondary>> |
| http://it-gost.ru/images/articles/uml/seq_4.gif | Control – Класс-контроллер – активный элемент, который используются для выполнения некоторых операций над объектами (программный компонент, модуль, обработчик) |
| http://it-gost.ru/images/articles/uml/seq_5.gif | Entity – Класс-сущность – обычно применяется для обозначения классов, которые хранят некую информацию о бизнес-объектах (соответствует таблице или элементу БД) |

Также одним из основных понятий, связанных с диаграммой последовательности, является Сообщение.  


На диаграмме деятельности выделяются сообщения, инициирующие ту или иную деятельность или являющиеся ее следствием. На диаграмме состояний частично показан обмен сообщениями в рамках сообщений инициирующих изменение состояния объекта.

Диаграмма последовательности объединяет диаграмму деятельности, диаграмму состояний и диаграмму классов.

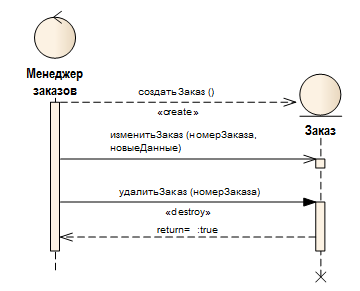
Таким образом, на диаграмме последовательности мы можем увидеть следующие аспекты:

Сообщения, побуждающие объект к действию

Действия, которые вызываются сообщениями (методы) – зачастую это передача сообщения следующему объекты или возвращение определенных данных объекта

Последовательность обмена сообщениями между объектами

Итак, прием сообщения инициирует выполнение определенных действий, направленных на решение отдельной задачи тем объектом, которому это сообщение отправлено. Сообщение в большинстве случаев (за исключением диаграмм, описывающих концептуальный уровень Системы) это вызов методов отдельных объектов, поэтому для корректного исполнения метода в сообщении необходимо передать какие-то данные и определить, что мы хотим видеть в ответ. При именовании сообщения на уровне проектирования реализации системы в качестве имени сообщения следует использовать имя метода.



В UML различают следующие виды сообщений:

* синхронное сообщение (synchCall) – соответствует синхронному вызову операции и подразумевает ожидание ответа от объекта получателя. Пока ответ не поступит, никаких действий в Системе не производится.
* асинхронное сообщение (asynchCall) – которое соответствует асинхронному вызову операции и подразумевает, что объект может продолжать работу, не ожидая ответа.
* ответное сообщение (reply) – ответное сообщение от вызванного метода. Данный вид сообщения показывается на диаграмме по мере необходимости или, когда возвращаемые им данные несут смысловую нагрузку.
* потерянное сообщение (lost) – сообщение, не имеющее адресата сообщения, т.е. для него существует событие передачи и отсутствует событие приема
* найденное сообщение (found) – сообщение, не имеющее инициатора сообщения, т.е. для него существует событие приема и отсутствует событие передачи

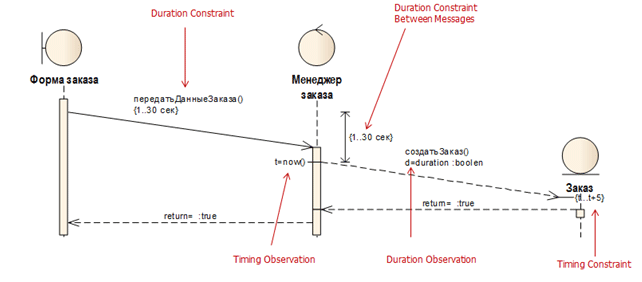
Для сообщений также доступен ряд предопределенных стереотипов. Наиболее часто используемые стереотипы это create и destroy.

Сообщение со стереотипом create, вызывает в классе метод, которые создает экземпляр класса. На диаграмме последовательности не обязательно показывать с самого начала все объекты, участвующие во взаимодействии. При использовании сообщения со стереотипом create, создаваемый объект отображается на уровне конца сообщения.

Для уничтожения экземпляра класса используется сообщение со стереотипом destroy, при этом в конце линии жизни объекта отображаются две перекрещенные линии.

При отображении работы с сообщениями иногда возникает необходимость указать некоторые временные ограничения. Например, длительность передачи сообщения или ожидание ответа от объекта не должно превышать определенный временной интервал. Можно указать следующие временные параметры:

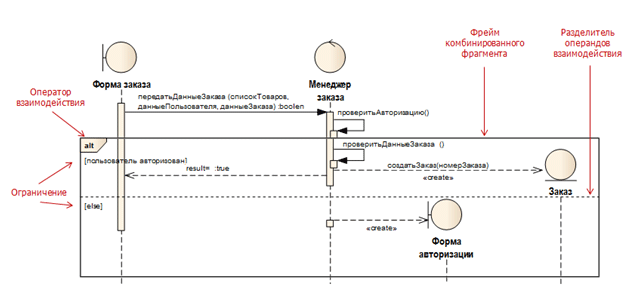
* ограничение продолжительности (Duration Constraint) – минимальное и максимальное значение продолжительности передачи сообщения
* ограничение продолжительности ожидания между передачей и получением сообщения (Duration Constraint Between Messages)
* перехват продолжительности сообщения (Duration Observation)
* временное ограничение (Timing Constraint) – временной интервал, в течение которого сообщение должно прийти к цели (устанавливается на стороне получателя)
* перехват времени, когда сообщение было отправлено (Timing Observation)



Форма заказа передает данные Менеджеру заказа, при этом передача данных не должна длиться больше 30 сек. – данное ограничение может понадобиться при выявлении требований к быстродействию Системы. Далее получение данных с формы инициирует запуск метода для создания экземпляра класса Заказ. Между получением данных от Формы заказа и инициализацией создания объекта должно пройти не более 30 сек., в противном случае пользователю может быть предоставлено сообщение об ошибке или недоступности сервера. Длительность передачи сообщения о создании объекта может быть зафиксирована в переменной d.

Данное значение может понадобиться при установке временного ограничения на получение ответного сообщения клиентом. В момент передачи сообщения фиксируется временное значение и заносится в переменную t. Таким образом, можно установить ограничение на стороне приемника, указав переменную t в качестве минимального значения и t+<допустимый интервал> в качестве максимального значения.

До появления UML 2.0 диаграмма последовательности рассматривалась только в рамках моделирования последовательности обмена сообщениями. Расширение сценария отображались с помощью ветвления линий сообщений, что не давало полной картины взаимодействия объектов. Таким образом, для целей моделирования расширения сценария, параллельности процессов или цикличности использовались диаграммы деятельности. Для решения данных задач в UML 2.0 было введено понятие фрейма взаимодействия и операторов взаимодействия.



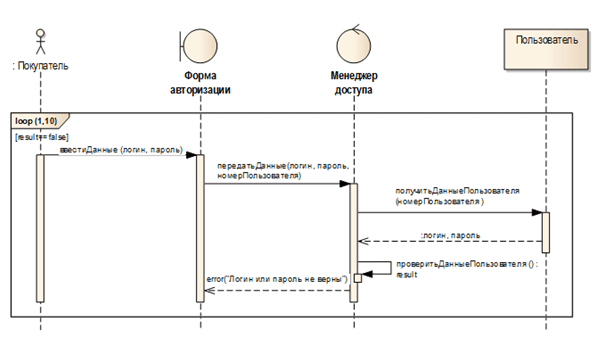
Отдельные фрагменты диаграммы взаимодействия можно выделить с помощью фрейма. Фрейм должен содержать метку оператора взаимодействия. UML содержит следующие операнды:

* Alt – Несколько альтернативных фрагментов (alternative); выполняется только тот фрагмент, условие которого истинно
* Opt – Необязательный (optional) фрагмент; выполняется, только если условие истинно. Эквивалентно alt с одной веткой
* Par – Параллельный (parallel); все фрагменты выполняются параллельно
* loop – Цикл (loop); фрагмент может выполняться несколько раз, а защита обозначает тело итерации
* region – Критическая область (critical region); фрагмент может иметь только один поток, выполняющийся за один прием
* Neg – Отрицательный (negative) фрагмент; обозначает неверное взаимодействие
* ref – Ссылка (reference); ссылается на взаимодействие, определенное на другой диаграмме. Фрейм рисуется, чтобы охватить линии жизни, вовлеченные во взаимодействие. Можно определять параметры и возвращать значение
* Sd – Диаграмма последовательности (sequence diagram); используется для очерчивания всей диаграммы последовательности, если это необходимо.

При использовании фрагмента условного операнда фрейм должен содержать условие для ограничения взаимодействия. При использовании условного или параллельного операнда фрейм делится на регионы взаимодействия с помощью разделителя операторов взаимодействия.

К условным операндам относятся alt и opt. Операнд alt используется при моделировании расширения сценария, т.е. при наличии альтернативного потока взаимодействия. Оператор opt используется, если сообщение должно быть передано, только при истинности какого-то условия. Данный фрейм используется без разделения на регионы.

Параллельность потоков взаимодействия можно изобразить с помощью операнда par. Внутри фрейма моделируются потоки взаимодействия в отдельных регионах.

Цикличность потока взаимодействия может быть представлена на диаграмме последовательности с помощью операнда loop. При использовании оператора цикла можно указать минимальное и максимальное число итераций. Также фрейм должен содержать условие, при наступлении которого взаимодействие повторяется.  


Диаграммы последовательности предназначены для моделирования взаимодействия между несколькими объектами. Зачастую диаграммы последовательности создаются для моделирования взаимодействия в рамках одного прецедента.

## Задание и порядок проведения работы

Ознакомиться с теоретическим материалом, разработать диаграмму последовательности для одного из важных вариантов использования в любом редакторе.

От вас ожидается как минимум наличие следующих взаимодействующих объектов: Пользователь (роль указать), Клиентская часть, Серверная часть. Разделение на клиент и сервер обязательное.

# Лабораторная работа №7 Построение диаграммы активности

Цель работы: освоение технологии проектирования ИС с помощью UML диаграмм

## Краткие теоретические сведения

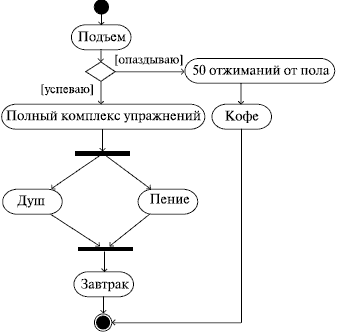
Диаграммы активностей (Activity Diagrams) являются представлением алгоритмов неких действий (активностей), выполняющихся в системе.

Можно построить несколько диаграмм деятельности для одной и той же системы, причем каждая из них будет фокусироваться на разных аспектах системы, показывать различные действия, выполняющиеся внутри ее. Говоря более формально, диаграммы активности, в общем-то, не имеют монополии на описание поведенческих особенностей динамических частей системы. Для этой же цели могут использоваться еще диаграммы прецедентов, последовательности, кооперации и состояний.

Именно на диаграмме деятельности представлены переходы потока управления от одной деятельности к другой. Это, по сути, разновидность диаграммы состояний, где все или большая часть состояний являются некоторыми деятельностями, а все или большая часть переходов срабатывают при завершении определенной деятельности и позволяют перейти к выполнению следующей. Как мы уже говорили (повторение – мать учения), диаграмма деятельности может быть присоединена к любому элементу модели, имеющему динамическое поведение. Кстати, исходя из вышесказанного, логичнее говорить не "диаграмма деятельности", а "диаграмма деятельностей" – во множественном числе. А еще мы предполагаем, что читатель понимает смысл понятий "деятельность", "переход" и "объект". Об объектах как об экземплярах классов мы уже говорили ранее. Понятия же деятельности (activity) как протяженного во времени составного (неатомарного) вычисления (действия, action) и перехода как передачи контроля, надеемся, понятны интуитивно, без дополнительных объяснений.

Диаграммы деятельности позволяют моделировать сложный жизненный цикл объекта, с переходами из одного состояния (деятельности) в другое. Но этот вид диаграмм может быть использован и для описания динамики совокупности объектов. Они применимы и для детализации некоторой конкретной операции, причем, как мы увидим далее, предоставляют для этого больше возможностей, чем "классическая" блок-схема. Диаграммы деятельности описывают переход от одной деятельности к другой, в отличие от диаграмм взаимодействия, где акцент делается на переходах потока управления от объекта к объекту.

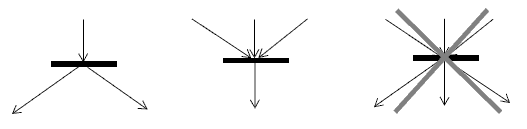
Как говорится, лучше один раз увидеть, чем сто раз услышать. Мы достаточно разрекламировали диаграммы деятельностей. Пора взглянуть на пример.



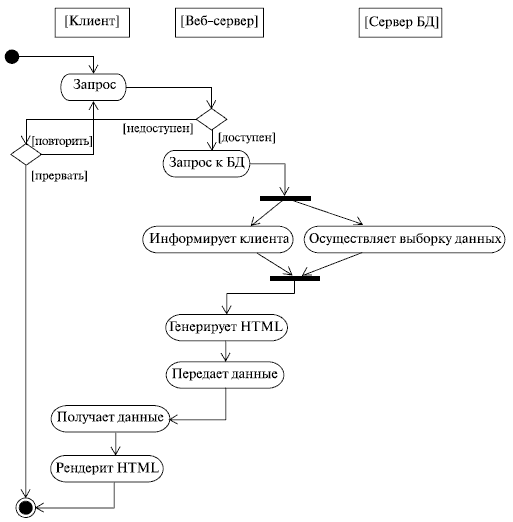
Эта диаграмма довольно точно описывает ежеутреннюю последовательность действий автора этих строк (до момента ухода на работу). Как видим, все очень просто и понятно. Действия показаны скругленными прямоугольниками, как в блок-схеме, – мы узнаем даже ромбик символа принятия решения с обозначениями условий возле переходов. Да, отличия от блок-схемы не так уж сильны. Более того, эти отличия выглядят как логичное расширение нотации блок-схем. Обратим внимание на то, что начало и конец уже не изображаются одинаковым безликим кружком. Начало теперь закрашено, а конец изображен в виде символа, напоминающего кошачий глаз (кстати, это образное название – "кошачий глаз" – уже намертво въелось в жаргон архитекторов и аналитиков).

http://www.intuit.ru/EDI/23_04_17_1/1492899714-28128/tutorial/356/objects/4/files/04_02.gif

Без пояснений понятен также смысл символа, предшествующего принятию душа и пению и следующего за ними – он означает распараллеливание, а затем опять слияние воедино ( синхронизацию ) потоков управления, т. е. операции "пение" и "душ" выполняются одновременно. Нотация проста: несколько потоков управления сливаются в один или один поток разделяется на несколько. Третьего не дано.

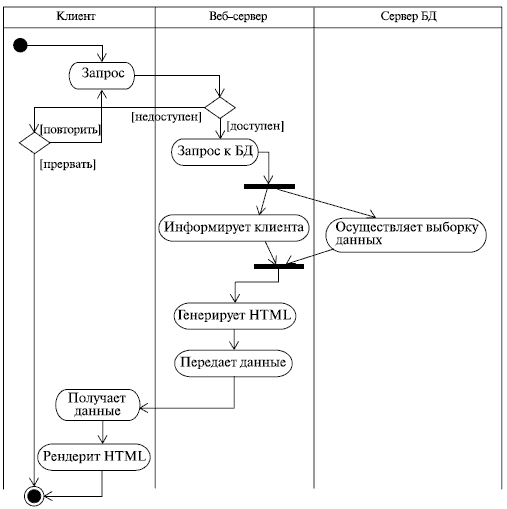


Конечно, это не единственные отличия диаграммы активностей от блок-схемы. На диаграмме деятельностей можно не только показать параллельно выполняемые действия, но и указать состояния объектов (так же, как и на представлениях конечных автоматов, о которых нам так много говорили в университетах), также есть возможность показывать распределение ролей и т. д. Вот еще пример, подтверждающий, что диаграмма активностей – это нечто большее, чем блок-схема.



Смысл диаграммы вполне понятен и без дополнительных объяснений. Как вы уже, конечно, догадались, на ней показана работа с веб-приложением, которое решает некую задачу в удаленной базе данных. Привлекает внимание странное расположение активностей на этой диаграмме: они как бы разбросаны по трем беговым дорожкам, каждая из которых соответствует поведению одного из трех объектов – клиента, веб-сервера и сервера баз данных. Благодаря этому легко определить, каким из объектов выполняется каждая из активностей, и неожиданно приходит понимание того, что "странность" этой диаграммы, оказывается, очень упрощает ее восприятие.

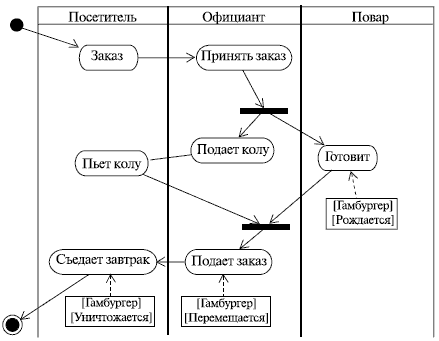
Аналогия с дорожками действительно очень удачна. Именно таково официальное название элемента нотации UML, позволяющего указать распределение ролей на диаграмме активностей. Только дорожки это не беговые, а плавательные – они так и называются: swimlanes. Более формально, дорожка – часть области диаграммы деятельности, на которой отображаются только те деятельности, за которые отвечает конкретный объект.



Предназначены они для разбиения диаграммы в соответствии с распределением ответственности за действия. Имя дорожки может означать роль или объект, которому она соответствует. При использовании дорожек нотация слегка изменяется. Вот как, к примеру, выглядит диаграмма из предыдущего примера, перерисованная с использованием дорожек.

Кстати, дорожки могут быть не только вертикальными, но и, если вам как автору так удобнее, горизонтальными. Изображаются горизонтальные дорожки аналогично – просто поверните "обычные" дорожки на 90 градусов против часовой стрелки!

Есть еще один нюанс нотации диаграмм активностей, о котором мы пока не говорили: это так называемая траектория объекта, или поток объекта ( object flow ). Суть его состоит в том, что на диаграмме деятельности можно изобразить и объекты, относящиеся к деятельности. С помощью символа зависимости (пунктирная стрелка, помните?) эти объекты можно соотнести с той деятельностью или переходом, где они создаются, изменяются или уничтожаются. Представим такую ситуацию из повседневной жизни: вы приходите в какой-нибудь фастфуд и заказываете гамбургер с колой. Что, знакомо? Во время приготовления завтрака повар создает новый объект – гамбургер. Пока вы нетерпеливо выпиваете колу, официант перемещает этот объект (подает ваш заказ). Естественно, во время завтрака вы уничтожаете этот объект. Вот как это выглядит на диаграмме .



На этом можно было бы и закончить наш разговор о нотации диаграмм активностей и их отличиях от блок-схем. Если бы не одно НО. Мы говорили, что деятельность – это протяженное по времени составное действие. Составное! То есть составленное из более простых действий. Вот эти-то самые простые (атомарные) действия, а вернее, последовательность их выполнения, частенько изображают внутри деятельности в виде маленькой диаграммы активностей. Это слегка напоминает матрешку – одна (а часто и не одна) диаграмма внутри другой. Мы не будем долго говорить об этом: нашей целью было просто обратить внимание читателя на подобную возможность "вложенных" диаграмм. Мы просто покажем пример, позаимствованный нами из Zicom Mentor

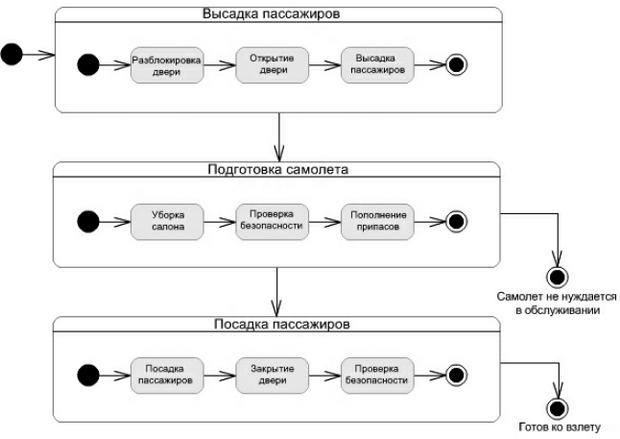


Диаграмма описывает высадку пассажиров самолета, достигших пункта назначения, и посадку новых пассажиров. Предлагаем читателю самому внимательно рассмотреть эту диаграмму. Из нее, например, можно почерпнуть, что конечных состояний может быть больше одного. Кстати, кроме начального и конечного состояний есть еще конечное состояние потока (Flow final mode). От конечного состояния оно отличается вот чем: конечное состояние потока означает завершение одного потока управления, а конечное состояние говорит о завершении всех потоков управления внутри деятельности. Обозначается конечное состояние потока простым символом, напоминающим лампочку накаливания в схемах электрических цепей



## Задание и порядок проведения работы

Ознакомиться с теоретическим материалом, разработать диаграмму активности для одного из важных вариантов использования в любом редакторе. Вариант использования не должен совпадать с тем, что выбран в ЛР 6.

От вас ожидается как минимум наличие следующих дорожек: Пользователь (роль указать), Клиентская часть, Серверная часть. Разделение на клиент и сервер обязательное.

# Лабораторная работа №8 Построение диаграммы состояний

Цель работы: освоение технологии проектирования ИС с помощью UML диаграмм

## Краткие теоретические сведения

Диаграмма состояний – UML State Machine Diagram – это диаграмма из разряда поведенческих, которая показывает то, какие состояния может иметь объект моделирования и как они меняются в течение жизни этого самого объекта.

Диаграмма состояний:

* Диаграмма состояний позволяет описывать поведение системы.
* В объектно-ориентированном подходе разрабатывается диаграмма состояний единственного класса, демонстрирующая поведение одного объекта в течение его жизни.
* Состояние на диаграмме является более абстрактным понятием, чем состояние объекта (последнее есть комбинация всех данных из полей объекта).
* Диаграмма позволяет проектировать различные способы реакции на события.

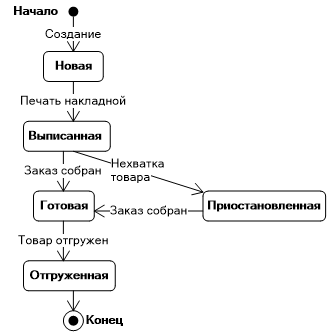
**Основные элементы диаграммы состояний**

На диаграмме состояний можно отобразить следующие элементы нотации UML, доступные в панели элементов:

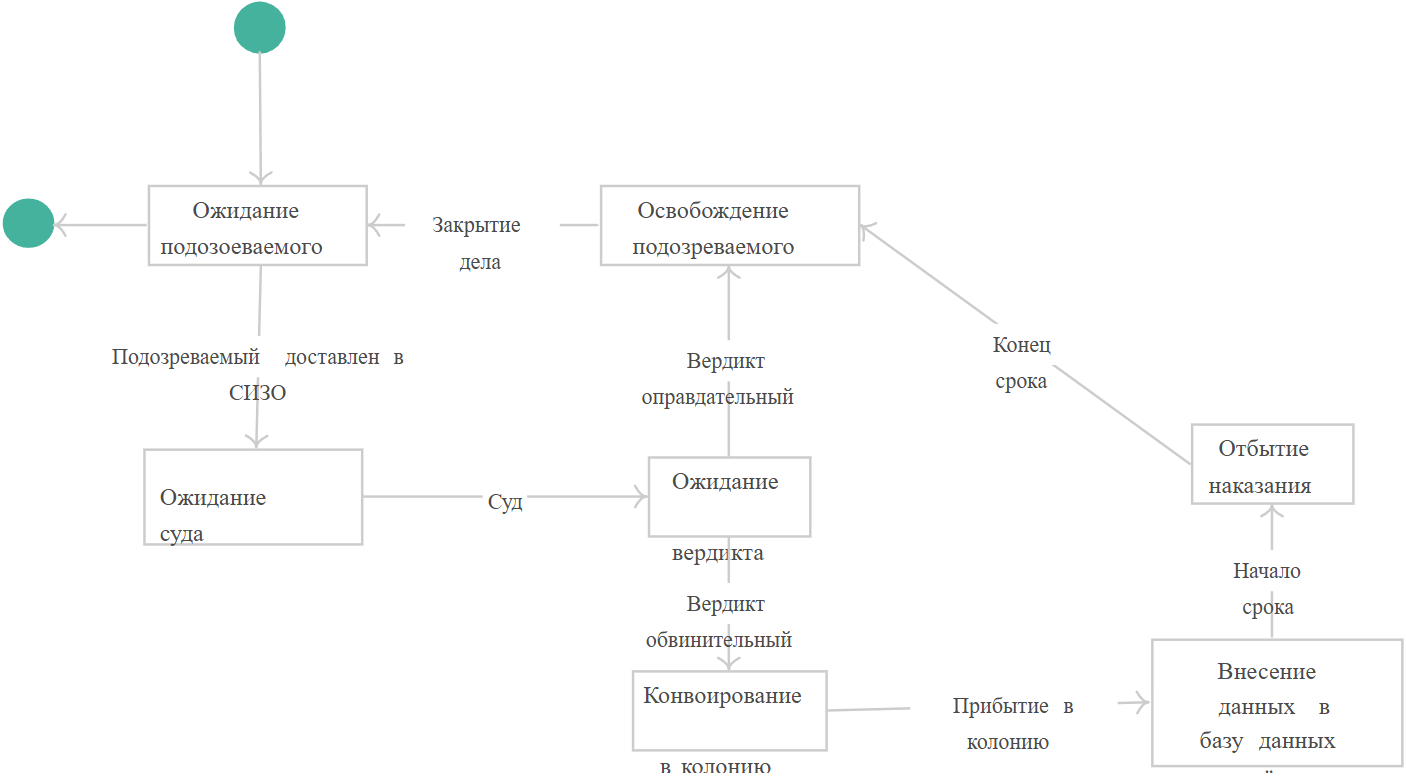
|  |  |
| --- | --- |
| Элемент/Нотация | Предназначение |
| Пример | Состояние (State) |
| Пример | Состояние с дополнительными параметрами |
| Пример | Составное состояние (Composite state) |
| Пример | Разделитель (Concurrent state) |
| Пример | Начальное состояние (Start state) |
| Пример | Конечное состояние (Final state) |
| Пример | Переход (Transition) |
| Пример | Сообщение (Event message) |
| Пример | Комментарий (Note) |
| Пример | Коннектор комментария (Note connector) |

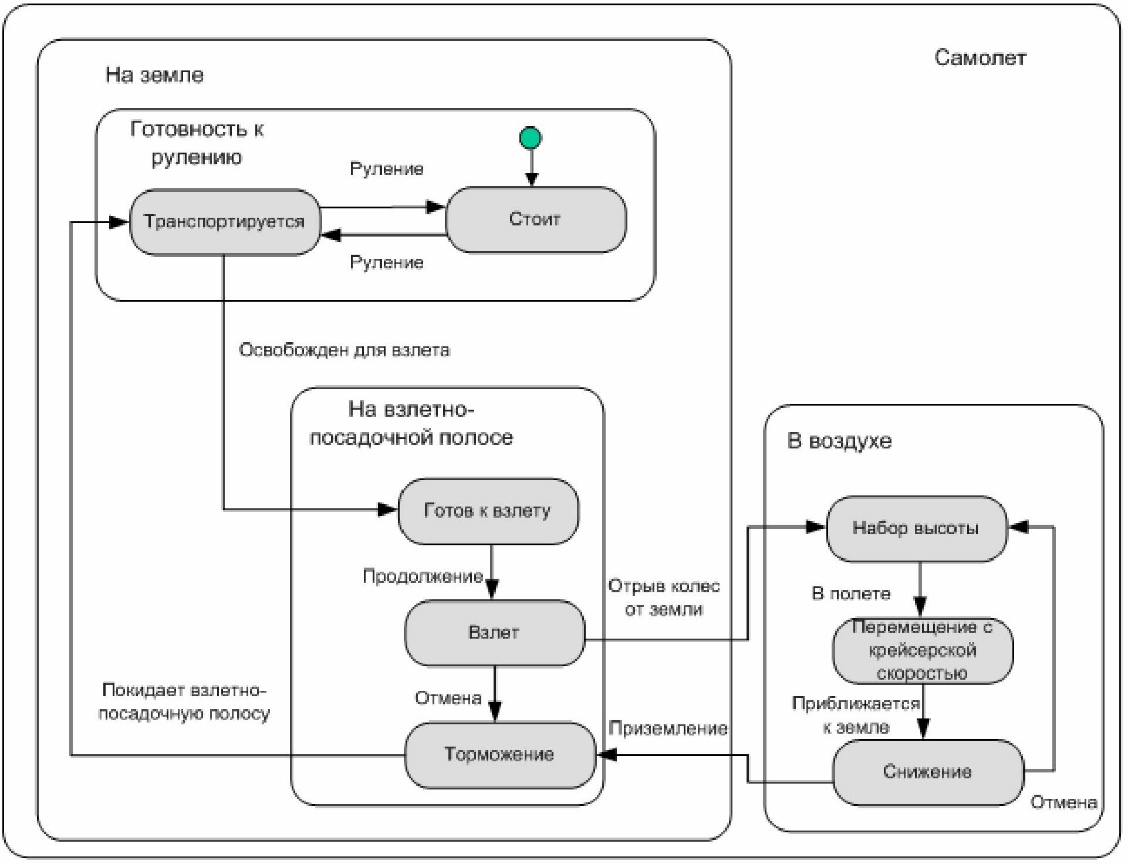
**Примеры диаграмм**

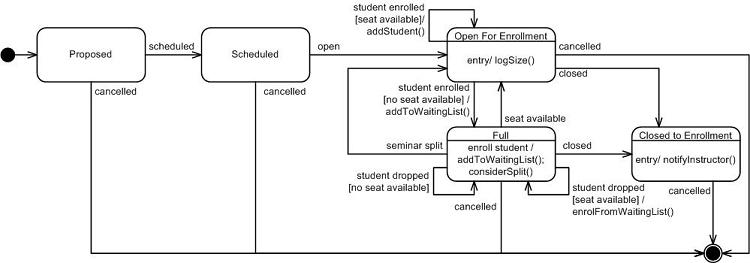
Самый минимум для ЛР:



Другие







## Задание и порядок проведения работы

Ознакомиться с теоретическим материалом, разработать диаграмму состояний для одного из значимых объектов системы, либо для всей ИС целиком/отдельной ее формы или компонента.