ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
КЕМЕРОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Кафедра ЮНЕСКО по Новым информационным технологиям

А.М. Гудов, С.Ю. Завозкин, С.Н. Трофимов

ТЕХНОЛОГИЯ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Лабораторный практикум

Математический факультет

Специальность 010503 – математическое обеспечение и администрирование информационных систем

Кемерово, 2009

УДК

Технология разработка программного обеспечения дисциплина. Лабораторный практикум. / А.М. Гудов, С.Ю. Завозкин, С.Н. Трофимов. – Кемерово, КемГУ, 2009. – С. 102

Рецензенты:

Пиманов А.Г., заведующий кафедрой вычислительной техники и информационных технологий Кузбасского государственного технического университета, доктор техн. наук, профессор.

Перминов В.А., доцент кафедры математики и естественных наук Беловского института (филиала) Кемеровского государственного университета, канд. физ.-мат. наук, доцент.

Лабораторная работа №1

«Разработка описания и анализ информационной системы»

1. Цель работы:

Описать и проанализировать информационную систему, распределить роли в группе разработчиков.

2. Методические указания

Лабораторная работа направлена на ознакомление с процессом описания информационной системы и получение навыков по использованию основных методов анализа ИС.

Требования к результатам выполнения лабораторного практикума:

1. наличие описания информационной системы;
2. проведение анализа осуществимости выполнения проекта;
3. наличие заключения о возможности реализации проекта, содержащего рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на странице http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретические сведения

Общие сведения о разработке программного обеспечения

Проблемы управления программными проектами впервые проявились в 60-х - начале 70-х годов, когда провалились многие большие проекты по разработке программных продуктов. Были зафиксированы задержки в создании ПО, оно было ненадежным, затраты на разработку в несколько раз превосходили первоначальные оценки, созданные программные системы часто имели низкие показатели производительности. Причины провалов коренились в тех подходах, которые использовались в управлении проектами. Применяемая методика была основана на опыте управления техническими проектами и оказалась неэффективной при разработке программного обеспечения.

Важно понимать разницу между профессиональной разработкой ПО и любительским программированием. Необходимость управления программными проектами вытекает из того факта, что процесс создания профессионального ПО всегда является субъектом бюджетной политики организации, где оно разрабатывается, и имеет временные ограничения. Работа *руководителя программного проекта* по большому счету заключается в том, чтобы гарантировать выполнение этих бюджетных и временных ограничений с учетом бизнес-целей организации относительно разрабатываемого ПО.

Руководители проектов призваны спланировать все этапы разработки программного продукта. Они также должны контролировать ход выполнения работ и соблюдения всех

требуемых стандартов. Постоянный контроль за ходом выполнения работ необходим для того, чтобы процесс разработки не выходил за временные и бюджетные ограничения. Хорошее управление не гарантирует успешного завершения проекта, но плохое управление обязательно приведет к его провалу. Это может выразиться в задержке сроков сдачи готового ПО, в превышении сметной стоимости проекта и в несоответствии готового ПО спецификации требований.

Процесс разработки ПО существенно отличается от процессов реализации технических проектов, что порождает определенные сложности в управлении программными проектами:

1. *Программный продукт нематериален.* Программное обеспечение нематериально, его нельзя увидеть или потрогать. Руководитель программного проекта не видит процесс "роста" разрабатываемого ПО. Он может полагаться только на документацию, которая фиксирует процесс разработки программного продукта.
2. *Не существует стандартных процессов разработки ПО.* На сегодняшний день не существует четкой зависимости между процессом создания ПО и типом создаваемого программного продукта. Другие технические дисциплины имеют длительную историю, процессы разработки технических изделий многократно опробованы и проверены. Процессы создания большинства технических систем хорошо изучены. Изучением же процессов создания ПО специалисты занимаются только последнее время. Поэтому пока нельзя точно предсказать, на каком этапе процесса разработки ПО могут возникнуть проблемы, угрожающие всему программному проекту.
3. *Большие программные проекты - это часто "одноразовые" проекты.* Большие программные проекты, как правило, значительно отличаются от проектов, реализованных ранее. Поэтому, чтобы уменьшить неопределенность в планировании проекта, руководители проектов должны обладать очень большим практическим опытом. Но постоянные технологические изменения в компьютерной технике и коммуникационном оборудовании обесценивают предыдущий опыт. Знания и навыки, накопленные опытом, могут не востребоваться в новом проекте.

Перечисленные отличия могут привести к тому, что реализация проекта выйдет из временного графика или превысит бюджетные ассигнования. Программные системы зачастую оказываются новинками как в "идеологическом", так и в техническом плане. Поэтому, предвидя возможные проблемы в реализации программного проекта, следует всегда помнить, что многим из них свойственно выходить за рамки временных и бюджетных ограничений.

Процесс управления разработкой программного обеспечения

Невозможно описать и стандартизировать все работы, выполняемые в проекте по созданию ПО. Эти работы весьма существенно зависят от организации, где выполняется разработка ПО, и от типа создаваемого программного продукта. Но всегда можно выделить следующие:

- Написание предложений по созданию ПО.
- Планирование и составление графика работ по созданию ПО.
- Оценивание стоимости проекта.

– Подбор персонала.

– Контроль за ходом выполнения работ.

– Написание отчетов и представлений.

Первая стадия программного проекта может состоять из написания предложений по реализации этого проекта. Предложения должны содержать описание целей проектов и способов их достижения. Они также обычно включают в себя оценки финансовых и временных затрат на выполнение проекта. При необходимости здесь могут приводиться обоснования для передачи проекта на выполнение сторонней организации или команде разработчиков.

Написание предложений — очень ответственная работа, так как для многих организаций вопрос о том, будет ли проект выполняться самой организацией или разрабатываться по контракту сторонней компанией, является критическим. Не существует каких-либо рекомендаций по написанию предложений, многое здесь зависит от опыт.

На этапе *планирования проекта* определяются процессы, этапы и полученные на каждом из них результаты, которые должны привести к выполнению проекта. Реализация этого плана приведет к достижению целей проекта. Определение стоимости проекта напрямую связано с его планированием, поскольку здесь оцениваются ресурсы, требующиеся для выполнения плана.

Контроль за ходом выполнения работ (мониторинг проекта) — это непрерывный процесс, продолжающийся в течение всего срока реализации проекта. Руководитель должен постоянно отслеживать ход реализации проекта и сравнивать фактические и плановые показатели выполнения работ с их стоимостью. Хотя многие организации имеют механизмы формального мониторинга работ, опытный руководитель может составить ясную картину о стадии развитии проекта просто путем неформального общения с разработчиками.

Неформальный мониторинг часто помогает обнаружить потенциальные проблемы, которые в явном виде могут обнаружиться позднее. Например, ежедневное обсуждение хода выполнения работ может выявить отдельные недоработки в создаваемом программном продукте. Вместо ожидания отчетов, в которых будет отражен факт "пробуксовки" графика работ, можно обсудить со специалистами намечающиеся программистские проблемы и не допустить срыва графика работ.

В течение реализации проекта обычно происходит несколько формальных контрольных проверок хода выполнения работ по созданию ПО. Такие проверки должны дать общую картину хода реализации проекта в целом и показать, насколько уже разработанная часть ПО соответствует целям проекта.

Время выполнения больших программных проектов может занимать несколько лет. В течение этого времени цели и намерения организации, заказавшей программный проект, могут существенно измениться. Может оказаться, что разрабатываемый программный продукт стал уже ненужным либо исходные требования к создаваемому ПО просто устарели и их необходимо кардинально менять. В такой ситуации руководство организации-разработчика может принять решение о прекращении разработки ПО или об изменении проекта в целом с тем, чтобы учесть изменившиеся цели и намерения организации-заказчика.

Руководители проектов обычно обязаны сами *подбирать исполнителей* для своих проектов. В идеальном случае профессиональный уровень исполнителей должен соответствовать той работе, которую они будут выполнять в ходе реализации проекта.

Однако во многих случаях руководители должны полагаться на команду разработчиков, которая далека от идеальной. Такая ситуация может быть вызвана следующими причинами:

1. Бюджет проекта не позволяет привлечь высококвалифицированный персонал. В таком случае за меньшую плату привлекаются менее квалифицированные специалисты.
2. Бывают ситуации, когда невозможно найти специалистов необходимой квалификации как в самой организации-разработчике, так и вне ее. Например, в организации "лучшие люди" могут быть уже заняты в других проектах.
3. Организация хочет повысить профессиональный уровень своих работников. В этом случае она может привлечь к участию в проекте неопытных или недостаточно квалифицированных работников, чтобы они приобрели необходимый опыт и поучились у более опытных специалистов.

Таким образом, почти всегда подбор специалистов для выполнения проекта имеет определенные ограничения и не является свободным. Вместе с тем необходимо, чтобы хотя бы несколько членов группы разработчиков имели квалификацию и опыт, достаточные для работы над данным проектом. В противном случае невозможно избежать ошибок в разработке ПО.

Руководитель проекта обычно обязан посылать *отчеты* о ходе его выполнения как заказчику, так и подрядным организациям. Это должны быть краткие документы, основанные на информации, извлекаемой из подробных' отчетов о проекте. В этих отчетах должна быть та информация, которая позволяет четко оценить степень готовности создаваемого программного продукта.

В рамках курса «Технология разработки программного обеспечения» выделены следующие роли в группе по разработке ПО:

- **Руководитель – общее руководство проектом, написание документации, общение с заказчиком ПО**
- **Системный аналитик – разработка требований (составление технического задания, проекта программного обеспечения)**
- **Тестер – составление плана тестирования и аттестации готового ПО (продукта), составление сценария тестирования, базовый пример, проведение мероприятий по плану тестирования**
- **Разработчик – моделирование компонент программного обеспечения, кодирование**

Планирование проекта разработки программного обеспечения

Эффективное управление программным проектом напрямую зависит от правильного планирования работ, необходимых для его выполнения. План помогает руководителю предвидеть проблемы, которые могут возникнуть на каких-либо этапах создания ПО, и разработать превентивные меры для их предупреждения или решения. План, разработанный на начальном этапе проекта, рассматривается всеми его участниками как руководящий документ, выполнение которого должно привести к успешному завершению

проекта. Этот первоначальный план должен максимально подробно описывать все этапы реализации проекта.

Процесс планирования начинается, исходя из описания системы, с определения проектных ограничений (временные ограничения, возможности наличного персонала, бюджетные ограничения и т.д.). Эти ограничения должны определяться параллельно с оцениванием проектных параметров, таких как структура и размер проекта, а также распределением функций среди исполнителей. Затем определяются этапы разработки и то, какие результаты документация, прототипы, подсистемы или версии программного продукта) должны быть получены по окончании этих этапов. Далее начинается циклическая часть планирования. Сначала разрабатывается график работ по выполнению проекта или дается разрешение на продолжение использования ранее созданного графика. После этого проводится контроль выполнения работ и отмечаются расхождения между реальным и плановым ходом работ.

Далее, по мере поступления новой информации о ходе выполнения проекта, возможен пересмотр первоначальных оценок параметров проекта. Это, в свою очередь, может привести к изменению графика работ. Если в результате этих изменений нарушаются сроки завершения проекта, должны быть пересмотрены (и согласованы с заказчиком ПО) проектные ограничения.

Конечно, большинство руководителей проектов не думают, что реализация их проектов пройдет гладко, без всяких проблем. Желательно описать возможные проблемы еще до того, как они проявят себя в ходе выполнения проекта. Поэтому лучше составлять "пессимистические" графики работ, чем "оптимистические". Но, конечно, невозможно построить план, учитывающий все, в том числе случайные, проблемы и задержки выполнения проекта, поэтому и возникает необходимость периодического пересмотра проектных ограничений и этапов создания программного продукта.

План проекта должен четко показать ресурсы, необходимые для реализации проекта, разделение работ на этапы и временной график выполнения этих этапов. В некоторых организациях план проекта составляется как единый документ, содержащий все виды планов, описанных выше. В других случаях план проекта описывает только технологический процесс создания ПО. В таком плане обязательно присутствуют ссылки на планы других видов, но они разрабатываются отдельно от плана проекта.

Детализация планов проектов очень разнится в зависимости от типа разрабатываемого программного продукта и организации-разработчика. Но в любом случае большинство планов содержат следующие разделы.

1. *Введение*. Краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом.
2. *Организация выполнения проекта*. Описание способа подбора команды разработчиков и распределение обязанностей между членами команды.
3. *Анализ рисков*. Описание возможных проектных рисков, вероятности их проявления и стратегий, направленных на их уменьшение.
4. *Аппаратные и программные ресурсы, необходимые для реализации проекта*. Перечень аппаратных средств и программного обеспечения, необходимого для разработки программного продукта. Если аппаратные средства требуется закупать, приводится их стоимость совместно с графиком закупки и поставки.
5. *Разбиение работ на этапы*. Процесс реализации проекта разбивается на отдельные процессы, определяются этапы выполнения проекта, приводится описание результатов ("выходов") каждого этапа и контрольные отметки.

6. *График работ*. В этом графике отображаются зависимости между отдельными процессами (этапами) разработки ПО, оценки времени их выполнения и распределение членов команды разработчиков по отдельным этапам.

7. *Механизмы мониторинга и контроля за ходом выполнения проекта*. Описываются предоставляемые руководителем отчеты о ходе выполнения работ, сроки их предоставления, а также механизмы мониторинга всего проекта.

План должен регулярно пересматриваться в процессе реализации проекта. Одни части плана, например график работ, изменяются часто, другие более стабильны. Для внесения изменений в план требуется специальная организация документопотока, позволяющая отслеживать эти изменения.

Общие сведения о требованиях к информационным системам

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно чётко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему, называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – разработкой требований.

Требования подразделяются на пользовательские и системные. Пользовательские требования – это описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на неё. Системные требования – это описание особенностей системы (архитектура системы, требования к параметрам оборудования и т.д.), необходимых для эффективной реализации требований пользователя.

Первые шаги по разработке требований к информационным системам - анализ осуществимости

Разработка требований — это процесс, включающий мероприятия, необходимые для создания и утверждения документа, содержащего спецификацию системных требований. Для новых программных систем процесс разработки требований должен начинаться с анализа осуществимости. Началом такого анализа является общее описание системы и ее назначения, а результатом анализа — отчет, в котором должна быть четкая рекомендация, продолжать или нет процесс разработки требований проектируемой системы. Другими словами, анализ осуществимости должен осветить следующие вопросы.

1. Отвечает ли система общим и бизнес-целям организации-заказчика и организации-разработчика?

2. Можно ли реализовать систему, используя существующие на данный момент технологии и не выходя за пределы заданной стоимости?

3. Можно ли объединить систему с другими системами, которые уже эксплуатируются?

Критическим является вопрос, будет ли система соответствовать целям организации. Если система не соответствует этим целям, она не представляет никакой ценности для организации. В то же время многие организации разрабатывают системы, не

соответствующие их целям, либо не совсем ясно понимая эти цели, либо под влиянием политических или общественных факторов.

Выполнение анализа осуществимости включает сбор и анализ информации о будущей системе и написание соответствующего отчета. Сначала следует определить, какая именно информация необходима, чтобы ответить на поставленные выше вопросы. Например, эту информацию можно получить, ответив на следующее:

1. Что произойдет с организацией, если система не будет введена в эксплуатацию?
2. Какие текущие проблемы существуют в организации и как новая система поможет их решить?
3. Каким образом система будет способствовать целям бизнеса?
4. Требует ли разработка системы технологии, которая до этого не использовалась в организации?

Далее необходимо определить источники информации. Это могут быть менеджеры отделов, где система будет использоваться, разработчики программного обеспечения, знакомые с типом будущей системы, технологи, конечные пользователи и т.д.

После обработки собранной информации готовится отчет по анализу осуществимости создания системы. В нем должны быть даны рекомендации относительно продолжения разработки системы. Могут быть предложены изменения бюджета и графика работ по созданию системы или предъявлены более высокие требования к системе.

4. **Порядок выполнения работы**

1. Изучить предлагаемый теоретический материал.
2. Составить подробное описание информационной системы.
3. На основании описания системы провести анализ осуществимости. В ходе анализа ответить на вопросы
   - *Что произойдет с организацией, если система не будет введена в эксплуатацию?*
   - *Какие текущие проблемы существуют в организации и как новая система поможет их решить?*
   - *Каким образом система будет способствовать целям бизнеса?*
   - *Требует ли разработка системы технологии, которая до этого не использовалась в организации?*

   Результатом анализа должно явиться заключение о возможности реализации проекта.
4. Распределить роли в группе (руководитель проекта-разработчик, системный аналитик-разработчик, тестер-разработчик).
5. Заполнить разделы плана:

- *Введение*
- *Организация выполнения проекта*
- *Анализ рисков*

Разделы должны содержать рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.

6. Составить отчет о проделанной работе.

5. Содержание отчета

В отчете следует указать:

1. Цель работы
2. Введение. Краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом
3. Описание информационной системы (ПО)

наличие заключения о возможности реализации проекта, содержащего рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению

4. Анализ осуществимости (согласно требованиям к результатам выполнения лабораторного практикума п.2), указать возможные проблемы и пути их решения.
5. Роли участников группы разработки ПО.
6. Программно-аппаратные средства, используемые при выполнении работы.
7. Заключение (выводы)
8. Список используемой литературы

6. Литература

1. Соммервиль Иан. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 624 с.
2. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. – СПб.:Питер, 2002. – 496 с.
3. Константайн Л., Локвуд Л. Разработка программного обеспечения. – СПб.:Питер, 2004. – 592 с.
4. Иванова Г.С. Технология программирования: Учебник для вузов. - М.: Изд-во

МГТУ им. Н.Э. Баумана, 2002. - 320 с.

Лабораторная работа №2

«Разработка требований к информационной системе»

1. Цель работы:

Составить и проанализировать требования к информационной системе, оформить техническое задание на разработку программного обеспечения.

2. Методические указания

Лабораторная работа направлена на ознакомление с процессом разработки требований к информационной системе и составления технического задания на разработку программного обеспечения, получение навыков по использованию основных методов формирования и анализа требований.

Требования к результатам выполнения лабораторного практикума:

1. наличие диаграммы идентификации точек зрения и диаграммы иерархии точек зрения;
2. наличие пользовательских требований, четко описывающих будущий функционал системы;
3. наличие системных требований, включающих требования к структуре, программному интерфейсу, технологиям разработки, общие требования к системе (надежность, масштабируемость, распределённость, модульность, безопасность, открытость, удобство пользования и т.д.);
4. наличие составленного технического задания.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на странице http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретические сведения

Общие сведения о требованиях к информационным системам

Проблемы, которые приходится решать специалистам в процессе создания программного обеспечения, очень сложны. Природа этих проблем не всегда ясна, особенно если разрабатываемая программная система инновационная. В частности, трудно чётко описать те действия, которые должна выполнять система. Описание функциональных возможностей и ограничений, накладываемых на систему, называется требованиями к этой системе, а сам процесс формирования, анализа, документирования и проверки этих функциональных возможностей и ограничений – разработкой требований.

Требования подразделяются на пользовательские и системные. Пользовательские требования – это описание на естественном языке (плюс поясняющие диаграммы) функций, выполняемых системой, и ограничений, накладываемых на неё. Системные требования – это описание особенностей системы (архитектура системы, требования к параметрам оборудования и т.д.), необходимых для эффективной реализации требований пользователя.

Разработка требований

Разработка требований — это процесс, включающий мероприятия, необходимые для создания и утверждения документа, содержащего спецификацию системных требований. Различают четыре основных этапа процесса разработки требований:

1. анализ технической осуществимости создания системы,
2. формирование и анализ требований,
3. специфицирование требований и создание соответствующей документации,
4. аттестация этих требований.

На рис. 1 показаны взаимосвязи между этими этапами и результаты, сопровождающие каждый этап процесса разработки системных требований.

Рис. 1. Процесс разработки требований

Но поскольку в процессе разработки системы в силу разнообразных причин требования могут меняться, управление требованиями, т.е. процесс управления изменениями системных требований, является необходимой составной частью деятельности по их разработке.

Формирование и анализ требований

Следующим этапом процесса разработки требований является формирование (определение) и анализ требований.

Обобщенная модель процесса формирования и анализа требований показана на рис. 2. Каждая организация использует собственный вариант этой модели, зависящий от “местных факторов”: опыта работы коллектива разработчиков, типа разрабатываемой системы, используемых стандартов и т.д.

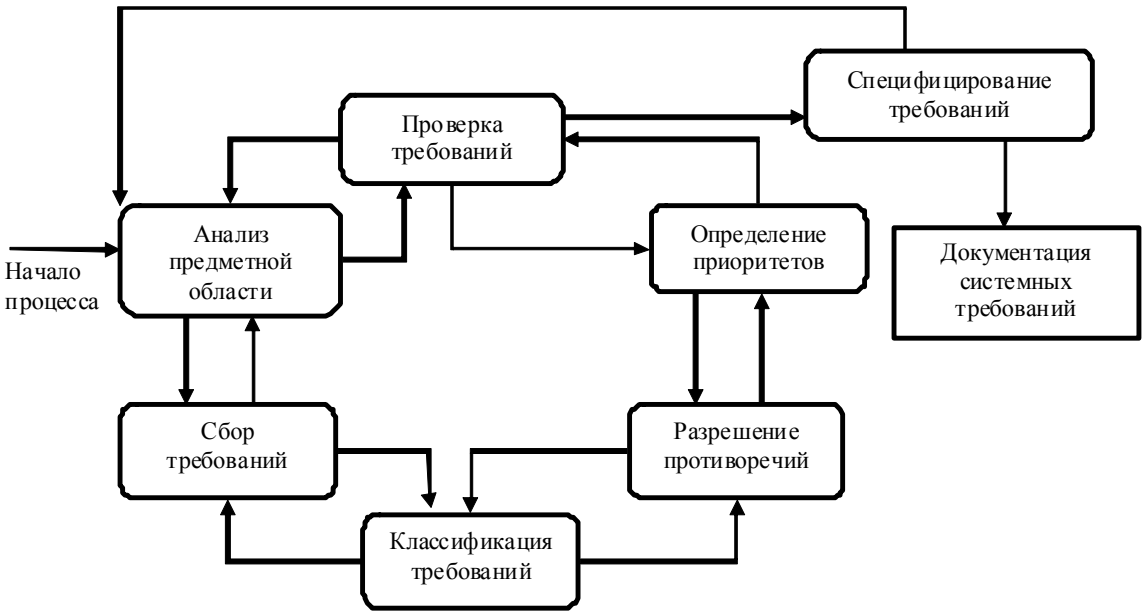


Рис. 2. Процесс формирования и анализа требований

Процесс формирования и анализа требований проходит через ряд этапов.

1. *Анализ предметной области.* Аналитики должны изучить предметную область, где будет эксплуатироваться система.
2. *Сбор требований.* Это процесс взаимодействия с лицами, формирующими требования. Во время этого процесса продолжается анализ предметной области.
3. *Классификация требований.* На этом этапе бесформенный набор требований преобразуется в логически связанные группы требований.
4. *Разрешение противоречий.* Без сомнения, требования многочисленных лиц, занятых в процессе формирования требований, будут противоречивыми. На этом этапе определяются и разрешаются противоречия различного рода.
5. *Назначение приоритетов.* В любом наборе требований одни из них будут более важны, чем другие. На этом этапе совместно с лицами, формирующими требования, определяются наиболее важные требования.
6. *Проверка требований.* На этом этапе определяется их полнота, последовательность и непротиворечивость.

Процесс формирования и анализа требований циклический, с обратной связью от одного этапа к другому. Цикл начинается с анализа предметной области и заканчивается проверкой требований. Понимание требований предметной области увеличивается в каждом цикле процесса формирования требований.

Рассмотрим три основных подхода к формированию требований: метод, основанный на множестве опорных точек зрения, сценарии и этнографический метод.

Опорные точки зрения

Подход с использованием различных *опорных* точек зрения к разработке требований признает различные (опорные) точки зрения на проблему и использует их в качестве

основы построения и организации как процесса формирования требований, так и непосредственно самих требований.

Различные методы предлагают разные трактовки выражения "точка зрения". Точки зрения можно трактовать следующим образом.

1. *Как источник информации о системных данных.* В этом случае на основе опорных точек зрения строится модель создания и использования данных в системе. В процессе формирования требований отбираются все такие точки зрения (и на их основе определяются данные), которые будут созданы или использованы при работе системы, а также способы обработки этих данных.

2. *Как структура представлений.* В этом случае точки зрения рассматриваются как особая часть модели системы. Например, на основе различных точек зрения могут разрабатываться модели "сущность-связь", модели конечного автомата и т.д.

3. *Как получатели системных сервисов.* В этом случае точки зрения являются внешними (относительно системы) получателями системных сервисов. Точки зрения помогают определить данные, необходимые для выполнения системных сервисов или их управления.

Наиболее эффективным подходом к анализу таких систем является использование внешних опорных точек зрения. На основе этого подхода разработан метод VORD (Viewpoint-Oriented Requirements Definition — определение требований на основе точек зрения) для формирования и анализа требований. Основные этапы метода VORD показаны на рис. 3.:

1. Идентификация точек зрения, получающих системные сервисы, и идентификация сервисов, соответствующих каждой точке зрения.

2. Структурирование точек зрения — создание иерархии сгруппированных точек зрения. Общесистемные сервисы предоставляются более высоким уровням иерархии и наследуются точками зрения низшего уровня.

3. Документирование опорных точек зрения, которое заключается в точном описании идентифицированных точек зрения и сервисов.

4. Отображение системы точек зрения, которая показывает системные объекты, определенные на основе информации, заключенной в опорных точках зрения.

Рис. 3. Метод VORD

Пример. Рассмотрим использование метода VORD на первых трех шагах анализа требований для системы системы поддержки заказа и учета товаров в бакалейной лавке. В бакалейной лавке для каждого товара фиксируется место хранения (определенная полка),

количество товара и его поставщик. Система поддержки заказа и учета товаров должна обеспечивать добавление информации о новом товаре, изменение или удаление информации об имеющемся товаре, хранение (добавление, изменение и удаление) информации о поставщиках, включающей в себя название фирмы, ее адрес и телефон. При помощи системы составляются заказы поставщикам. Каждый заказ может содержать несколько позиций, в каждой позиции указываются наименование товара и его количество в заказе. Система по требованию пользователя формирует и выдает на печать следующую справочную информацию:

- список всех товаров;
- список товаров, имеющихся в наличии;
- список товаров, количество которых необходимо пополнить;
- список товаров, поставляемых данным поставщиком.

Первым шагом в формировании требований является идентификация опорных точек зрения. Во всех методах формирования требований, основанных на использовании точек зрения, начальная идентификация является наиболее трудной задачей. Один из подходов к идентификации точек зрения — метод "мозговой атаки", когда определяются потенциальные системные сервисы и организации, взаимодействующие с системой. Организуется встреча лиц, участвующих в формировании требований, которые предлагают свои точки зрения. Эти точки зрения представляются в виде диаграммы, состоящей из ряда круговых областей, отображающих возможные точки зрения (рис. 4). Во время "мозговой атаки" необходимо идентифицировать потенциальные опорные точки зрения, системные сервисы, входные данные, нефункциональные требования, управляющие события и исключительные ситуации.

Следующей стадией процесса формирования требований будет идентификация опорных точек зрения (на рис. 4 показаны в виде темных круговых областей) и сервисов (показаны в виде затененных областей). Сервисы должны соответствовать опорным точкам зрения. Но могут быть сервисы, которые не поставлены им в соответствие. Это означает, что на начальном этапе "мозговой атаки" некоторые опорные точки зрения не были идентифицированы.

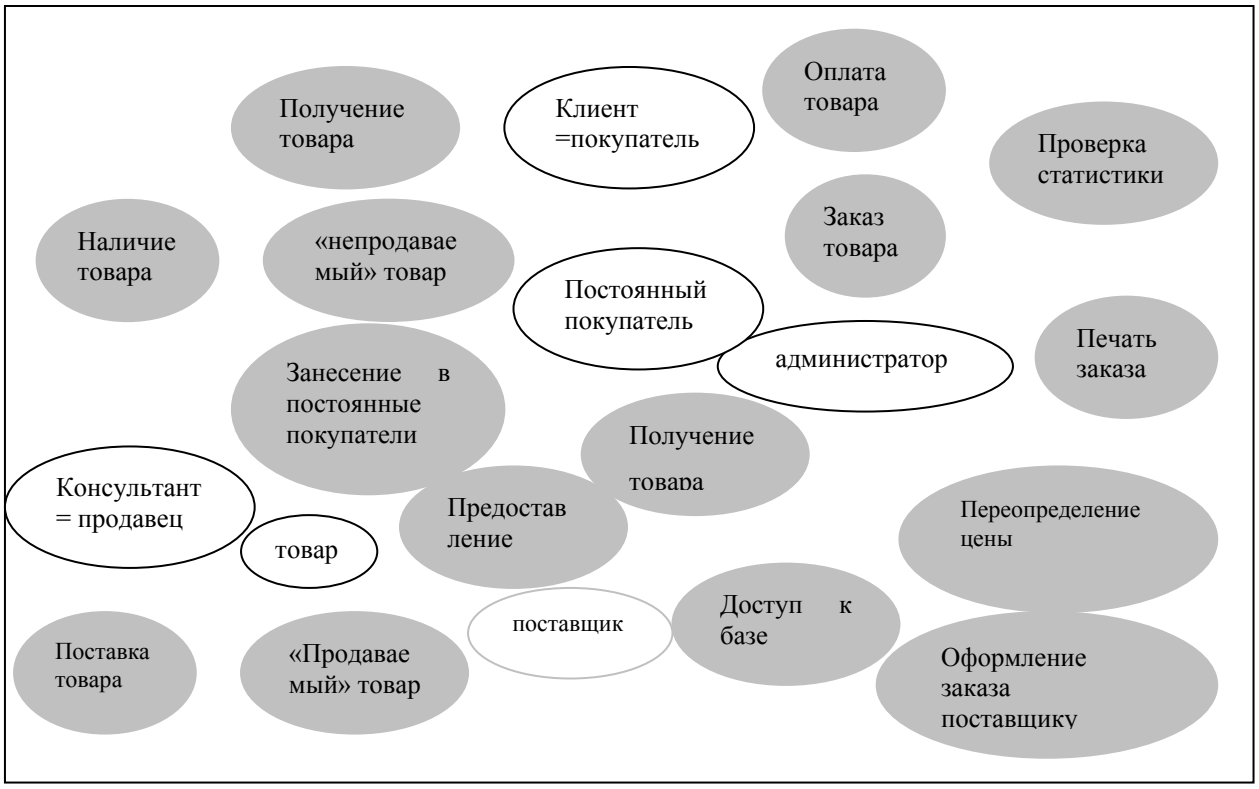


Рис. 4. Диаграмма идентификации точек зрения

В таблице 1 показано распределение сервисов для некоторых идентифицированных на рис. 4 точек зрения. Один и тот же сервис может быть соотнесен с несколькими точками зрения.

Таблица 1. Сервисы, соотнесенные с точками зрения

клиент	**покупатель**	**постоянный покупатель**	**товар**	**поставщик**	**продавец**	**администратор**
Проверка наличия товара	Занесение в список постоянных клиентов	Получение скидки	Прием товара	Занесение в базу данных (название, адрес, телефон и т.д.)	Продажа товара	Доступ к базе данных
Покупка товара		Получение информацию о новых поступлениях	Занесение в базу данных (данные о поставщике, кол-ве, месте хранения и.д.)		Печать чека	Проверка статистики
Получение чека			Назначение цены		Доступ к каталогу	Переопределени е цены
Заказ товара			Переопределен ие цены		Проверка наличия товара	Оформление заказа поставщику
Занесение покупателя и суммы покупки в базу данных			«Покупаемый» или «непокупаемы й» товар		Оформление заказа покупателю	Печать заказа

Информация, извлеченная из точек зрения, используется для заполнения форм шаблонов точек зрения и организации точек зрения в иерархию наследования. Это позволяет увидеть общие точки зрения и повторно использовать информацию в иерархии наследования. Сервисы, данные и управляющая информация наследуются подмножеством точек зрения. На рис. 5 показана часть иерархии точек зрения для системы поддержки заказа и учета товаров.

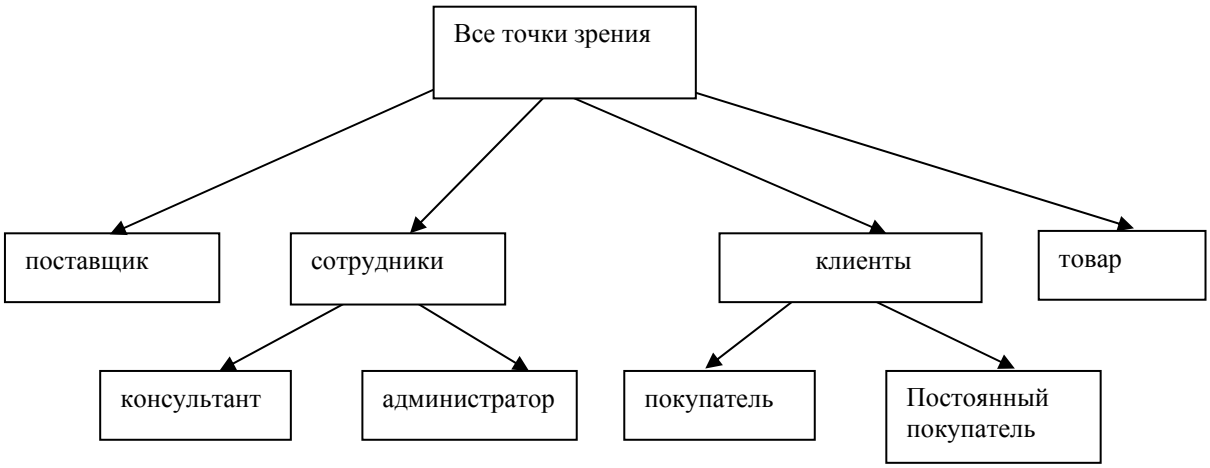


Рис. 5. Иерархия точек зрения

Аттестация требований

Аттестация должна продемонстрировать, что требования действительно определяют ту систему, которую хочет иметь заказчик. Проверка требований важна, так как ошибки в спецификации требований могут привести к переделке системы и большим затратам, если будут обнаружены во время процесса разработки системы или после введения ее в эксплуатацию. Стоимость внесения в систему изменений, необходимых для устранения ошибок в требованиях, намного выше, чем исправление ошибок проектирования или кодирования. Причина в том, что изменение требований обычно влечет за собой значительные изменения в системе, после внесения которых она должна пройти повторное тестирование.

Во время процесса аттестации должны быть выполнены различные типы проверок требований.

1. *Проверка правильности требований.* Пользователь может считать, что система необходима для выполнения некоторых определенных функций. Однако дальнейшие размышления и анализ могут привести к необходимости введения дополнительных или новых функций. Системы предназначены для разных пользователей с различными потребностями, и поэтому набор требований будет представлять собой некоторый компромисс между требованиями пользователей системы.
2. *Проверка на непротиворечивость.* Спецификация требований не должна содержать противоречий. Это означает, что в требованиях не должно быть противоречащих друг другу ограничений или различных описаний одной и той же системной функции.
3. *Проверка на полноту.* Спецификация требований должна содержать требования, которые определяют все системные функции и ограничения, налагаемые на систему.

4. *Проверка на выполнимость.* На основе знания существующих технологий требования должны быть проверены на возможность их реального выполнения. Здесь также проверяются возможности финансирования и график разработки системы.

Существует ряд методов аттестации требований, которые можно использовать совместно или каждый в отдельности.

1. *Обзор требований.* Требования системно анализируются рецензентами.
2. *Прототипирование.* На этом этапе прототип системы демонстрируется конечным пользователям и заказчику. Они могут экспериментировать с этим прототипом, чтобы убедиться, что он отвечает их потребностям.
3. *Генерация тестовых сценариев.* В идеале требования должны быть такими, чтобы их реализацию можно было протестировать. Если тесты для требований разрабатываются как часть процесса аттестации, то часто это позволяет обнаружить проблемы в спецификации. Если такие тесты сложно или невозможно разработать, то обычно это означает, что требования трудно выполнить и поэтому необходимо их пересмотреть.
4. *Автоматизированный анализ непротиворечивости.* Если требования представлены в виде структурных или формальных системных моделей, можно использовать инструментальные CASE-средства для проверки непротиворечивости моделей. Для автоматизированной проверки непротиворечивости необходимо построить базу данных требований и затем проверить все требования в этой базе данных. Анализатор требований готовит отчет обо всех обнаруженных противоречиях.

Пользовательские и системные требования

На основании полученных моделей строятся пользовательские требования, т.е. как было сказано в начале описание на естественном языке функции, выполняемых системой, и ограничений, накладываемых на неё.

Пользовательские требования должны описывать внешнее поведение системы, основные функции и сервисы предоставляемые системой, её нефункциональные свойства. Необходимо выделить опорные точки зрения и сгруппировать требования в соответствии с ними. Пользовательские требования можно оформить как простым перечислением, так и используя нотацию вариантов использования.

Далее составляются системные требования. Они включат в себя:

1. Требования к архитектуре системы. Например, число и размещение хранилищ и серверов приложений.
2. Требования к параметрам оборудования. Например, частота процессоров серверов и клиентов, объём хранилищ, размер оперативной и видео памяти, пропускная способность канала и т.д.
3. Требования к параметрам системы. Например, время отклика на действие пользователя, максимальный размер передаваемого файла, максимальная скорость передачи данных, максимальное число одновременно работающих пользователей и т.д.
4. Требования к программному интерфейсу.

5. Требования к структуре системы. Например, Масштабируемость, распределённость, модульность, открытость.

   - масштабируемость – возможность распространения системы на большое количество машин, не приводящая к потере работоспособности и эффективности, при этом способность системы наращивать свою мощность должна определяться только мощностью соответствующего аппаратного обеспечения.
   - распределенность - система должна поддерживать распределённое хранение данных.
   - модульность - система должна состоять из отдельных модулей, интегрированных между собой.
   - открытость - наличие открытых интерфейсов для возможной доработки и интеграции с другими системами.

6. Требования по взаимодействию и интеграции с другими системами. Например, использование общей базы данных, возможность получения данных из баз данных определённых систем и т.д.

4. Порядок выполнения работы

1. Изучить предлагаемый теоретический материал.
2. Построить опорные точки зрения на основании метода VORD для формирования и анализа требований. Результатом должны явиться две диаграммы: диаграмма идентификации точек зрения и диаграмма иерархии точек зрения.
3. Составить информационную модель будущей системы, включающую в себя описание основных объектов системы и взаимодействия между ними. На основании полученной информационной модели и диаграмм идентификации точек зрения, диаграмма иерархии точек зрения сформировать требования пользователя и системные требования.
4. Провести аттестацию требований, указать какие типы проверок выбрали.
5. На основании описания системы (Лабораторная работа №1), информационной модели, пользовательских и системных требований составить техническое задание на создание программного обеспечения (пример см. Приложение Б). ТЗ должно содержать основные разделы, описанные в ГОСТ 34.602-89 (см. Приложение А).
6. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

5. Содержание отчета

В отчете следует указать:

1. Цель работы
2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума (п.2).

5. Заключение (выводы)
6. Список используемой литературы

6. Литература

1. Соммервиль Иан. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 624 с.
2. Якобсон А., Буч Г., Рамбо Дж. Унифицированный процесс разработки программного обеспечения. – СПб.:Питер, 2002. – 496 с.
3. Константайн Л., Локвуд Л. Разработка программного обеспечения. – СПб.:Питер, 2004. – 592 с.
4. Иванова Г.С. Технология программирования: Учебник для вузов. - М.: Изд-во МГТУ им. Н.Э. Баумана, 2002. - 320 с.
5. ГОСТ 34.602-89 Техническое задание на создание автоматизированной системы
6. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению

7. Контрольные вопросы

1. Предложите, кто бы мог участвовать в формировании требований для университетской системы регистрации студентов. Объясните, почему почти неизбежно, что требования, сформулированные разными лицами, будут противоречивы.
2. Разрабатывается система ПО для автоматизации библиотечного каталога. Эта система будет содержать информацию относительно всех книг в библиотеке и будет полезна библиотечному персоналу, абонентам и читателям. Система должна иметь средства просмотра каталога, средства создания запросов и средства, позволяющие пользователям резервировать книги, находящиеся в данный момент на руках. Определите основные опорные точки зрения, которые необходимо учесть в спецификации системы, и покажите их взаимоотношения, используя диаграмму иерархии точек зрения.
3. Для трех точек зрения, определенных в системе библиотечного каталога, укажите сервисы и соответствующие данные, которые обеспечиваются этими точками зрения, и события, которые управляют этими сервисами.
4. Кто должен проводить обзор требований? Нарисуйте модель процесса обзора требований.
5. Ваша компания использует стандартный метод анализа требований. В процессе работы вы обнаружили, что этот метод не учитывает социальные факторы, важные для системы, которую вы анализируете. Ваш руководитель дал вам ясно понять, какому методу анализа нужно следовать. Обсудите, что вы должны делать в такой ситуации.

Приложение А

УДК 668.012.011.56:006.354 **Группа П87**

ГОСУДАРСТВЕННЫЙ СТАНДАРТ СОЮЗА ССР

ИНФОРМАЦИОННАЯ ТЕХНОЛОГИЯ.
Комплекс стандартов на автоматизированные системы

ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА СОЗДАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ

34.602-89

Information technology. Set of standards for automated systems.
Technical directions for developing of automated system

ОКСТУ 0034

Дата введения с 01.01.1990г.

Настоящий стандарт распространяется на автоматизированные системы (АС) для автоматизации различных видов деятельности (управление, проектирование, исследование и т. п.), включая их сочетания, и устанавливает состав, содержание, правила оформления документа «Техническое задание на создание (развитие или модернизацию) системы» (далее - ТЗ на АС).

Рекомендуемый порядок разработки, согласования и утверждения ТЗ на АС приведен в приложении 1.

1. ОБЩИЕ ПОЛОЖЕНИЯ

1.1. ТЗ на АС является основным документом, определяющим требования и порядок создания (развития или модернизации - далее создания) автоматизированной системы, в соответствии с которым проводится разработка АС и ее приемка при вводе в действие.

1.2. ТЗ на АС разрабатывают на систему в целом, предназначенную для работы самостоятельно или в составе другой системы.

Дополнительно могут быть разработаны ТЗ на части АС:

- на подсистемы АС, комплексы задач АС и т. п. в соответствии с требованиями настоящего стандарта;
- на комплектующие средства технического обеспечения и программно-технические комплексы в соответствии со стандартами ЕСКД и СРПП;
- на программные средства в соответствии со стандартами ЕСПД;
- на информационные изделия в соответствии с ГОСТ 19.201 и НТД, действующей в ведомстве заказчика АС.

Примечание. В ТЗ на АСУ для группы взаимосвязанных объектов следует включать только общие для группы объектов требования. Специфические требования отдельного объекта управления следует отражать в ТЗ на АСУ этого объекта.

1.3. Требования к АС в объеме, установленном настоящим стандартом, могут быть включены в задание на проектирование вновь создаваемого объекта автоматизации. В этом случае ТЗ на АС не разрабатывают.

1.4. Включаемые в ТЗ на АС требования должны соответствовать современному уровню развития науки и техники и не уступать аналогичным требованиям, предъявляемым к лучшим современным отечественным и зарубежным аналогам. Задаваемые в ТЗ на АС требования не должны ограничивать разработчика системы в поиске и реализации наиболее эффективных технических, технико-экономических и других решений.

1.5. ТЗ на АС разрабатывают на основании исходных данных в том числе содержащихся в итоговой документации стадии «Исследование и обоснование создания АС», установленной ГОСТ 24.601.

1.6. В ТЗ на АС включают только те требования, которые дополняют требования к системам данного вида (АСУ, САПР, АСНИ и т. д.), содержащиеся в действующих НТД, и определяются спецификой конкретного объекта, для которого создается система.

1.7. Изменения к ТЗ на АС оформляют дополнением или подписанным заказчиком и разработчиком протоколом. Дополнение или указанный протокол являются неотъемлемой частью ТЗ на АС. На титульном листе ТЗ на АС должна быть запись «Действует с ... ».

2. СОСТАВ И СОДЕРЖАНИЕ

2.1. ТЗ на АС содержит следующие разделы, которые могут быть разделены на подразделы:

- 1) общие сведения;
- 2) назначение и цели создания (развития) системы;
- 3) характеристика объектов автоматизации;
- 4) требования к системе;
- 5) состав и содержание работ по созданию системы;
- 6) порядок контроля и приемки системы;
- 7) требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие;
- 8) требования к документированию;

- 9) источники разработки.

В ТЗ на АС могут включаться приложения.

2.2. В зависимости от вида, назначения, специфических особенностей объекта автоматизации и условий функционирования системы допускается оформлять разделы ТЗ в виде приложений, вводить дополнительные, исключать или объединять подразделы ТЗ.

В ТЗ на части системы не включают разделы, дублирующие содержание разделов ТЗ на АС в целом.

2.3. В разделе «Общие сведения» указывают:

- 1) полное наименование системы и ее условное обозначение;
- 2) шифр темы или шифр (номер) договора;
- 3) наименование предприятий (объединений) разработчика и заказчика (пользователя) системы и их реквизиты;
- 4) перечень документов, на основании которых создается система, кем и когда утверждены эти документы;
- 5) плановые сроки начала и окончания работы по созданию системы;
- 6) сведения об источниках и порядке финансирования работ;
- 7) порядок оформления и предъявления заказчику результатов работ по созданию системы (ее частей), по изготовлению и наладке отдельных средств (технических, программных, информационных) и программно-технических (программно-методических) комплексов системы.

2.4. Раздел «Назначение и цели создания (развития) системы» состоит из подразделов:

- 1) назначение системы;
- 2) цели создания системы.

2.4.1. В подразделе «Назначение системы» указывают вид автоматизируемой деятельности (управление, проектирование и т. п.) и перечень объектов автоматизации (объектов), на которых предполагается ее использовать.

Для АСУ дополнительно указывают перечень автоматизируемых органов (пунктов) управления и управляемых объектов.

2.4.2. В подразделе «Цели создания системы» приводят наименования и требуемые значения технических, технологических, производственно-экономических или других показателей объекта автоматизации, которые должны быть достигнуты в результате создания АС, и указывают критерии оценки достижения целей создания системы.

2.5. В разделе «Характеристики объекта автоматизации» приводят:

- 1) краткие сведения об объекте автоматизации или ссылки на документы, содержащие такую информацию;
- 2) сведения об условиях эксплуатации объекта автоматизации и характеристиках окружающей среды.

Примечание: Для САПР в разделе дополнительно приводят основные параметры и характеристики объектов проектирования.

2.6. Раздел «Требования к системе» состоит из следующих подразделов:

- 1) требования к системе в целом;
- 2) требования к функциям (задачам), выполняемым системой;
- 3) требования к видам обеспечения.

Состав требований к системе, включаемых в данный раздел ТЗ на АС, устанавливают в зависимости от вида, назначения, специфических особенностей и условий функционирования конкретной системы. В каждом подразделе приводят ссылки на действующие НТД, определяющие требования к системам соответствующего вида.

2.6.1. В подразделе «Требования к системе в целом» указывают:

- требования к структуре и функционированию системы;
- требования к численности и квалификации персонала системы и режиму его работы;
- показатели назначения;
- требования к надежности;
- требования безопасности;
- требования к эргономике и технической эстетике;
- требования к транспортабельности для подвижных АС;
- требования к эксплуатации, техническому обслуживанию, ремонту и хранению компонентов системы;
- требования к защите информации от несанкционированного доступа;
- требования по сохранности информации при авариях;
- требования к защите от влияния внешних воздействий;
- требования к патентной чистоте;
- требования по стандартизации и унификации;
- дополнительные требования.

2.6.1.1. В требованиях к структуре и функционированию системы приводят:

- 1) перечень подсистем, их назначение и основные характеристики, требования к числу уровней иерархии и степени централизации системы;
- 2) требования к способам и средствам связи для информационного обмена между компонентами системы;
- 3) требования к характеристикам взаимосвязей создаваемой системы со смежными системами, требования к ее совместимости, в том числе указания о способах обмена информацией (автоматически, пересылкой документов, по телефону и т. п.);
- 4) требования к режимам функционирования системы;
- 5) требования по диагностированию системы;
- 6) перспективы развития, модернизации системы.

2.6.1.2. В требованиях к численности и квалификации персонала на АС приводят:

- требования к численности персонала (пользователей) АС;

- требования к квалификации персонала, порядку его подготовки и контроля знаний и навыков;
- требуемый режим работы персонала АС.

2.6.1.3. В требованиях к показателям назначения АС приводят значения параметров, характеризующие степень соответствия системы ее назначению.

Для АСУ указывают:

- степень приспособляемости системы к изменению процессов и методов управления, к отклонениям параметров объекта управления;
- допустимые пределы модернизации и развития системы;
- вероятностно-временные характеристики, при которых сохраняется целевое назначение системы.

2.6.1.4. В требования к надежности включают:

- 1) состав и количественные значения показателей надежности для системы в целом или ее подсистем;
- 2) перечень аварийных ситуаций, по которым должны быть регламентированы требования к надежности, и значения соответствующих показателей;
- 3) требования к надежности технических средств и программного обеспечения;
- 4) требования к методам оценки и контроля показателей надежности на разных стадиях создания системы в соответствии с действующими нормативно-техническими документами.

2.6.1.5. В требования по безопасности включают требования по обеспечению безопасности при монтаже, наладке, эксплуатации, обслуживании и ремонте технических средств системы (защита от воздействий электрического тока, электромагнитных полей, акустических шумов и т. п.), по допустимым уровням освещенности, вибрационных и шумовых нагрузок.

2.6.1.6. В требования по эргономике и технической эстетике включают показатели АС, задающие необходимое качество взаимодействия человека с машиной и комфортность условий работы персонала.

2.6.1.7. Для подвижных АС в требования к транспортабельности включают конструктивные требования, обеспечивающие транспортабельность технических средств системы, а также требования к транспортным средствам.

2.6.1.8. В требования к эксплуатации, техническому обслуживанию, ремонту и хранению включают:

- 1) условия и регламент (режим) эксплуатации, которые должны обеспечивать использование технических средств (ТС) системы с заданными техническими показателями, в том числе виды и периодичность обслуживания ТС системы или допустимость работы без обслуживания;
- 2) предварительные требования к допустимым площадям для размещения персонала и ТС системы, к параметрам сетей энергоснабжения и т. п.;
- 3) требования по количеству, квалификации обслуживающего персонала и режимам его работы;

- 4) требования к составу, размещению и условиям хранения комплекта запасных изделий и приборов;
- 5) требования к регламенту обслуживания.

2.6.9. В требования к защите информации от несанкционированного доступа включают требования, установленные в НТД, действующей в отрасли (ведомстве) заказчика.

2.6.1.10. В требованиях по сохранности информации приводят перечень событий: аварий, отказов технических средств (в том числе - потеря питания) и т. п., при которых должна быть обеспечена сохранность информации в системе.

2.6.1.11. В требованиях к средствам защиты от внешних воздействий приводят:

- 1) требования к радиоэлектронной защите средств АС;
- 2) требования по стойкости, устойчивости и прочности к внешним воздействиям (среде применения).

2.6.1.12. В требованиях по патентной чистоте указывают перечень стран, в отношении которых должна быть обеспечена патентная чистота системы и ее частей.

2.6.1.13. В требования к стандартизации и унификации включают: показатели, устанавливающие требуемую степень использования стандартных, унифицированных методов реализации функций (задач) системы, поставляемых программных средств, типовых математических методов и моделей, типовых проектных решений, унифицированных форм управленческих документов, установленных ГОСТ 6.10.1, общесоюзных классификаторов технико-экономической информации и классификаторов других категорий в соответствии с областью их применения, требования к использованию типовых автоматизированных рабочих мест, компонентов и комплексов.

2.6.1.14. В дополнительные требования включают:

- 1) требования к оснащению системы устройствами для обучения персонала (тренажерами, другими устройствами аналогичного назначения) и документацией на них;
- 2) требования к сервисной аппаратуре, стендам для проверки элементов системы;
- 3) требования к системе, связанные с особыми условиями эксплуатации;
- 4) специальные требования по усмотрению разработчика или заказчика системы.

2.6.2. В подразделе «Требование к функциям (задачам)», выполняемым системой, приводят:

 1) по каждой подсистеме перечень функций, задач или их комплексов (в том числе обеспечивающих взаимодействие частей системы), подлежащих автоматизации;

при создании системы в две или более очереди - перечень функциональных подсистем, отдельных функций или задач, вводимых в действие в 1-й и последующих очередях;

 2) временной регламент реализации каждой функции, задачи (или комплекса задач);

 3) требования к качеству реализации каждой функции (задачи или комплекса задач), к форме представления выходной информации, характеристики необходимой точности и времени выполнения, требования одновременности выполнения группы функций, достоверности выдачи результатов;

 4) перечень и критерии отказов для каждой функции, по которой задаются требования по надежности.

2.6.3. В подразделе «Требования к видам обеспечения» в зависимости от вида системы приводят требования к математическому, информационному, лингвистическому, программному, техническому, метрологическому, организационному, методическому и другие видам обеспечения системы.

2.6.3.1. Для математического обеспечения системы приводят требования к составу, области применения (ограничения) и способам, использования в системе математических методов и моделей, типовых алгоритмов и алгоритмов, подлежащих разработке.

2.6.3.2. Для информационного обеспечения системы приводят требования:

- 1) к составу, структуре и способам организации данных в системе;
- 2) к информационному обмену между компонентами системы;
- 3) к информационной совместимости со смежными системами;
- 4) по использованию общесоюзных и зарегистрированных республиканских, отраслевых классификаторов, унифицированных документов и классификаторов, действующих на данном предприятии;
- 5) по применению систем управления базами данных;
- 6) к структуре процесса сбора, обработки, передачи данных в системе и представлению данных;
- 7) к защите данных от разрушений при авариях и сбоях в электропитании системы;
- 8) к контролю, хранению, обновлению и восстановлению данных;
- 9) к процедуре придания юридической силы документам, продуцируемым техническими средствами АС (в соответствии с ГОСТ 6.10.4).

2.6.3.3. Для лингвистического обеспечения системы приводят требования к применению в системе языков программирования высокого уровня, языков взаимодействия пользователей и технических средств системы, а также требования к кодированию и декодированию данных, к языкам ввода-вывода данных, языкам манипулирования данными, средствам описания предметной области (объекта автоматизации), к способам организации диалога.

2.6.3.4. Для программного обеспечения системы приводят перечень покупных программных средств, а также требования:

- 1) к независимости программных средств от используемых СВТ и операционной среды;
- 2) к качеству программных средств, а также к способам его обеспечения и контроля;
- 3) по необходимости согласования вновь разрабатываемых программных средств с фондом алгоритмов и программ.

2.6.3.5. Для технического обеспечения системы приводят требования:

- 1) к видам технических средств, в том числе к видам комплексов технических средств, программно-технических комплексов и других комплектующих изделий, допустимых к использованию в системе;
- 2) к функциональным, конструктивным и эксплуатационным характеристикам средств технического обеспечения системы.

2.6.3.6. В требованиях к метрологическому обеспечению приводят:

- 1) предварительный перечень измерительных каналов;
- 2) требования к точности измерений параметров и (или) к метрологическим характеристикам измерительных каналов;
- 3) требования к метрологической совместимости технических средств системы;
- 4) перечень управляющих и вычислительных каналов системы, для которых необходимо оценивать точностные характеристики;
- 5) требования к метрологическому обеспечению технических и программных средств, входящих в состав измерительных каналов системы, средств, встроенного контроля, метрологической пригодности измерительных каналов и средств измерений, используемых при наладке и испытаниях системы;
- 6) вид метрологической аттестации (государственная или ведомственная) с указанием порядка ее выполнения и организаций, проводящих аттестацию.

2.6.3.7. Для организационного обеспечения приводят требования:

 1) к структуре и функциям подразделений, участвующих в функционировании системы или обеспечивающих эксплуатацию;

 2) к организации функционирования системы и порядку взаимодействия персонала АС и персонала объекта автоматизации;

 3) к защите от ошибочных действий персонала системы.

2.6.3.8. Для методического обеспечения САПР приводят требования к составу нормативно-технической документации системы (перечень применяемых при ее функционировании стандартов, нормативов, методик и т. п.).

2.7. Раздел «Состав и содержание работ по созданию (развитию) системы» должен содержать перечень стадий и этапов работ по созданию системы в соответствии с ГОСТ 24.601, сроки их выполнения, перечень организаций - исполнителей работ, ссылки на документы, подтверждающие согласие этих организаций на участие в создании системы, или запись, определяющую ответственного (заказчик или разработчик) за проведение этих работ.

В данном разделе также приводят:

- 1) перечень документов, по ГОСТ 34.201-89, предъявляемых по окончании соответствующих стадий и этапов работ;
- 2) вид и порядок проведения экспертизы технической документации (стадия, этап, объем проверяемой документации, организация-эксперт);
- 3) программу работ, направленных на обеспечение требуемого уровня надежности разрабатываемой системы (при необходимости);

- 4) перечень работ по метрологическому обеспечению на всех стадиях создания системы с указанием их сроков выполнения и организаций-исполнителей (при необходимости).

2.8. В разделе «Порядок контроля и приемки системы» указывают:

- 1) виды, состав, объем и методы испытаний системы и ее составных частей (виды испытаний в соответствии с действующими нормами, распространяющимися на разрабатываемую систему);
- 2) общие требования к приемке работ по стадиям (перечень участвующих предприятий и организаций, место и сроки проведения), порядок согласования и утверждения приемочной документации;
- 3) статус приемочной комиссии (государственная, межведомственная, ведомственная).

2.9. В разделе «Требования к составу и содержанию работ по подготовке объекта автоматизации к вводу системы в действие» необходимо привести перечень основных мероприятий и их исполнителей, которые следует выполнить при подготовке объекта автоматизации к вводу АС в действие.

В перечень основных мероприятий включают:

- 1) приведение поступающей в систему информации (в соответствии с требованиями к информационному и лингвистическому обеспечению) к виду, пригодному для обработки с помощью ЭВМ;
- 2) изменения, которые необходимо осуществить в объекте автоматизации;
- 3) создание условий функционирования объекта автоматизации, при которых гарантируется соответствие создаваемой системы требованиям, содержащимся в ТЗ;
- 4) создание необходимых для функционирования системы подразделений и служб;
- 5) сроки и порядок комплектования штатов и обучения персонала.

Например, для АСУ приводят:

- изменения применяемых методов управления;
- создание условий для работы компонентов АСУ, при которых гарантируется соответствие системы требованиям, содержащимся в ТЗ.

2.10. В разделе «Требования к документированию» приводят:

- 1) согласованный разработчиком и Заказчиком системы перечень подлежащих разработке комплектов и видов документов, соответствующих требованиям ГОСТ 34.201-89 и НТД отрасли заказчика; перечень документов, выпускаемых на машинных носителях; требования к микрофильмированию документации;
- 2) требования по документированию комплектующих элементов межотраслевого применения в соответствии с требованиями ЕСКД и ЕСПД;
- 3) при отсутствии государственных стандартов, определяющих требования к документированию элементов системы, дополнительно включают требования к составу и содержанию таких документов.

2.11. В разделе «Источники разработки» должны быть перечислены документы и информационные материалы (технико-экономическое обоснование, отчеты о законченных научно-исследовательских работах, информационные материалы на отечественные, зарубежные системы-аналоги и др.), на основании которых разрабатывалось ТЗ и которые должны быть использованы при создании системы.

2.12. В состав ТЗ на АС при наличии утвержденных методик включают приложения, содержащие:

- 1) расчет ожидаемой эффективности системы;
- 2) оценку научно-технического уровня системы.

Приложения включают в состав ТЗ на АС по согласованию между разработчиком и заказчиком системы.

3. ПРАВИЛА ОФОРМЛЕНИЯ

3.1. Разделы и подразделы ТЗ на АС должны быть размещены в порядке, установленном в разд. 2 настоящего стандарта.

3.2. ТЗ на АС оформляют в соответствии с требованиями ГОСТ 2.105 на листах формата А4 по ГОСТ 2.301 без рамки, основной надписи и дополнительных граф к ней.

Номера листов (страниц) проставляют, начиная с первого листа, следующего за титульным листом, в верхней части листа (над текстом, посередине) после обозначения кода ТЗ на АС.

3.3. Значения показателей, норм и требований указывают, как правило, с предельными отклонениями или максимальным и минимальным значениями. Если эти показатели, нормы, требования однозначно регламентированы НТД, в ТЗ на АС следует приводить ссылку на эти документы или их разделы, а также дополнительные требования, учитывающие особенности создаваемой системы. Если конкретные значения показателей, норм и требований не могут быть установлены в процессе разработки ТЗ на АС, в нем следует сделать запись о порядке установления и согласования этих показателей, норм и требований:

«Окончательное требование (значение) уточняется в процессе ...

и согласовывается протоколом с ... на стадии ...».

При этом в текст ТЗ на АС изменений не вносят.

3.4. На титульном листе помещают подписи заказчика, разработчика и согласующих организаций, которые скрепляют гербовой печатью. При необходимости титульный лист оформляют на нескольких страницах. Подписи разработчиков ТЗ на АС и должностных лиц, участвующих в согласовании и рассмотрении проекта ТЗ на АС, помещают на последнем листе.

Форма титульного листа ТЗ на АС приведена в приложении 2. Форма последнего листа ТЗ на АС приведена в приложении 3.

3.5. При необходимости на титульном листе ТЗ на АС допускается помещать установленные в отрасли коды, например: гриф секретности, код работы, регистрационный номер ТЗ и др.

3.6. Титульный лист дополнения к ТЗ на АС оформляют аналогично титульному листу технического задания. Вместо наименования «Техническое задание» пишут «Дополнение № ... к ТЗ на АС ... ».

3.7. На последующих листах дополнения к ТЗ на АС помещают основание для изменения, содержание изменения и ссылки на документы, в соответствии с которыми вносятся эти изменения.

3.8. При изложении текста дополнения к ТЗ следует указывать номера соответствующих пунктов, подпунктов, таблиц основного ТЗ на АС и т. п. и применять слова: «заменить», «дополнить», «исключить», «изложить в новой редакции».

ПРИЛОЖЕНИЕ 1
Рекомендуемое

ПОРЯДОК РАЗРАБОТКИ, СОГЛАСОВАНИЯ И УТВЕРЖДЕНИЯ ТЗ НА АС

1. Проект ТЗ на АС разрабатывает организация-разработчик системы с участием заказчика на основании технических требований (заявки, тактико-технического задания и т. п.).

При конкурсной организации работ варианты проекта ТЗ на АС рассматриваются заказчиком, который - либо выбирает предпочтительный, вариант, либо на основании сопоставительного анализа подготавливает с участием будущего разработчика АС окончательный вариант ТЗ на АС.

2. Необходимость согласования проекта ТЗ на АС с органами государственного надзора и другими заинтересованными организациями определяют совместно заказчик системы и разработчик проекта ТЗ на АС,

Работу по согласованию проекта ТЗ на АС осуществляют совместно разработчик ТЗ на АС и заказчик системы, каждый в организациях своего министерства (ведомства).

3. Срок согласования проекта ТЗ на АС в каждой организации не должен превышать 15 дней со дня его получения. Рекомендуется рассылать на согласование экземпляры проекта ТЗ на АС (копий) одновременно во все организации (подразделения).

4. Замечания по проекту ТЗ на АС должны быть представлены с техническим обоснованием. Решения по замечаниям должны быть приняты разработчиком проекта ТЗ на АС и заказчиком системы до утверждения ТЗ на АС.

5. Если при согласовании проекта ТЗ на АС возникли разногласия между разработчиком и заказчиком (или другими заинтересованными организациями), то

составляется протокол разногласий (форма произвольная) и конкретное решение принимается в установленном порядке.

6. Согласование проекта ТЗ на АС разрешается оформлять отдельным документом (письмом). В этом случае под грифом «Согласовано» делают ссылку на этот документ.

7. Утверждение ТЗ на АС осуществляют руководители предприятий (организаций) разработчика и заказчика системы.

8. ТЗ на АС (дополнение к ТЗ) до передачи его на утверждение должно быть проверено службой нормоконтроля организации - разработчика ТЗ и, при необходимости, подвергнуто метрологической экспертизе.

9. Копии, утвержденного ТЗ на АС в 10-дневный срок после утверждения высылаются разработчиком ТЗ на АС участникам создания системы.

10. Согласование и утверждение дополнений к ТЗ на АС проводят в порядке, установленном для ТЗ на АС.

11. Изменения к ТЗ на АС не допускается утверждать после представления системы или ее очереди на приемо-сдаточные испытания.

12. Регистрация, учет и хранение ТЗ на АС и дополнений к нему проводят в соответствии, с требованиями ГОСТ 2.501.

ПРИЛОЖЕНИЕ 2
Рекомендуемое

ФОРМА ТИТУЛЬНОГО ЛИСТА ТЗ НА АС

_____

.

наименование организации - разработчика ТЗ на АС

УТВЕРЖДАЮ

Руководитель (должность, наименование предприятия - заказчика АС)

Личная подпись Расшифровка подписи

Печать

Дата

УТВЕРЖДАЮ

Руководитель (должность, наименование предприятия - разработчик" АС)

Личная подпись Расшифровка подписи

Печать

Дата

_____

наименование вида АС

_____

наименование объекта автоматизации

_____

сокращенное наименование АС

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

На ____ листах

Действует с

СОГЛАСОВАНО

Руководитель (должность, наименование согласующей организации)

Личная подпись Расшифровка подписи

Печать

Дата

ПРИЛОЖЕНИЕ 3
Рекомендуемое

ФОРМА ПОСЛЕДНЕГО ЛИСТА ТЗ НА АС

(код ТЗ)

СОСТАВИЛИ

Наименование организации, предприятия	Должность исполнителя	Фамилия имя, отчество	Подпись	Дата

СОГЛАСОВАНО

Наименование организации, предприятия	Должность исполнителя	Фамилия имя, отчество	Подпись	Дата

ПРИЛОЖЕНИЕ 4
Справочное

ПОЛОЖЕНИЯ ПО СОЗДАНИЮ ЕДИНОГО КОМПЛЕКСА СТАНДАРТОВ АВТОМАТИЗИРОВАННЫХ СИСТЕМ

1. Исходные предпосылки создания комплекса

1.1. Создание и внедрение автоматизированных систем различных классов и назначений ведется во многих отраслях промышленности по нормативно-технической документации, устанавливающей разнообразные организационно-методические и технические нормы, правила и положения, затрудняющие интеграцию систем и эффективное их совместное функционирование.

1.2. В период принятия Госстандартом СССР решения о совершенствовании межотраслевых комплексов стандартов действовали следующие комплексы и системы стандартов, устанавливающие требования к различным видам АС:

- 1) единая система стандартов автоматизированных систем управления (24-я система), распространяющаяся на АСУ, АСУП, АСУ ТП и другие организационно-экономические системы;
- 2) комплекс стандартов (система 23501); распространяющихся на системы автоматизированного проектирования;
- 3) четвертая группа 14-й системы стандартов, распространяющаяся на автоматизированные системы технологической подготовки производства.

1.3. Практика применения стандартов на АСУ, САПР, АСУ ТП, АСТПП показала, что в них применяется одинаковый понятийный аппарат, имеется много общих объектов стандартизации, однако требования стандартов не согласованы между собой, имеются различия по составу и содержанию работ, различия по обозначению, составу, содержанию и оформлению документов и пр.

1.4. На фоне отсутствия единой технической политики в области создания АС многообразие стандартов не обеспечивало широкой совместимости АС при их взаимодействии, не позволяло тиражировать системы, тормозило развитие перспективных направлений использования средств вычислительной техники.

1.5. В настоящее время осуществляется переход к созданию сложных АС (за рубежом системы CAD - CAM), включающих в свой состав АСУ технологическими процессами и производствами, САПР - конструктора, САПР - технолога, АСНИ и др. системы. Использование противоречивых правил при создании таких систем приводит к

снижению качества, увеличению стоимости работ, затягиванию сроков ввода АС в действие.

1.6. Единый комплекс стандартов и руководящих документов должен распространяться на автоматизированные системы различного назначения: АСНИ, САПР, ОАСУ, АСУП, АСУТП, АСУГПС, АСК, АСТПП, включая их интеграцию.

1.7. При разработке межотраслевых документов следует учитывать следующие особенности АС как объектов стандартизации:

- 1) техническое задание является основным документом, в соответствии с которым проводят создание АС и приемку его заказчиком;
- 2) АС, как правило, создают проектным путем с комплектацией изделиями серийного и единичного производства и проведением строительных, монтажных, наладочных и пусковых работ, необходимых для ввода в действие АС;
- 3) в общем случае АС (подсистема АС) состоит из программно-технических (ПТК), программно-методических комплексов (ПМК) и компонентов технического, программного и информационного обеспечений. Компоненты этих видов обеспечения, а также ПМК и ПТК должны изготовляться и поставляется как продукция производственно-технического назначения. Компоненты могут входить в АС в качестве самостоятельных частей или могут быть объединены в комплексы;
- 4) создание АС в организациях (предприятиях) требует специальной подготовки пользователей и обслуживающего персонала системы;
- 5) функционирование АС и комплексов обеспечивается совокупностью организационно-методических документов, рассматриваемых в процессе создания как компоненты правового, методического, лингвистического, математического, организационного и др. видов обеспечений. Отдельные решения, получаемые в процессе разработки этих обеспечений, могут реализовываться в виде компонентов технического, программного или информационного обеспечений;
- 6) совместное функционирование и взаимодействие различных систем и комплексов осуществляется на базе локальных сетей ЭВМ.

Спецификации и соглашения, принятые для локальных сетей ЭВМ, обязательны для обеспечения совместимости систем, комплексов и компонентов.

2. Взаимосвязь ЕКС АС с другими системами и комплексами стандартов

2.1. Стандартизация в области АС является составной частью работ по обобщенной проблеме «Информационная технология».

2.2. Единый комплекс стандартов руководящих документов на автоматизированные системы совместно с другими системами и комплексами стандартов должен образовывать полное нормативно-техническое обеспечение процессов создания и функционирования АС.

2.3. ЕКС АС должен охватывать специфические для автоматизированных систем направления стандартизации и распространять традиционные направления стандартизации на программно-технические, программно-методические комплексы и автоматизированные системы в целом.

2.4. Направления и задачи стандартизации при нормативно-техническом обеспечении процессов создания и функционирования АС группируют следующим образом:

- 1) установление технических требований к продукции;
- 2) регламентация методов испытаний и правил аттестации и сертификации продукции;
- 3) регламентация правил и порядка разработки;
- 4) установление правил документирования;
- 5) обеспечение совместимости;
- 6) регламентация организационно-методических вопросов функционирования систем.

Направления 1-4 являются традиционными при разработке, изготовлении и поставке продукции. Направления 5, 6 являются специфичными и вытекают из особенностей, присущих АС.

2.5. Обеспеченность АС в целом и их составных частей нормативно-технической документацией в рамках принятых направлений и задач стандартизации различна.

Компоненты технического, программного и информационного обеспечений, как продукцию производственно-технического назначения, рассматривают, соответственно, как конструкторские, программные и информационные изделия. На эти изделия распространяются действующие комплексы стандартов ЕСКД, СРПП, ЕСПД, СГИП, УСД, классификаторы и кодификаторы технико-экономической информации, комплексы стандартов вида «ОТТ», «Методы испытаний», «ТУ», а также ОТТ заказчика.

2.5.1. Весь жизненный цикл конструкторских изделий полностью обеспечен нормативно-технической документацией, действующей в машиностроении и приборостроении.

2.5.2. Программные изделия обеспечены НТД, входящей в ЕСПД и ОТТ заказчика. Однако область распространения этих НТД должна быть расширена с целью отражения вопросов, связанных с разработкой, созданием, распространением и эксплуатацией программных изделий.

2.5.3. Информационные изделия в настоящее время не обеспечены НТД, хотя отдельные вопросы проработаны в рамках УСД, классификаторах и кодификаторах технико-экономической информации.

2.6. Программно-технические и программно-методические комплексы рассматриваются как сложные изделия, не имеющие аналогов в машиностроении. Учитывая статус ПТК и ПМК как продукции производственно-технического назначения, правила и порядок их разработки должен быть аналогичен требованиям, установленным стандартами системы разработки и постановки продукции на производство (СРПП).

Приложение Б

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на разработку ИС «Система»

Общие сведения

1.1. Наименование системы

Аналитическая информационная система «Система».

Назначение и цели создания системы

Система «Система» предназначена для информационного обеспечения процессов, которые происходят на кафедре связанных с учебно-методической, научной, общественной, организационно-методической и воспитательной работой.

Характеристика объектов информатизации

3.1. Краткое описание работы кафедры

К основным направлениям работы кафедры относятся:

- Учебно-методическая работа;
- Научная работа;
- Организационно-методическая работа;
- Работа со студентами заочниками;
- Общественная работа;
- Воспитательная работа.

…

3.2. Описание объектов информатизации

К основным объектам информатизации системы относятся:

Кафедра

- Наименование кафедры
- Факультет, к которому относится кафедра
- Веб-сайт кафедры
- Заведующий кафедрой

…

3.2.1. Учебно-методическая работа

План учебно-методической работы кафедры

- Учебный год
- Заведующий кафедрой, составивший план
- Кафедра

Тема для учебно-методической работы

- Названия работ
- Сроки исполнения
- Ответственные за выполнение темы

…

Требования к информационной системе

4.1. Базовые принципы разработки подсистем

При проектировании и разработке подсистем должны использоваться следующие базовые принципы:

- Исключение дублирования ввода информации и повышение ее достоверности, за счет отождествления ранее введенной информации;

…

Система должна удовлетворять следующим требованиям:

- Пользовательский интерфейс системы должен быть сформирован в соответствии с навыками и профилем пользователей;

…

Система должна содержать:

- Средства поиска информации;

…

Выбор прикладного программного обеспечения системы должен удовлетворять следующим критериям:

- Интеграция с базами данных, поддерживающих Web-технологии;

…

4.2. Требования к архитектуре системы.

Архитектура системы «Система» является трехзвенной. В качестве клиентского приложения выступает стандартный веб-браузер.

…

4.3. Требования к способам и средствам связи для информационного обмена между компонентами (модулями) Системы

Подсистемы должны взаимодействовать в пределах единой компьютерной сети (Интернет/Интранет), в которой происходит весь обмен информацией.

…

4.4. Требования к характеристикам взаимосвязей системы со смежными системами

Смежными системами для информационной системы «Система» являются: «Система2»,

…

4.5. Требования к режимам функционирования подсистемы

Разрабатываемая система должна функционировать 24 часа в сутки, 365 дней в году…

…

4.6. Требования к пользователям

Система подразумевает четыре типа пользователя:

- Сотрудник – имеет доступ к просмотру общих данных по своей кафедре, а также к просмотру и редактированию личных данных, имеет возможность ;

…

4.7. Требования по эргономике и технической эстетике

Основными требованиями по эргономике и технической эстетике является адекватность времени реакции модулей системы на сложность запроса пользователя к базам данных:

- При выполнении стандартных запросов пользователь должен работать с системой в реальном режиме времени;

…

4.8. Требования к численности и квалификации персонала системы и режиму его работы.

Квалификация персонала, порядок его подготовки и контроль знаний и навыков.

…

4.9. Требования к защите информации от несанкционированного доступа.

Разрабатываемая система должна обладать специализированной подсистемой разграничения доступа к информационным ресурсам, функционирующей на основе системы пользователей и пользовательских групп.

…

4.10. Требования к обмену данными

- Обмен данными должен происходить по сети в среде Intranet/Internet с поддержкой протокола TCP/IP;

…

4.11. Требования к внешней среде системы

Сервер баз данных или сервер приложений должен обеспечивать:

…

4.12. Требования к хранению данных

База данных «Система» должна содержать следующие данные:

- Данные о планировании учебно-методической работы;

…

4.13. Требования к отдельным подсистемам

4.13.1. Учебно-методическая работа

Функции заведующего кафедрой

- Создание плана учебно-методической работы на учебный семестр, заполнения, редактирования и удаления данных плана;

…

Состав и содержание работ по созданию Системы

Разработать модель БД, позволяющую хранить и обрабатывать все необходимые…

…

Приемо-сдаточные испытания Системы

После завершения всех работ по разработке компонентов, настройке подсистем и

…

Внесение корректировок в программный продукт, связанных с ошибками в Системе

Все ошибки, которые будут выявлены в работе Системы в течении 12 месяцев

…

Тестирование

Перед сдачей Модулей и Компонент Заказчику для выявления возможных сбоев в работе

…

Порядок контроля и приемки Системы

Для проверки выполнения заданных функций Системы, определения и проверки соответствия требованиям ТЗ количественных и (или) качественных характеристик Системы, выявления и устранения недостатков в действиях Системы и в разработанной документации, поэтапного контроля над ходом разработки должны быть проведены следующие виды испытаний:

- Предварительные;

…

Процедуры тестирования и контроля качества

При проведении испытаний должны использоваться следующие типы процедур тестирования и контроля качества:

- функциональное тестирование - тестирование ПО на соответствие функциональным спецификациям;

…

Общие требования к приемке работ

Сроки и место приемки, порядок приемки работ определяются в соответствии с настоящим ТЗ.

…

Требования к документированию

12.1. Требования к проектной документации

Состав и комплектность проектной документации должна соответствовать требованиям ГОСТ 34.201-89.

Перечень документации по созданию системы включает:

- Описание информационного обеспечения системы (П5);

…

Лабораторная работа №3

«Методология функционального моделирования»

8. Цель работы:

Изучить методологии функционального моделирования IDEF0 и IDEF3.

9. Методические указания

Лабораторная работа направлена на ознакомление с методологиями функционального моделирования IDEF0 и IDEF3, получение навыков по применению данных методологий для построения функциональных моделей на основании требований к информационной системе.

Требования к результатам выполнения лабораторного практикума:

- модель должна отражать весь указанный в описании функционал, а также чётко отражать существующие потоки данных и описывать правила их движения;
- наличие в модели не менее трёх уровней;
- не менее двух уровней декомпозиции в стандарте IDEF0 (контекстная диаграмма + диаграммы A0);
- на диаграмме 1-го уровня (A0) не менее 4-х функциональных блоков;
- на диаграмме 2-го и далее уровнях должна быть декомпозиция в стандарте IDEF3, на каждой диаграмме не менее 2-х функциональных блоков.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных в работе [1] или на странице http://unesco.kemsu.ru/student/rule/rule.html.

10. Теоретические сведения

IDEF0. Основные понятия IDEF0

IDEF0 (Integrated Definition Function Modeling) - методология функционального моделирования. В основе IDEF0 методологии лежит понятие блока, который отображает некоторую бизнес-функцию. Четыре стороны блока имеют разную роль: левая сторона имеет значение "входа", правая - "выхода", верхняя - "управления", нижняя - "механизма" (рис. 1).

Взаимодействие между функциями в IDEF0 представляется в виде дуги, которая отображает поток данных или материалов, поступающий с выхода одной функции на вход другой. В зависимости от того, с какой стороной блока связан поток, его называют соответственно "входным", "выходным", "управляющим".

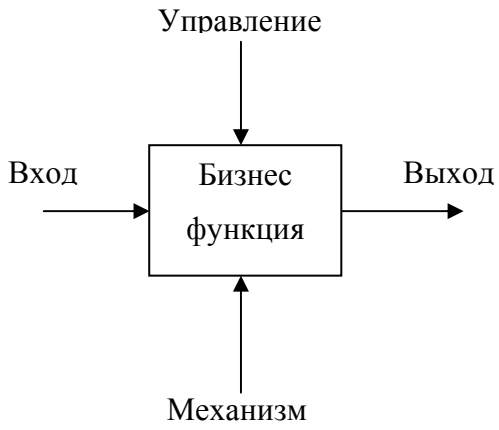


Рис. 1. Функциональный блок

Принципы моделирования в IDEF0

В IDEF0 реализованы три базовых принципа моделирования процессов:

– принцип функциональной декомпозиции;

– принцип ограничения сложности;

– принцип контекста.

Принцип функциональной декомпозиции представляет собой способ моделирования типовой ситуации, когда любое действие, операция, функция могут быть разбиты (декомпозированы) на более простые действия, операции, функции. Другими словами, сложная бизнес-функция может быть представлена в виде совокупности элементарных функций. Представляя функции графически, в виде блоков, можно как бы заглянуть внутрь блока и детально рассмотреть ее структуру и состав (рис. 2).

Принцип ограничения сложности. При работе с IDEF0 диаграммами существенным является условие их разборчивости и удобочитаемости. Суть принципа ограничения сложности состоит в том, что количество блоков на диаграмме должно быть не менее двух и не более шести. Практика показывает, что соблюдение этого принципа приводит к тому, что функциональные процессы, представленные в виде IDEF0 модели, хорошо структурированы, понятны и легко поддаются анализу.

Принцип контекстной диаграммы. Моделирование делового процесса начинается с построения контекстной диаграммы. На этой диаграмме отображается только один блок - главная бизнес-функция моделируемой системы. Если речь идет о моделировании целого предприятия или даже крупного подразделения, главная бизнес-функция не может быть сформулирована как, например, "продавать продукцию". Главная бизнес-функция системы - это "миссия" системы, ее значение в окружающем мире. Нельзя правильно сформулировать главную функцию предприятия, не имея представления о его стратегии.

При определении главной бизнес-функции необходимо всегда иметь ввиду цель моделирования и точку зрения на модель. Одно и то же предприятие может быть описано по-разному, в зависимости от того, с какой точки зрения его рассматривают: директор предприятия и налоговой инспектор видят организацию совершенно по-разному.

Контекстная диаграмма играет еще одну роль в функциональной модели. Она "фиксирует" границы моделируемой бизнес-системы, определяя то, как моделируемая система взаимодействует со своим окружением. Это достигается за счет описания дуг, соединенных с блоком, представляющим главную бизнес-функцию.

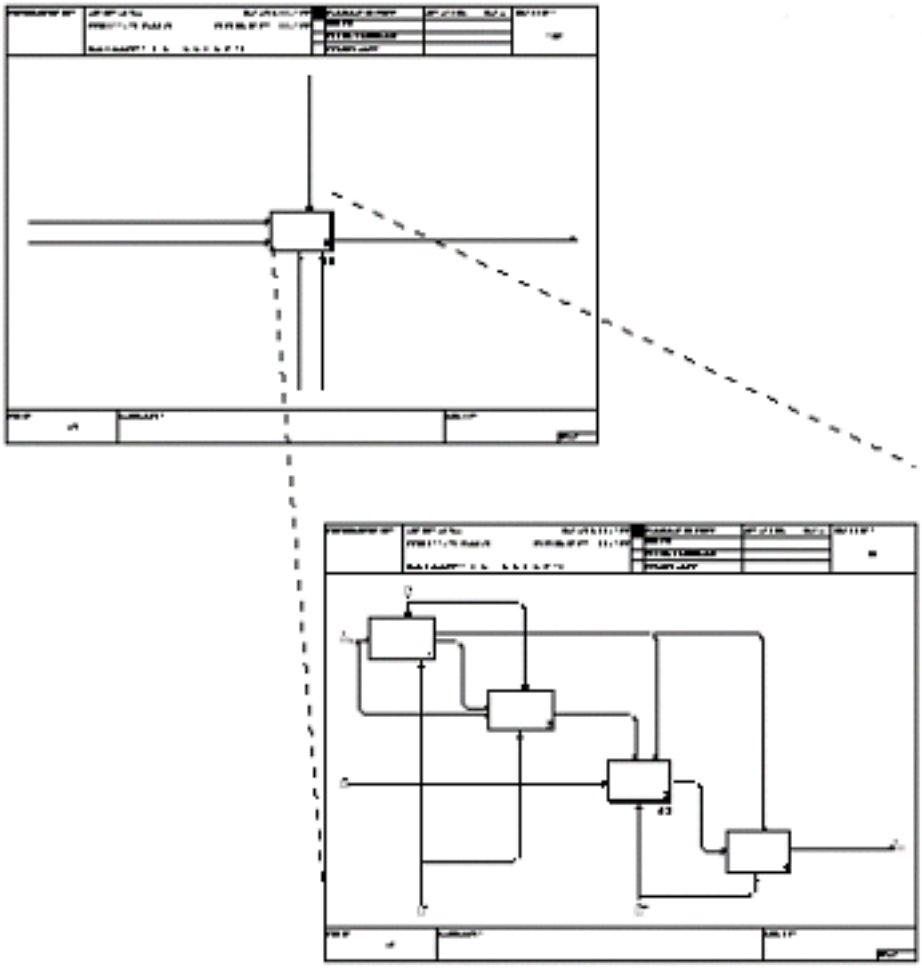


Рис. 2. Декомпозиция функционального блока

Пример.

На рис. 3 и рис. 4 представлен пример построения функциональной диаграммы, описывающей изготовление изделия. Рис. 3 - контекстная диаграмма. Рис. 4 – первый уровень декомпозиции.

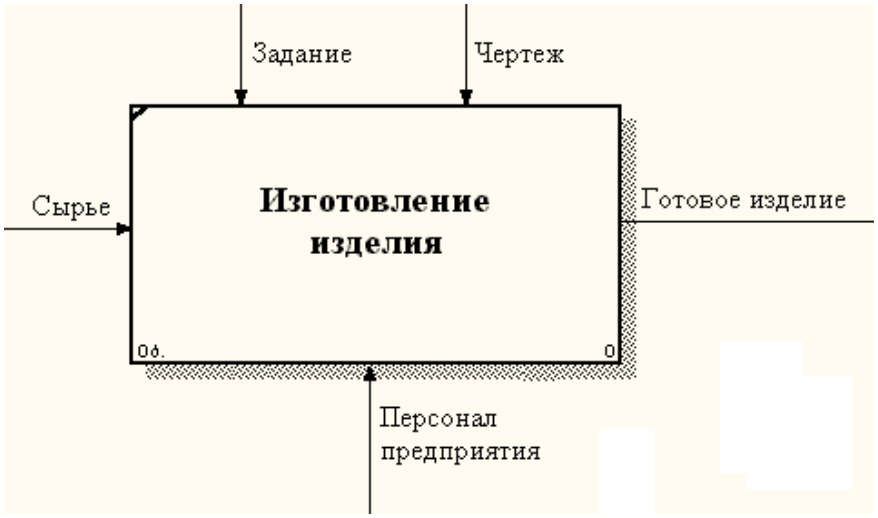


Рис. 3. Контекстная диаграмма

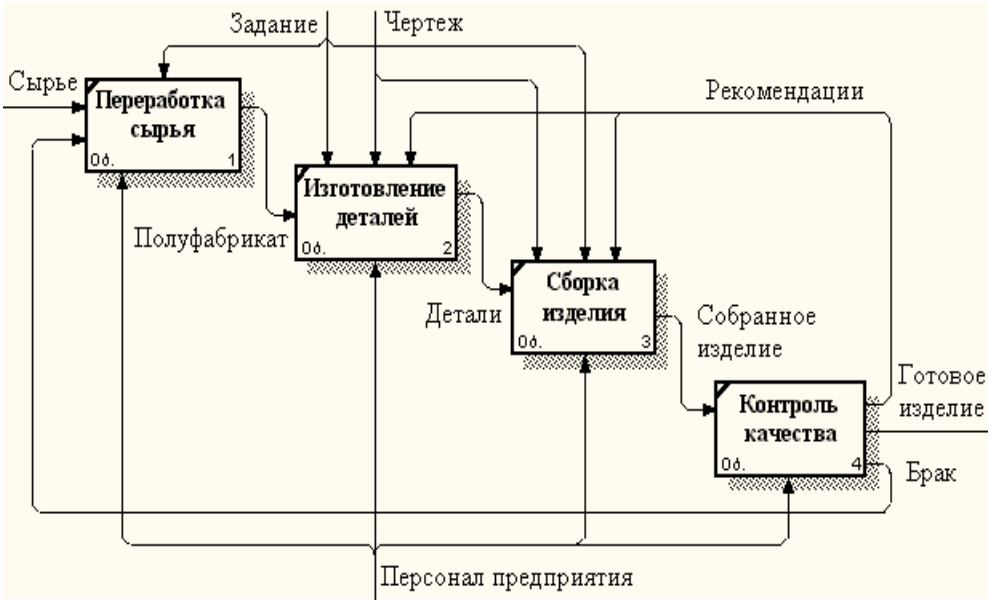


Рис.4. Диаграмма первого уровня декомпозиции

1.1. Применение IDEF0

Существует два ключевых подхода к построению функциональной модели: построение “как есть” и построение “как будет”.

Построение модели “как есть”. Обследование предприятия является обязательной частью любого проекта создания или развития корпоративной информационной системы.

Построение функциональной модели “как есть” позволяет четко зафиксировать, какие деловые процессы осуществляются на предприятии, какие информационные объекты используются при выполнении деловых процессов и отдельных операций. Функциональная модель “как есть” является отправной точкой для анализа потребностей предприятия, выявления проблем и "узких" мест и разработки проекта совершенствования деловых процессов.

Построение модели “как будет”. Создание и внедрение корпоративной информационной системы приводит к изменению условий выполнения отдельных операций, структуры деловых процессов и предприятия в целом. Это приводит к необходимости изменения системы бизнес-правил, используемых на предприятии, модификации должностных инструкций сотрудников. Функциональная модель “как будет” позволяет уже на стадии проектирования будущей информационной системы определить эти изменения. Применение функциональной модели “как будет” позволяет не только сократить сроки внедрения информационной системы, но также снизить риски, связанные с невосприимчивостью персонала к информационным технологиям.

IDEF3. Метод описания процессов IDEF3

Для описания логики взаимодействия информационных потоков наиболее подходит IDEF3, называемая также workflow diagramming - методологией моделирования, использующая графическое описание информационных потоков, взаимоотношений между процессами обработки информации и объектов, являющихся частью этих процессов.

Диаграммы Workflow могут быть использованы в моделировании бизнес-процессов для анализа завершенности процедур обработки информации. С их помощью можно описывать сценарии действий сотрудников организации, например последовательность обработки заказа или события, которые необходимо обработать за конечное время. Каждый сценарий сопровождается описанием процесса и может быть использован для документирования каждой функции.

IDEF3 - это метод, имеющий основной целью дать возможность аналитикам описать ситуацию, когда процессы выполняются в определенной последовательности, а также описать объекты, участвующие совместно в одном процессе,

Техника описания набора данных IDEF3 является частью структурного анализа. В отличие от некоторых методик описаний процессов IDEF3 не ограничивает аналитика чрезмерно жесткими рамками синтаксиса, что может привести к созданию неполных или противоречивых моделей.

IDEF3 может быть также использован как метод создания процессов. IDEF3 дополняет IDEF0 и содержит все необходимое для построения моделей, которые в дальнейшем могут быть использованы для имитационного анализа.

Каждая работа в IDEF3 описывает какой-либо сценарий бизнес-процесса и может являться составляющей другой работы. Поскольку сценарий описывает цель и рамки модели, важно, чтобы работы именовались отглагольным существительным, обозначающим процесс действия, или фразой, содержащей такое существительное.

Точка зрения на модель должна быть задокументирована. Обычно это точка зрения человека, ответственного за работу в целом. Также необходимо задокументировать цель модели - те вопросы, на которые призвана ответить модель.

Диаграммы. Диаграмма является основной единицей описания в IDEF3.

Единицы работы - Unit of Work (UOW). UOW, также называемые работами (activity), являются центральными компонентами модели. В IDEF3 работы изображаются прямоугольниками с прямыми углами и имеют имя, выраженное отглагольным существительным, обозначающим процесс действия, одиночным или в составе фразы, и номер (идентификатор); другое имя существительное в составе той же фразы обычно отображает основной выход (результат) работы, например, "Изготовление изделия".

Связи. Связи показывают взаимоотношения работ. Все связи в IDEF3 однонаправлены и могут быть направлены куда угодно, но обычно диаграммы IDEF3 стараются построить так, чтобы связи были направлены слева направо. В IDEF3 различают три типа стрелок, изображающих связи, стиль которых устанавливается через меню Edit/Arrow Style:

– Старшая (Precedence) - сплошная линия, связывающая единицы работ (UOW), Рисуется слева направо или сверху вниз. Показывает, что работа-источник должна закончиться прежде, чем работа-цель начнется.

– Отношения (Relational Link) - пунктирная линия, использующаяся для изображения связей между единицами работ (UOW) а также между единицами работ и объектами ссылок.

– Потоки объектов (Object Flow) - стрелка с двумя наконечниками, применяется для описания того факта, что объект используется в двух или более единицах работы, например, когда объект порождается в одной работе и используется в другой.

– Старшая связь и поток объектов. Старшая связь показывает, что работа-источник заканчивается ранее, чем начинается работа-цель. Часто результатом работы-

источника становится объект, необходимый для запуска работы-цели. В этом случае стрелку, обозначающую объект, изображают с двойным наконечником. Имя стрелки должно ясно идентифицировать отображаемый объект. Поток объектов имеет ту же семантику, что и старшая стрелка.

Перекрестки (Junction). Окончание одной работы может служить сигналом к началу нескольких работ, или же одна работа для своего запуска может ожидать окончания нескольких работ. Перекрестки используются для отображения логики взаимодействия стрелок при слиянии и разветвлении или для отображения множества событий, которые могут или должны быть завершены перед началом следующей работы. Различают перекрестки для слияния (Fan-in Junction) и разветвления (Fan-out Junction) стрелок. Перекресток не может использоваться одновременно для слияния и для разветвления. Для внесения перекрестка служит кнопка в палитре инструментов - добавить в диаграмму перекресток Junction. В диалоге Junction Type Editor необходимо указать тип перекрестка. Смысл каждого типа приведен в таблице 1.

Таблица 1 - Типы перекрестков

Обозначение	**Наименование**	**Смысл в случае слияния стрелок**	**Смысл в случае разветвления стрелок**
	Asynchronous AND	Все предшествующие процессы должны быть завершены	Все следующие процессы должны быть запущены
	Synchronous AND	Все предшествующие процессы завершены одновременно	Все следующие процессы запускаются одновременно
	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
	Synchronous OR	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
	XOR (Exclusive OR)	Только один предшествующий процесс завершен	Только один следующий процесс запускается

В отличие от IDEF0 в IDEF3 стрелки могут сливаться и разветвляться только через перекрестки.

Декомпозиция работ. В IDEF3 декомпозиция используется для детализации работ. Методология IDEF3 позволяет декомпозировать работу многократно, т.е. работа может иметь множество дочерних работ. Это позволяет в одной модели описать альтернативные потоки. Возможность множественной декомпозиции предъявляет дополнительные требования к нумерации работ. Так, номер работы состоит из номера родительской работы, версии декомпозиции и собственного номера работы на текущей диаграмме (рис. 5).

Рис. 5. Номер единицы работы (UOW)

11. Порядок выполнения работы

1. Изучить предлагаемый теоретический материал.
2. Построить функциональную модель системы, описанной в Лабораторной работе № 1 так, чтобы она отвечала всем предъявленным к системе требованиям, представляла полный функционал системы (каждой функции в описании системы должен соответствовать по крайней мере один функциональный блок) и её основные бизнес-процессы:
   - с помощью методологии IDEF0 построить контекстную диаграмму;
   - с помощью методологии IDEF0 построить диаграмму 1-го уровня (A0) – модель окружения;
   - с помощью методологии IDEF3 декомпозировать функциональные блоки модели окружения на 1-2 уровня вглубь до потоков, связи с внешними системами и хранилищами;
   - на каждой диаграмме 2-го уровня должно быть не менее 4-х функциональных блоков;
   - на каждой диаграмме 3-го уровня и далее не менее 2-х функциональных блоков.
3. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

12. Содержание отчета

В отчете следует указать:

13. Цель работы
14. Введение
15. Программно-аппаратные средства, используемые при выполнении работы.
16. Основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума (п.2).
17. Заключение (выводы)
18. Список используемой литературы

6. Литература

7. http://www.idef.com

8. http://www.idefinfo.ru/

9. Свиридов С., Курьян А.. IDEF0: функциональное моделирование деловых процессов // Центр ОТСМ-ТРИЗ технологий, Минск, Беларусь 1997. http://www.trizminsk.org

10. Чувахин В. А. Описание отдельных концепций IDEF0// Сайт “Корпоративный менеджмент”. http://www.cfin.ru/chuvakhin/idef0-r.shtml

11. Курьян А. Г., Серенков П.С. Использование IDEF0 для описания и классификации процессов в рамках системы качества МС ИСО семейства 9000 версии 2000. // http://www.interface.ru/

12. Рубцов С.. IDEF0 и опыт разработки. Секреты моделирования и проектирования бизнес-процессов. // Открытые системы, 2002. http://big.spb.ru/

13. Верников Г.. Основные методологии обследования организаций. Стандарт IDEF0. // Управленческое консультирование. www.consulting.ru

14. Ляхов В. Ф. Практикум по Bpwin // СевКавГТУ кафедра «Информационных систем и технологий»

15. Маклаков С. В. BPwin и ERwin: CASE-средства для разработки информационных систем // http://www.isuct.ru/~ivt/books/CASE/case5

7. Контрольные вопросы

16. Перечислите основные объекты IDEF0, их описание и назначение.

17. Назовите базовые принципы моделирования в IDEF0.

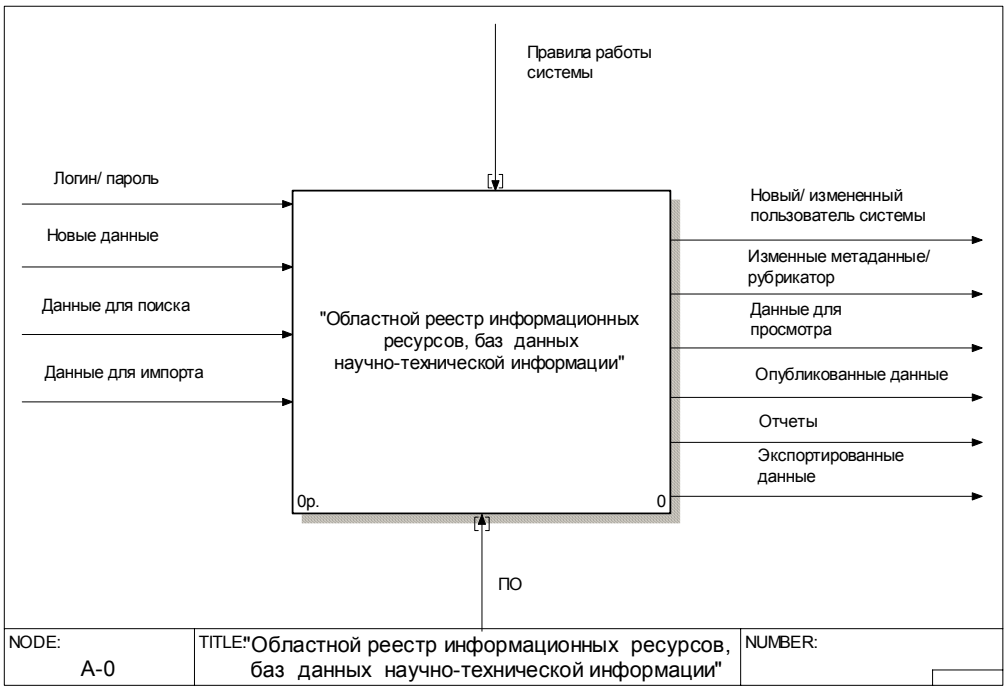
18. В каких случаях целесообразно применять построение модели “как есть”, а в каких “как будет”?

19. Перечислите основные объекты IDEF3, их описание и назначение.

20. В чём смысл использования перекрёстков в IDEF3?

21. В чём отличия IDEF0 и IDEF3? Когда целесообразней использовать IDEF0, а когда IDEF3?

Пример построения функциональной модели для информационной системы «Областной реестр информационных ресурсов, баз данных научно-технической информации»



Контекстная диаграмма (IDEF0)

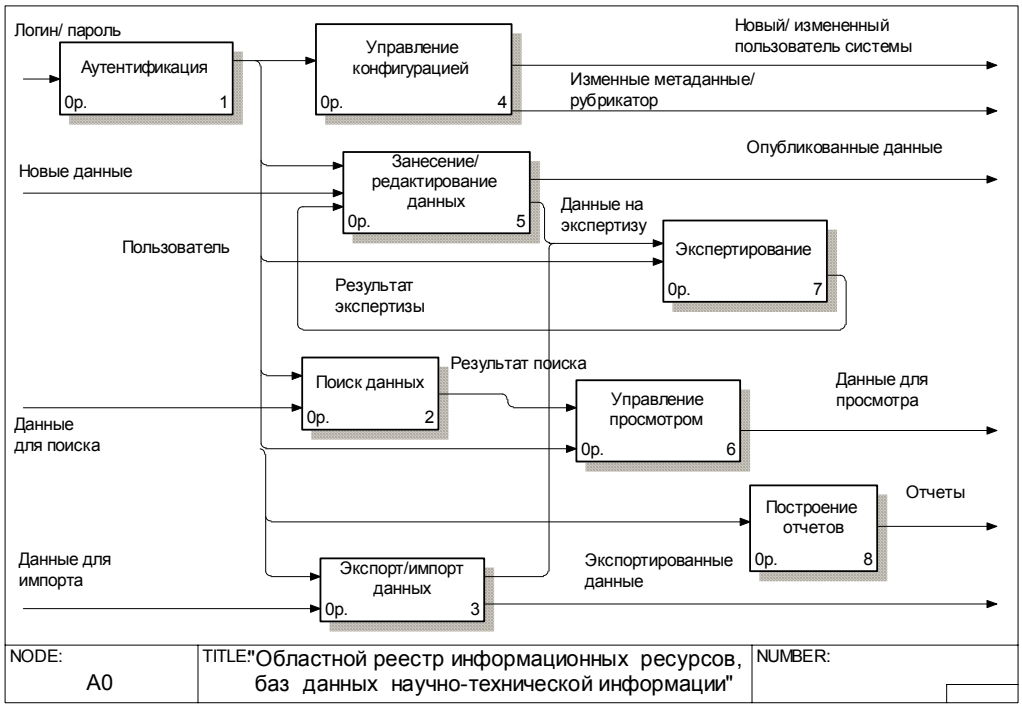


Диаграмма 1-го уровня (IDEF0)

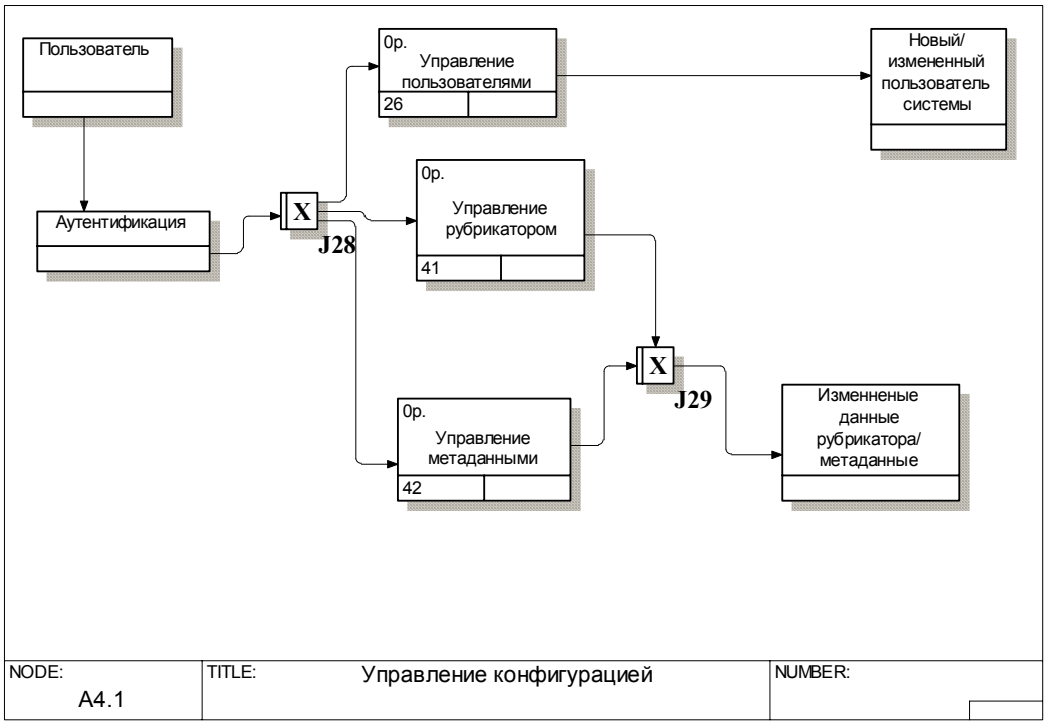


Диаграмма 2-го уровня (IDEF3) – декомпозиция функционального блока «Управление конфигурацией»

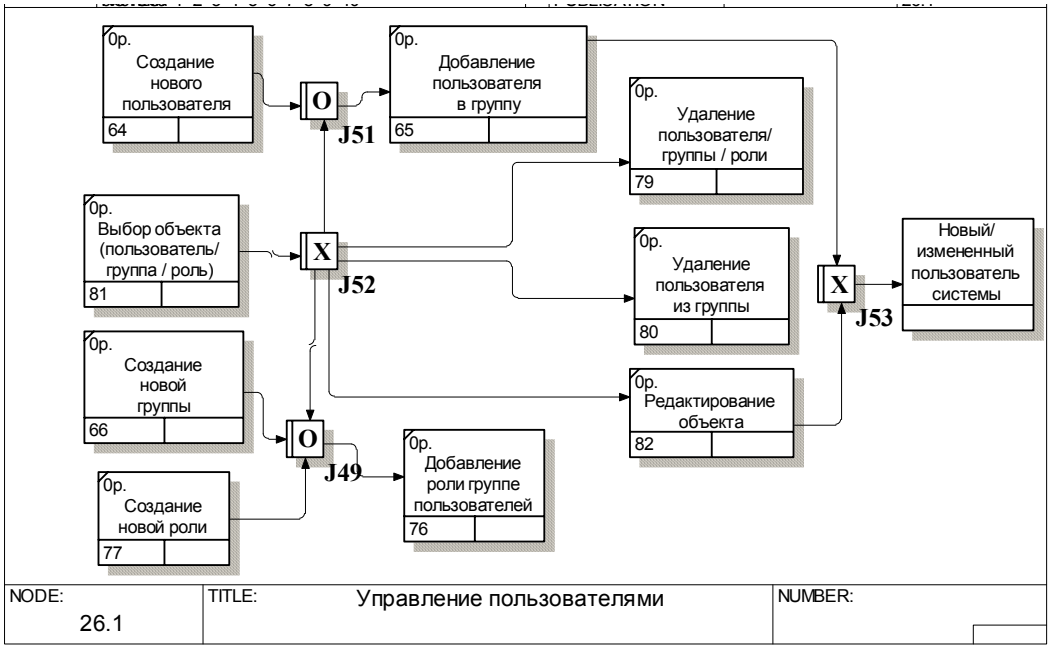


Диаграмма 3-го уровня (IDEF3) - декомпозиция функционального блока «Управление пользователями»

Лабораторная работа №4

«Методология объектно-ориентированного моделирования»

1. Цель работы:

Ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML.

2. Методические указания

Лабораторная работа направлена на ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML, получение навыков по применению данных элементов для построения объектно-ориентированных моделей ИС на основании требований.

Требования к результатам выполнения лабораторного практикума:

- модель системы должна содержать: диаграмму вариантов использования; диаграммы взаимодействия для каждого варианта использования; диаграмму классов, позволяющая реализовать весь описанный функционал ИС; объединенную диаграмму компонентов и размещения
- для классов указать стереотипы;
- в зависимости от варианта задания диаграмма размещения должна показывать расположение компонентов в распределенном приложении или связи между встроенным процессором и устройствами.

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных в работе [1] или на странице http://unesco.kemsu.ru/student/rule/rule.html.

3. Общие сведения об объектном моделировании ИС

Существует множество технологий и инструментальных средств, с помощью которых можно реализовать в некотором смысле оптимальный проект ИС, начиная с этапа анализа и заканчивая созданием программного кода системы. В большинстве случаев эти технологии предъявляют весьма жесткие требования к процессу разработки и используемым ресурсам, а попытки трансформировать их под конкретные проекты оказываются безуспешными. Эти технологии представлены CASE-средствами верхнего уровня или CASE-средствами полного жизненного цикла (upper CASE tools или full life-cycle CASE tools). Они не позволяют оптимизировать деятельность на уровне отдельных элементов проекта, и, как следствие, многие разработчики перешли на так называемые CASE-средства нижнего уровня (lower CASE tools). Однако они столкнулись с новой проблемой — проблемой организации взаимодействия между различными командами, реализующими проект.

Унифицированный язык объектно-ориентированного моделирования Unified Modeling Language (UML) явился средством достижения компромисса между этими подходами. Существует достаточное количество инструментальных средств, поддерживающих с помощью *UML* жизненный цикл информационных систем, и,

одновременно, *UML* является достаточно гибким для настройки и поддержки специфики деятельности различных команд разработчиков.

Создание UML началось в октябре 1994 г., когда Джим Рамбо и Гради Буч из Rational Software Corporation стали работать над объединением своих методов OMT и Booch. В настоящее время консорциум пользователей UML Partners включает в себя представителей таких грандов информационных технологий, как Rational Software, Microsoft, IBM, Hewlett-Packard, Oracle, DEC, Unisys, IntelliCorp, Platinum Technology.

UML представляет собой *объектно-ориентированный* язык моделирования, обладающий следующими основными характеристиками:

- является языком визуального моделирования, который обеспечивает разработку репрезентативных моделей для организации взаимодействия заказчика и разработчика ИС, различных групп разработчиков ИС;
- содержит механизмы расширения и специализации базовых концепций языка.

UML — это стандартная нотация визуального моделирования программных систем, принятая консорциумом Object Managing Group (OMG) осенью 1997 г., и на сегодняшний день она поддерживается многими объектно-ориентированными CASE-продуктами.

UML включает внутренний набор средств моделирования, которые сейчас приняты во многих методах и средствах моделирования. Эти концепции необходимы в большинстве прикладных задач, хотя не каждая концепция необходима в каждой части каждого приложения. Пользователям языка предоставлены возможности:

- строить модели на основе средств ядра, без использования механизмов расширения для большинства типовых приложений;
- добавлять при необходимости новые элементы и условные обозначения, если они не входят в ядро, или специализировать компоненты, систему условных обозначений (нотацию) и ограничения для конкретных предметных областей.

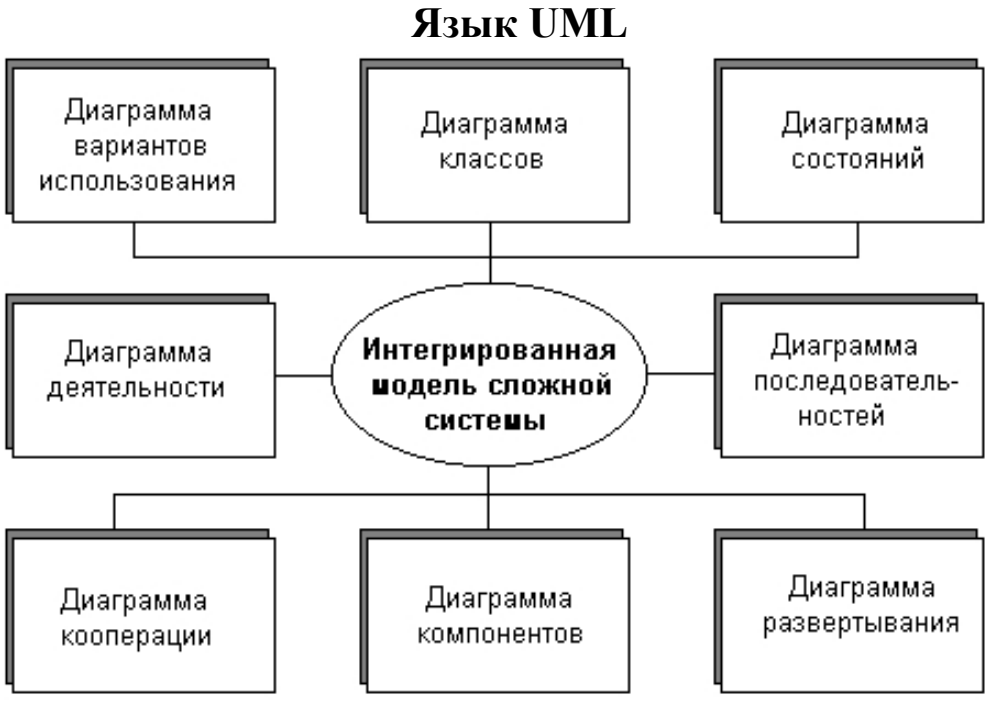


Рис. 1. Интегрированная модель сложной системы в нотации языка UML

Стандарт UML предлагает следующий набор диаграмм для моделирования:

- диаграммы вариантов использования (use case diagrams) – для моделирования бизнес-процессов организации и требований к создаваемой системе);
- диаграммы классов (class diagrams) – для моделирования статической структуры классов системы и связей между ними;
- диаграммы поведения системы (behavior diagrams):
  - диаграммы взаимодействия (interaction diagrams):
    - диаграммы последовательности (sequence diagrams) и
    - кооперативные диаграммы (collaboration diagrams) – для моделирования процесса обмена сообщениями между объектами;
  - диаграммы состояний (statechart diagrams) – для моделирования поведения объектов системы при переходе из одного состояния в другое;
  - диаграммы деятельностей (activity diagrams) – для моделирования поведения системы в рамках различных вариантов использования, или моделирования деятельностей;
- диаграммы реализации (implementation diagrams):
  - диаграммы компонентов (component diagrams) – для моделирования иерархии компонентов (подсистем) системы;
  - диаграммы развертывания (deployment diagrams) – для моделирования физической архитектуры системы.

Диаграммы вариантов использования

Понятие варианта использования (use case) впервые ввел Ивар Якобсон и придал ему такую значимость, что в настоящее время вариант использования превратился в основной элемент разработки и планирования проекта.

Вариант использования представляет собой последовательность действий (транзакций), выполняемых системой в ответ на событие, инициируемое некоторым внешним объектом (действующим лицом). Вариант использования описывает типичное взаимодействие между пользователем и системой. В простейшем случае вариант использования определяется в процессе обсуждения с пользователем тех функций, которые он хотел бы реализовать. На языке UML вариант использования изображают следующим образом:

Рис.2. Вариант использования

Действующее лицо (actor) – это роль, которую пользователь играет по отношению к системе. Действующие лица представляют собой роли, а не конкретных людей или наименования работ. Несмотря на то, что на диаграммах вариантов использования они изображаются в виде стилизованных человеческих фигурок, действующее лицо может также быть внешней системой, которой необходима некоторая информация от данной системы. Показывать на диаграмме действующих лиц следует только в том случае, когда им действительно необходимы некоторые варианты использования. На языке UML действующие лица представляют в виде фигур:

Рис.3. Действующее лицо (актер)

Действующие лица делятся на три основных типа:

- пользователи;
- системы;
- другие системы, взаимодействующие с данной;
- время.

Время становится действующим лицом, если от него зависит запуск каких-либо событий в системе.

Связи между вариантами использования и действующими лицами

В языке UML на диаграммах вариантов использования поддерживается несколько типов связей между элементами диаграммы. Это связи коммуникации (communication), включения (include), расширения (extend) и обобщения (generalization).

Связь коммуникации – это связь между вариантом использования и действующим лицом. На языке UML связи коммуникации показывают с помощью однонаправленной ассоциации (сплошной линии).

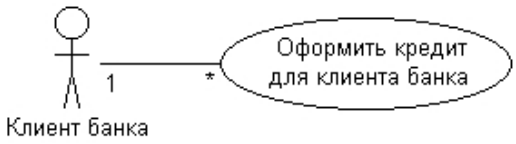


Рис.4. Пример связи коммуникации

Связь включения применяется в тех ситуациях, когда имеется какой-либо фрагмент поведения системы, который повторяется более чем в одном варианте использования. С помощью таких связей обычно моделируют многократно используемую функциональность.

Связь расширения применяется при описании изменений в нормальном поведении системы. Она позволяет варианту использования только при необходимости использовать функциональные возможности другого.

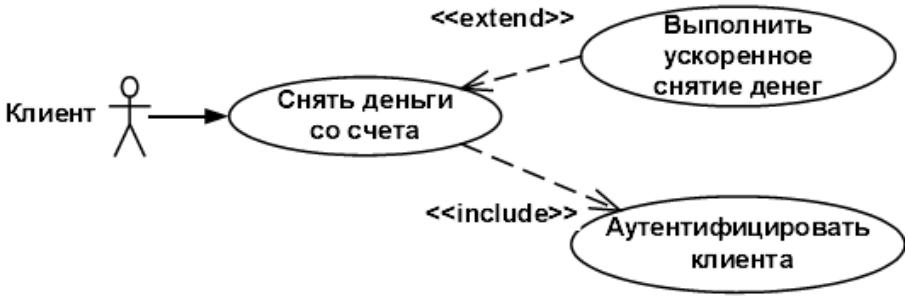


Рис.5. Пример связи включения и расширения

С помощью связи обобщения показывают, что у нескольких действующих лиц имеются общие черты.

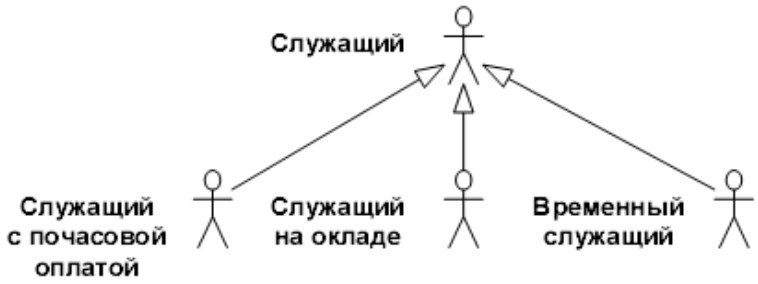


Рис.6. Пример связи обобщения

Диаграммы взаимодействия (interaction diagrams)

Диаграммы взаимодействия (interaction diagrams) описывают поведение взаимодействующих групп объектов. Как правило, диаграмма взаимодействия охватывает поведение объектов в рамках только одного варианта использования. На такой диаграмме отображается ряд объектов и те сообщения, которыми они обмениваются между собой.

Сообщение (message) – это средство, с помощью которого объект-отправитель запрашивает у объекта получателя выполнение одной из его операций.

Информационное (informative) сообщение – это сообщение, снабжающее объект-получатель некоторой информацией для обновления его состояния.

Сообщение-запрос (interrogative) – это сообщение, запрашивающее выдачу некоторой информации об объекте-получателе.

Императивное (imperative) сообщение – это сообщение, запрашивающее у объекта-получателя выполнение некоторых действий.

Существует два вида диаграмм взаимодействия: диаграммы последовательности (sequence diagrams) и кооперативные диаграммы (collaboration diagrams).

Диаграмма последовательности (sequence diagrams)

Диаграмма последовательности отражает поток событий, происходящих в рамках варианта использования.

Все действующие лица показаны в верхней части диаграммы. Стрелки соответствуют сообщениям, передаваемым между действующим лицом и объектом или между объектами для выполнения требуемых функций.

На диаграмме последовательности объект изображается в виде прямоугольника, от которого вниз проведена пунктирная вертикальная линия. Эта линия называется линией жизни (lifeline) объекта. Она представляет собой фрагмент жизненного цикла объекта в процессе взаимодействия.

Каждое сообщение представляется в виде стрелки между линиями жизни двух объектов. Сообщения появляются в том порядке, как они показаны на странице сверху вниз. Каждое сообщение помечается как минимум именем сообщения. При желании можно добавить также аргументы и некоторую управляющую информацию. Можно показать самоделегирование (self-delegation) – сообщение, которое объект

посылает самому себе, при этом стрелка сообщения указывает на ту же самую линию жизни.

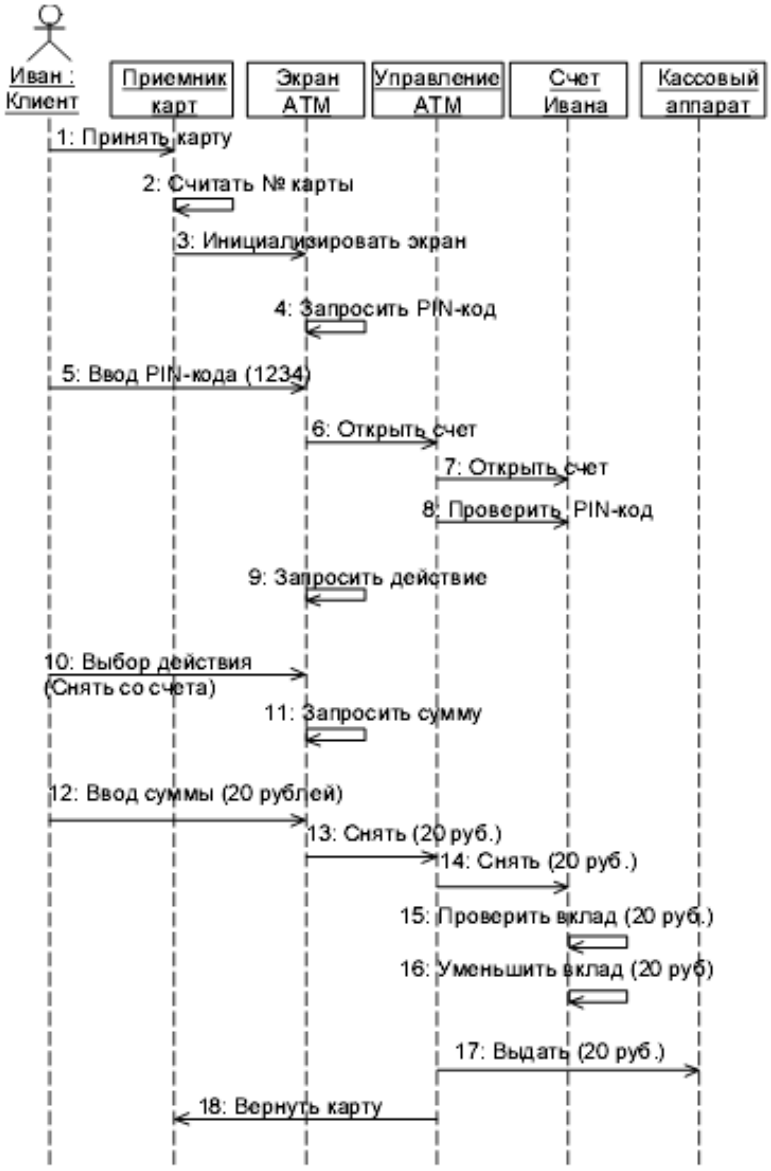


Рис. 7. Пример диаграммы последовательности

2.2.2. Диаграмма кооперации (collaboration diagram)

Диаграммы кооперации отображают поток событий через конкретный сценарий варианта использования, упорядочены по времени, а кооперативные диаграммы больше внимания заостряют на связях между объектами.

На диаграмме кооперации представлена вся та информация, которая есть и на диаграмме последовательности, но кооперативная диаграмма по-другому описывает поток событий. Из нее легче понять связи между объектами, однако, труднее уяснить последовательность событий.

На кооперативной диаграмме так же, как и на диаграмме последовательности, стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность указывается путем нумерации сообщений.

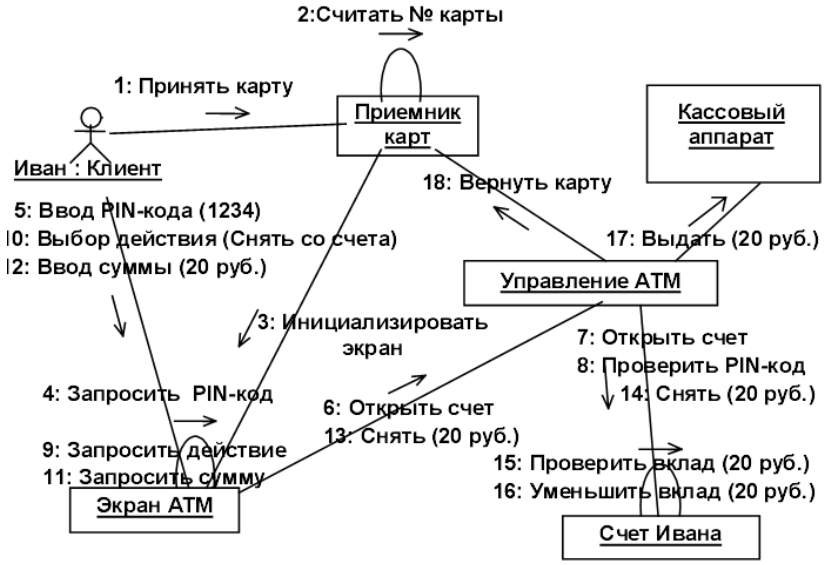


Рис. 8. Пример диаграммы кооперации

Диаграммы классов

Общие сведения

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними. На диаграммах классов изображаются также атрибуты классов, операции классов и ограничения, которые накладываются на связи между классами.

Диаграмма классов UML - это граф, узлами которого являются элементы статической структуры проекта (классы, интерфейсы), а дугами - отношения между узлами (ассоциации, наследование, зависимости).

На диаграмме классов изображаются следующие элементы:

– Пакет (package) - набор элементов модели, логически связанных между собой;
– Класс (class) - описание общих свойств группы сходных объектов;
– Интерфейс (interface) - абстрактный класс, задающий набор операций, которые объект произвольного класса, связанного с данным интерфейсом, предоставляет другим объектам.

Класс

Класс - это группа сущностей (объектов), обладающих сходными свойствами, а именно, данными и поведением. Отдельный представитель некоторого класса называется объектом класса или просто объектом.

Под поведением объекта в UML понимаются любые правила взаимодействия объекта с внешним миром и с данными самого объекта.

На диаграммах класс изображается в виде прямоугольника со сплошной границей, разделенного горизонтальными линиями на 3 секции:

– Верхняя секция (секция имени) содержит имя класса и другие общие свойства (в частности, стереотип).
– В средней секции содержится список атрибутов
– В нижней - список операций класса, отражающих его поведение (действия, выполняемые классом).

Любая из секций атрибутов и операций может не изображаться (а также обе сразу). Для отсутствующей секции не нужно рисовать разделительную линию и как-либо указывать на наличие или отсутствие элементов в ней.

На усмотрение конкретной реализации могут быть введены дополнительные секции, например, исключения (Exceptions).

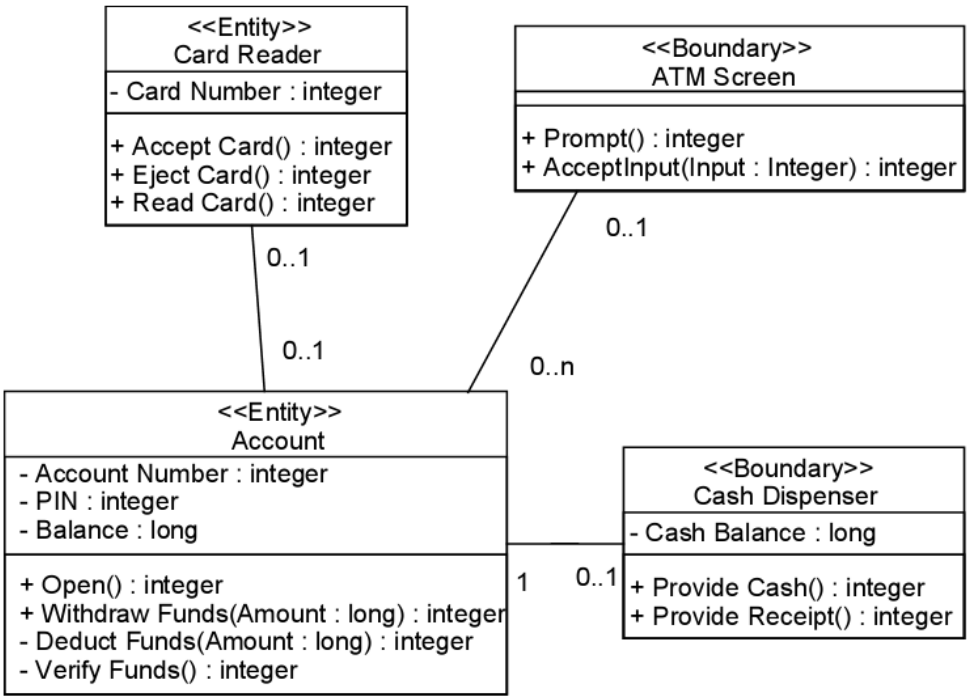


Рис. 9. Пример диаграммы классов

Стереотипы классов

Стереотипы классов – это механизм, позволяющий разделять классы на категории.

В языке UML определены три основных стереотипа классов:

– Boundary (граница);
– Entity (сущность);
– Control (управление).

Граничные классы

Граничными классами (boundary classes) называются такие классы, которые расположены на границе системы и всей окружающей среды. Это экранные формы, отчеты, интерфейсы с аппаратурой (такой как принтеры или сканеры) и интерфейсы с другими системами.

Чтобы найти граничные классы, надо исследовать диаграммы вариантов использования. Каждому взаимодействию между действующим лицом и вариантом использования должен соответствовать, по крайней мере, один граничный класс. Именно такой класс позволяет действующему лицу взаимодействовать с системой.

Классы-сущности

Классы-сущности (entity classes) содержат хранимую информацию. Они имеют наибольшее значение для пользователя, и потому в их названиях часто используют термины из предметной области. Обычно для каждого класса-сущности создают таблицу в базе данных.

Управляющие классы

Управляющие классы (control classes) отвечают за координацию действий других классов. Обычно у каждого варианта использования имеется один управляющий класс, контролирующий последовательность событий этого варианта использования. Управляющий класс отвечает за координацию, но сам не несет в себе никакой функциональности, так как остальные классы не посылают ему большого количества сообщений. Вместо этого он сам посылает множество сообщений. Управляющий класс просто делегирует ответственность другим классам, по этой причине его часто называют классом-менеджером.

В системе могут быть и другие управляющие классы, общие для нескольких вариантов использования. Например, может быть класс SecurityManager (менеджер безопасности), отвечающий за контроль событий, связанных с безопасностью. Класс TransactionManager (менеджер транзакций) занимается координацией сообщений, относящихся к транзакциям с базой данных. Могут быть и другие менеджеры для работы с другими элементами функционирования системы, такими как разделение ресурсов, распределенная обработка данных или обработка ошибок.

Помимо упомянутых выше стереотипов можно создавать и свои собственные.

Атрибуты

Атрибут – это элемент информации, связанный с классом. Атрибуты хранят инкапсулированные данные класса.

Так как атрибуты содержатся внутри класса, они скрыты от других классов. В связи с этим может понадобиться указать, какие классы имеют право читать и изменять атрибуты. Это свойство называется видимостью атрибута (attribute visibility).

У атрибута можно определить четыре возможных значения этого параметра:

- Public (общий, открытый). Это значение видимости предполагает, что атрибут будет виден всеми остальными классами. Любой класс может просмотреть или изменить значение атрибута. В соответствии с нотацией UML общему атрибуту предшествует знак « + ».
- Private (закрытый, секретный). Соответствующий атрибут не виден никаким другим классом. Закрытый атрибут обозначается знаком « – » в соответствии с нотацией UML.
- Protected (защищенный). Такой атрибут доступен только самому классу и его потомкам. Нотация UML для защищенного атрибута – это знак « # ».
- Package or Implementation (пакетный). Предполагает, что данный атрибут является общим, но только в пределах его пакета. Этот тип видимости не обозначается никаким специальным значком.

В общем случае, атрибуты рекомендуется делать закрытыми или защищенными. Это позволяет лучше контролировать сам атрибут и код.

С помощью закрытости или защищенности удается избежать ситуации, когда значение атрибута изменяется всеми классами системы. Вместо этого логика изменения атрибута будет заключена в том же классе, что и сам этот атрибут. Задаваемые параметры видимости повлияют на генерируемый код.

Операции

Операции реализуют связанное с классом поведение. Операция включает три части – имя, параметры и тип возвращаемого значения.

Параметры – это аргументы, получаемые операцией «на входе». Тип возвращаемого значения относится к результату действия операции.

На диаграмме классов можно показывать как имена операций, так и имена операций вместе с их параметрами и типом возвращаемого значения. Чтобы уменьшить загруженность диаграммы, полезно бывает на некоторых из них показывать только имена операций, а на других их полную сигнатуру.

В языке UML операции имеют следующую нотацию:

Имя Операции (аргумент: тип данных аргумента, аргумент2:тип данных аргумента2,...): тип возвращаемого значения

Следует рассмотреть четыре различных типа операций:

– Операции реализации;
– Операции управления;
– Операции доступа;
– Вспомогательные операции.

Операции реализации

Операции реализации (implementor operations) реализуют некоторые бизнес-функции. Такие операции можно найти, исследуя диаграммы взаимодействия. Диаграммы этого типа фокусируются на бизнес-функциях, и каждое сообщение диаграммы, скорее всего, можно соотнести с операцией реализации.

Каждая операция реализации должна быть легко прослеживаема до соответствующего требования. Это достигается на различных этапах моделирования. Операция выводится из сообщения на диаграмме взаимодействия, сообщения исходят из подробного описания потока событий, который создается на основе варианта использования, а последний – на основе требований. Возможность проследить всю эту цепочку позволяет гарантировать, что каждое требование будет реализовано в коде, а каждый фрагмент кода реализует какое-то требование.

Операции управления

Операции управления (manager operations) управляют созданием и уничтожением объектов. В эту категорию попадают конструкторы и деструкторы классов.

Операции доступа

Атрибуты обычно бывают закрытыми или защищенными. Тем не менее, другие классы иногда должны просматривать или изменять их значения. Для этого существуют операции доступа (access operations). Такой подход дает возможность безопасно

инкапсулировать атрибуты внутри класса, защитив их от других классов, но все же позволяет осуществить к ним контролируемый доступ. Создание операций Get и Set (получения и изменения значения) для каждого атрибута класса является стандартом.

Вспомогательные операции

Вспомогательными (helper operations) называются такие операции класса, которые необходимы ему для выполнения его ответственностей, но о которых другие классы не должны ничего знать. Это закрытые и защищенные операции класса.

Чтобы идентифицировать операции, выполните следующие действия:

1. Изучите диаграммы последовательности и кооперативные диаграммы. Большая часть сообщений на этих диаграммах является операциями реализации. Рефлексивные сообщения будут вспомогательными операциями.
2. Рассмотрите управляющие операции. Может потребоваться добавить конструкторы и деструкторы.
3. Рассмотрите операции доступа. Для каждого атрибута класса, с которым должны будут работать другие классы, надо создать операции Get и Set.

Связи

Связь представляет собой семантическую взаимосвязь между классами. Она дает классу возможность узнавать об атрибутах, операциях и связях другого класса. Иными словами, чтобы один класс мог послать сообщение другому на диаграмме последовательности или кооперативной диаграмме, между ними должна существовать связь.

Существуют четыре типа связей, которые могут быть установлены между классами: ассоциации, зависимости, агрегации и обобщения.

Ассоциации

Ассоциация (association) – это семантическая связь между классами. Их рисуют на диаграмме классов в виде обыкновенной линии.

Рис. 10. Связь ассоциация

Ассоциации могут быть двунаправленными, как в примере, или однонаправленными. На языке UML двунаправленные ассоциации рисуют в виде простой линии без стрелок или со стрелками с обеих ее сторон. На однонаправленной ассоциации изображают только одну стрелку, показывающую ее направление.

Направление ассоциации можно определить, изучая диаграммы последовательности и кооперативные диаграммы. Если все сообщения на них отправляются только одним классом и принимаются только другим классом, но не наоборот, между этими классами имеет место однонаправленная связь. Если хотя бы одно сообщение отправляется в обратную сторону, ассоциация должна быть двунаправленной.

Ассоциации могут быть рефлексивными. Рефлексивная ассоциация предполагает, что один экземпляр класса взаимодействует с другими экземплярами этого же класса.

Зависимости

Связи зависимости (dependency) также отражают связь между классами, но они всегда однонаправлены и показывают, что один класс зависит от определений, сделанных в другом. Например, класс A использует методы класса B. Тогда при изменении класса B необходимо произвести соответствующие изменения в классе A.

Зависимость изображается пунктирной линией, проведенной между двумя элементами диаграммы, и считается, что элемент, привязанный к концу стрелки, зависит от элемента, привязанного к началу этой стрелки.

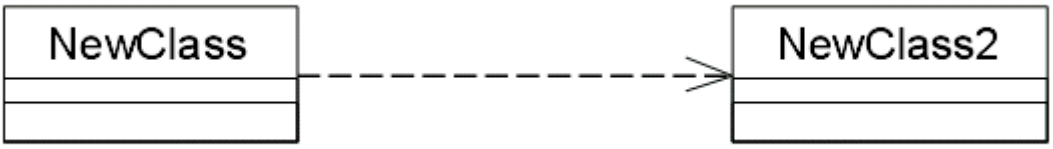


Рис. 11. Связь зависимость

При генерации кода для этих классов к ним не будут добавляться новые атрибуты. Однако, будут созданы специфические для языка операторы, необходимые для поддержки связи.

Агрегации

Агрегации (aggregations) представляют собой более тесную форму ассоциации. Агрегация – это связь между целым и его частью. Например, у вас может быть класс Автомобиль, а также классы Двигатель, Покрышки и классы для других частей автомобиля. В результате объект класса Автомобиль будет состоять из объекта класса Двигатель, четырех объектов Покрышек и т. д. Агрегации визуализируют в виде линии с ромбиком у класса, являющегося целым:

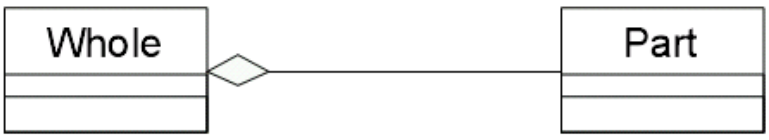


Рис. 11. Связь агрегация

В дополнение к простой агрегации UML вводит более сильную разновидность агрегации, называемую композицией. Согласно композиции, объект-часть может принадлежать только единственному целому, и, кроме того, как правило, жизненный цикл частей совпадает с циклом целого: они живут и умирают вместе с ним. Любое удаление целого распространяется на его части.

Такое каскадное удаление нередко рассматривается как часть определения агрегации, однако оно всегда подразумевается в том случае, когда множественность роли составляет 1..1; например, если необходимо удалить Клиента, то это удаление должно распространиться и на Заказы (и, в свою очередь, на Строки заказа).

Обобщения (Наследование)

Обобщение (наследование) - это отношение типа общее-частное между элементами модели. С помощью обобщений (generalization) показывают связи наследования между двумя классами. Большинство объектно-ориентированных языков непосредственно поддерживают концепцию наследования. Она позволяет одному классу наследовать все атрибуты, операции и связи другого. Наследование пакетов означает, что в пакете-наследнике все сущности пакета-предка будут видны под своими собственными именами (т.е. пространства имен объединяются). Наследование показывается сплошной линией, идущей от класса-потомка к классу-предку (в терминологии ООП - от потомка к предку, от сына к отцу, или от подкласса к суперклассу). Со стороны более общего элемента рисуется большой полый треугольник.

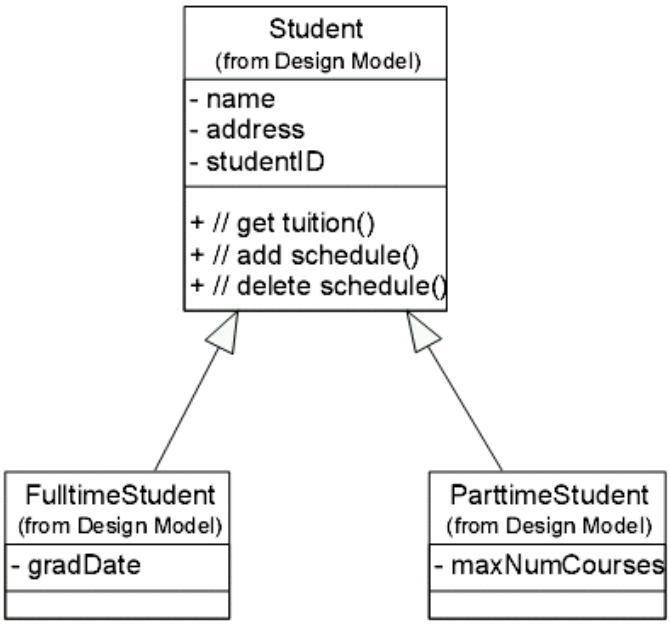


Рис. 12. Пример связи наследование

Помимо наследуемых, каждый подкласс имеет свои собственные уникальные атрибуты, операции и связи.

Множественность

Множественность (multiplicity) показывает, сколько экземпляров одного класса взаимодействуют с помощью этой связи с одним экземпляром другого класса в данный момент времени.

Например, при разработке системы регистрации курсов в университете можно определить классы Course (курс) и Student (студент). Между ними установлена связь: у курсов могут быть студенты, а у студентов – курсы. Вопросы, на который должен ответить параметр множественности: «Сколько курсов студент может посещать в данный момент? Сколько студентов может за раз посещать один курс?»

Так как множественность дает ответ на оба эти вопроса, её индикаторы устанавливаются на обоих концах линии связи. В примере регистрации курсов мы решили, что один студент может посещать от нуля до четырех курсов, а один курс могут слушать от 0 до 20 студентов.

В языке UML приняты определенные нотации для обозначения множественности.

Таблица 1. Обозначения множественности связей в UML

Множественность	Значение
0..*	Ноль или больше
1..*	Один или больше
0..1	Ноль или один
1..1 (сокращенная запись: 1)	Ровно один

Имена связей

Связи можно уточнить с помощью имен связей или ролевых имен. Имя связи – это обычно глагол или глагольная фраза, описывающая, зачем она нужна. Например, между классом Person (человек) и классом Company (компания) может существовать ассоциация. Можно задать в связи с этим вопрос, является ли объект класса Person клиентом компании, её сотрудником или владельцем? Чтобы определить это, ассоциацию можно назвать «employs» (нанимает):

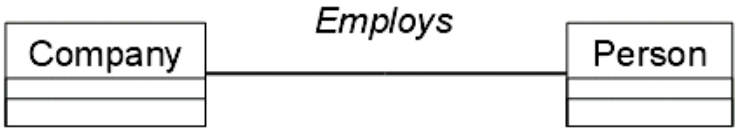


Рис. 13. Пример имен связей

Роли

Ролевые имена применяют в связях ассоциации или агрегации вместо имен для описания того, зачем эти связи нужны. Возвращаясь к примеру с классами Person и Company, можно сказать, что класс Person играет роль сотрудника класса Company. Ролевые имена – это обычно имена существительные или основанные на них фразы, их показывают на диаграмме рядом с классом, играющим соответствующую роль. Как правило, пользуются или ролевым именем, или именем связи, но не обоими сразу. Как и имена связей, ролевые имена не обязательны, их дают, только если цель связи не очевидна. Пример ролей приводится ниже:

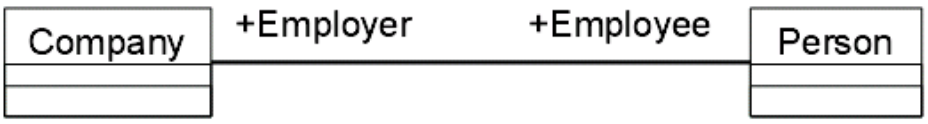


Рис. 14. Пример ролей связей

Пакет. Механизм пакетов

В контексте диаграмм классов, пакет - это вместилище для некоторого набора классов и других пакетов. Пакет является самостоятельным пространством имен.

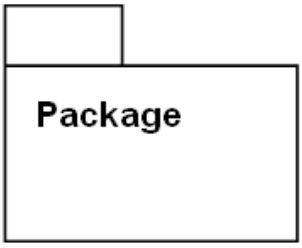


Рис. 15. Обозначение пакета в UML

В UML нет каких-либо ограничений на правила, по которым разработчики могут или должны группировать классы в пакеты. Но есть некоторые стандартные случаи, когда такая группировка уместна, например, тесно взаимодействующие классы, или более общий случай - разбиение системы на подсистемы.

Пакет физически содержит сущности, определенные в нем (говорят, что "сущности принадлежат пакету"). Это означает, что если будет уничтожен пакет, то будут уничтожено и все его содержимое.

Существует несколько наиболее распространенных подходов к группировке.

Во-первых, можно группировать их по стереотипу. В таком случае получается один пакет с классами-сущностями, один с граничными классами, один с управляющими классами и т.д. Этот подход может быть полезен с точки зрения размещения готовой системы, поскольку все находящиеся на клиентских машинах пограничные классы уже оказываются в одном пакете.

Другой подход заключается в объединении классов по их функциональности. Например, в пакете Security (безопасность) содержатся все классы, отвечающие за безопасность приложения. В таком случае другие пакеты могут называться Employee Maintenance (Работа с сотрудниками), Reporting (Подготовка отчетов) и Error Handling (Обработка ошибок). Преимущество этого подхода заключается в возможности повторного использования.

Механизм пакетов применим к любым элементам модели, а не только к классам. Если для группировки классов не использовать некоторые эвристики, то она становится произвольной. Одна из них, которая в основном используется в UML, – это зависимость. Зависимость между двумя пакетами существует в том случае, если между любыми двумя классами в пакетах существует любая зависимость.

Таким образом, диаграмма пакетов представляет собой диаграмму, содержащую пакеты классов и зависимости между ними. Строго говоря, пакеты и зависимости являются элементами диаграммы классов, то есть диаграмма пакетов – это форма диаграммы классов.

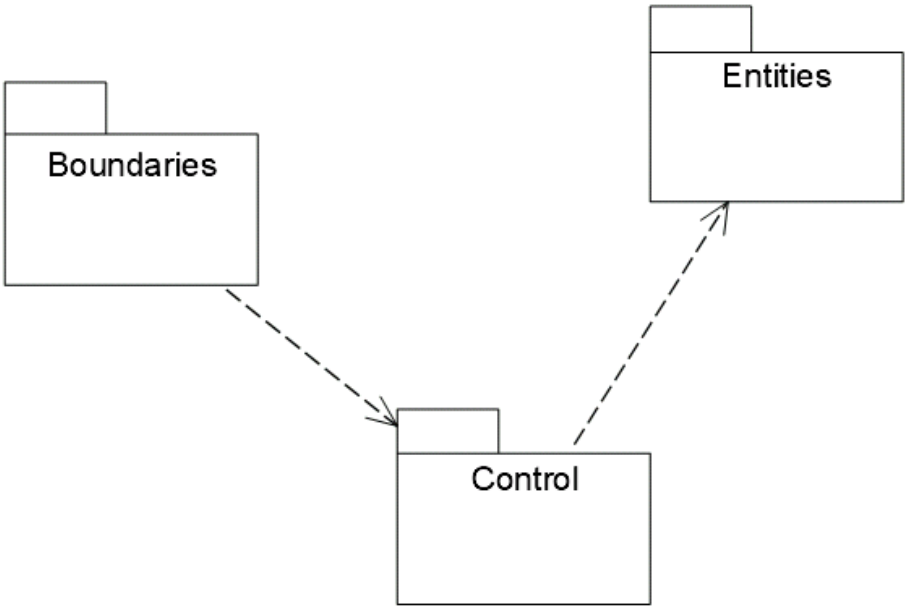


Рис. 16. Пример диаграммы пакетов

Зависимость между двумя элементами имеет место в том случае, если изменения в определении одного элемента могут повлечь за собой изменения в другом. Что касается классов, то причины для зависимостей могут быть самыми разными:

– один класс посылает сообщение другому;
– один класс включает часть данных другого класса; один класс использует другой в качестве параметра операции.

Если класс меняет свой интерфейс, то любое сообщение, которое он посылает, может утратить свою силу.

Пакеты не дают ответа на вопрос, каким образом можно уменьшить количество зависимостей в вашей системе, однако они помогают выделить эти зависимости, а после того, как они все окажутся на виду, остается только поработать над снижением их количества. Диаграммы пакетов можно считать основным средством управления общей структурой системы.

Пакеты являются жизненно необходимым средством для больших проектов. Их следует использовать в тех случаях, когда диаграмма классов, охватывающая всю систему в целом и размещенная на единственном листе бумаги формата А4, становится нечитаемой.

Диаграммы состояний

Диаграммы состояний определяют все возможные состояния, в которых может находиться конкретный объект, а также процесс смены состояний объекта в результате наступления некоторых событий.

Существует много форм диаграмм состояний, незначительно отличающихся друг от друга семантикой.

На диаграмме имеются два специальных состояния – начальное (start) и конечное (stop). Начальное состояние выделено черной точкой, оно соответствует состоянию объекта, когда он только что был создан. Конечное состояние обозначается черной

точкой в белом кружке, оно соответствует состоянию объекта непосредственно перед его уничтожением. На диаграмме состояний может быть одно и только одно начальное состояние. В то же время, может быть столько конечных состояний, сколько вам нужно, или их может не быть вообще. Когда объект находится в каком-то конкретном состоянии, могут выполняться различные процессы. Процессы, происходящие, когда объект находится в определенном состоянии, называются действиями (actions).

С состоянием можно связывать данные пяти типов: деятельность, входное действие, выходное действие, событие и история состояния.

Деятельность

Деятельностью (activity) называется поведение, реализуемое объектом, пока он находится в данном состоянии. Деятельность – это прерываемое поведение. Оно может выполняться до своего завершения, пока объект находится в данном состоянии, или может быть прервано переходом объекта в другое состояние. Деятельность изображают внутри самого состояния, ей должно предшествовать слово do (делать) и двоеточие.

Входное действие

Входным действием (entry action) называется поведение, которое выполняется, когда объект переходит в данное состояние. Данное действие осуществляется не после того, как объект перешел в это состояние, а, скорее, как часть этого перехода. В отличие от деятельности, входное действие рассматривается как непрерываемое. Входное действие также показывают внутри состояния, ему предшествует слово entry (вход) и двоеточие.

Выходное действие

Выходное действие (exit action) подобно входному. Однако, оно осуществляется как составная часть процесса выхода из данного состояния. Оно является частью процесса такого перехода. Как и входное, выходное действие является непрерываемым.

Выходное действие изображают внутри состояния, ему предшествует слово exit (выход) и двоеточие.

Поведение объекта во время деятельности, при входных и выходных действиях может включать отправку события другому объекту. В этом случае описанию деятельности, входного действия или выходного действия предшествует знак « ^ ».

Соответствующая строка на диаграмме выглядит как

Do: ^Цель.Событие (Аргументы)

Здесь Цель – это объект, получающий событие, Событие – это посылаемое сообщение, а Аргументы являются параметрами посылаемого сообщения.

Деятельность может также выполняться в результате получения объектом некоторого события. При получении некоторого события выполняется определенная деятельность.

Переходом (Transition) называется перемещение из одного состояния в другое. Совокупность переходов диаграммы показывает, как объект может перемещаться

между своими состояниями. На диаграмме все переходы изображают в виде стрелки, начинающейся на первоначальном состоянии и заканчивающейся последующим.

Переходы могут быть рефлексивными. Объект может перейти в то же состояние, в котором он в настоящий момент находится. Рефлексивные переходы изображают в виде стрелки, начинающейся и завершающейся на одном и том же состоянии.

У перехода существует несколько спецификаций. Они включают события, аргументы, ограждающие условия, действия и посылаемые события.

События

Событие (event) – это то, что вызывает переход из одного состояния в другое. Событие размещают на диаграмме вдоль линии перехода.

На диаграмме для отображения события можно использовать как имя операции, так и обычную фразу.

Большинство переходов должны иметь события, так как именно они, прежде всего, заставляют переход осуществиться. Тем не менее, бывают и автоматические переходы, не имеющие событий. При этом объект сам перемещается из одного состояния в другое со скоростью, позволяющей осуществиться входным действиям, деятельности и выходным действиям.

Ограждающие условия

Ограждающие условия (guard conditions) определяют, когда переход может, а когда не может осуществиться. В противном случае переход не осуществится.

Ограждающие условия изображают на диаграмме вдоль линии перехода после имени события, заключая их в квадратные скобки.

Ограждающие условия задавать необязательно. Однако если существует несколько автоматических переходов из состояния, необходимо определить для них взаимно исключающие ограждающие условия. Это поможет читателю диаграммы понять, какой путь перехода будет автоматически выбран.

Действие

Действием (action), как уже говорилось, является непрерываемое поведение, осуществляющееся как часть перехода. Входные и выходные действия показывают внутри состояний, поскольку они определяют, что происходит, когда объект входит или выходит из него. Большую часть действий, однако, изображают вдоль линии перехода, так как они не должны осуществляться при входе или выходе из состояния.

Действие рисуют вдоль линии перехода после имени события, ему предшествует косая черта.

Событие или действие могут быть поведением внутри объекта, а могут представлять собой сообщение, посылаемое другому объекту. Если событие или действие посылается другому объекту, перед ним на диаграмме помещают знак « ^ ».

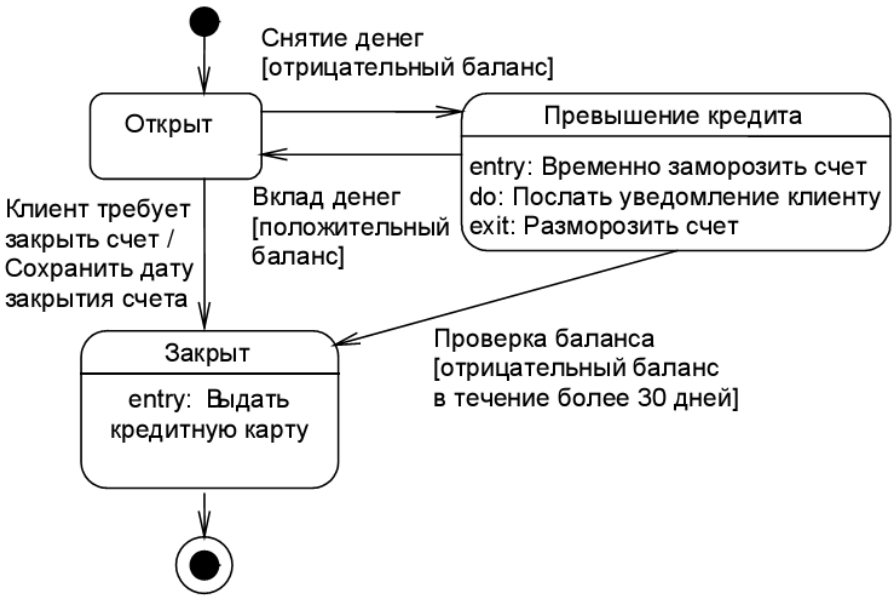


Рис. 17. Пример диаграммы состояний

Диаграммы состояний не надо создавать для каждого класса, они применяются только в сложных случаях. Если объект класса может существовать в нескольких состояниях и в каждом из них ведет себя по-разному, для него может потребоваться такая диаграмма.

Диаграммы размещения

Диаграмма размещения (deployment diagram) отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать маршруты перемещения объектов и компонентов в распределенной системе.

Каждый узел на диаграмме размещения представляет собой некоторый тип вычислительного устройства – в большинстве случаев, часть аппаратуры. Эта аппаратура может быть простым устройством или датчиком, а может быть и мэйнфреймом.

Диаграмма размещения показывает физическое расположение сети и местонахождение в ней различных компонентов.

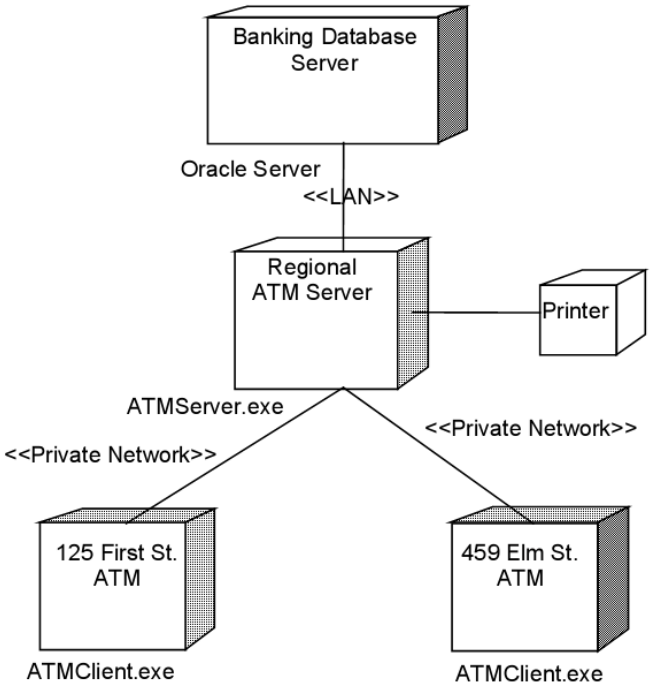


Рис. 19. Пример диаграммы размещения

Диаграмма размещения используется менеджером проекта, пользователями, архитектором системы и эксплуатационным персоналом, чтобы понять физическое размещение системы и расположение её отдельных подсистем.

Диаграммы компонентов

Диаграммы компонентов показывают, как выглядит модель на физическом уровне. На них изображены компоненты программного обеспечения и связи между ними. При этом на такой диаграмме выделяют два типа компонентов: исполняемые компоненты и библиотеки кода.

Каждый класс модели (или подсистема) преобразуется в компонент исходного кода. После создания они сразу добавляются к диаграмме компонентов. Между отдельными компонентами изображают зависимости, соответствующие зависимостям на этапе компиляции или выполнения программы.

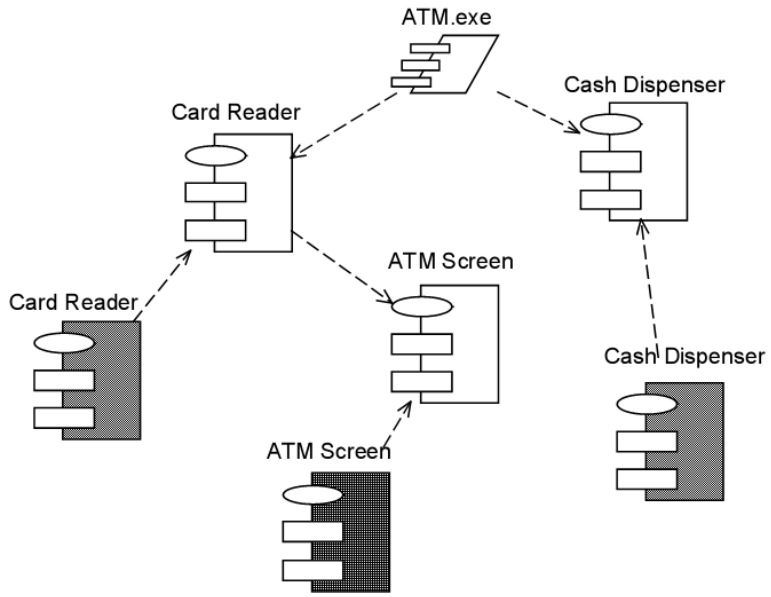


Рис. 18. Пример диаграммы компонентов

Диаграммы компонентов применяются теми участниками проекта, кто отвечает за компиляцию системы. Из нее видно, в каком порядке надо компилировать компоненты, а также какие исполняемые компоненты будут созданы системой. На такой диаграмме показано соответствие классов реализованным компонентам. Она нужна там, где начинается генерация кода.

Объединение диаграмм компонентов и развертывания

В некоторых случаях допускается размещать диаграмму компонентов на диаграмме развертывания. Это позволяет показать какие компоненты выполняются и на каких узлах.

4. Порядок выполнения работы

1. Изучить предлагаемый теоретический материал.
2. Постройте диаграмму вариантов использования для выбранной информационной системы.
3. Выполните реализацию вариантов использования в терминах взаимодействующих объектов и представляющую собой набор диаграмм:
   - диаграмм классов, реализующих вариант использования;
   - диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм), отражающих взаимодействие объектов в процессе реализации варианта использования.
4. Разделить классы по пакетам использую один из механизм разбиения.
5. Постройте диаграмму состояний для конкретных объектов информационной системы.
6. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

5. Содержание отчета

В отчете следует указать:

1. Цель работы

2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума (п.2).
5. Заключение (выводы)
6. Список используемой литературы

6. Литература:

1. http://www.uml.org
2. http://www.uml.ru
3. http://www.uml2.ru
4. http://www.omg.org/technology/documents/formal/uml.htm
5. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. – С-П.: Издательство «Питер», 2003. – 432 с.
6. Шмуллер Дж. Освой самостоятельно UML 2 за 24 часа. Практическое руководство. - М.: «Вильямс», 2005. - 416 с.

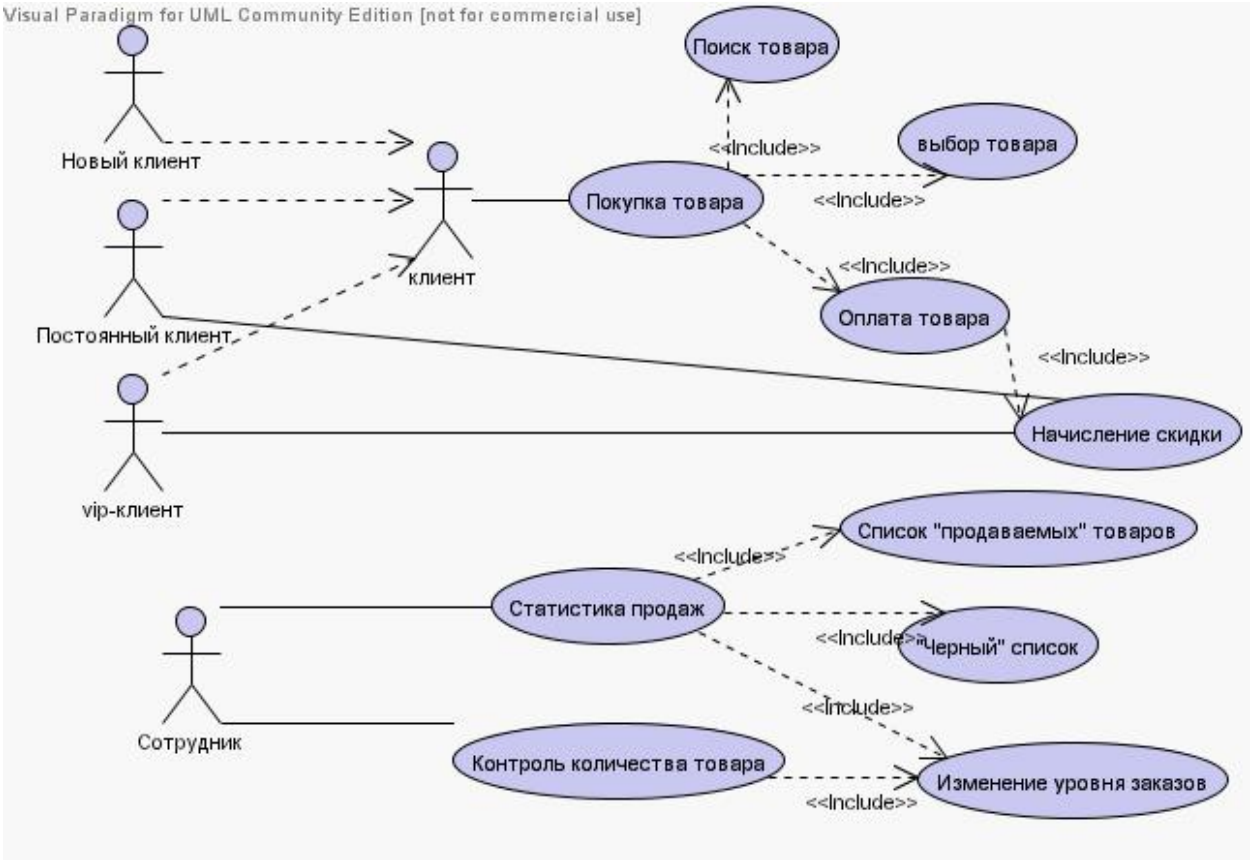
7. Контрольные вопросы

1. Дайте определение понятию «вариант использования».
2. Какие типы связи могут присутствовать на диаграмме вариантов использования?
3. Дайте определение понятию «действующее лицо».
4. Какие типы сообщений могут присутствовать на диаграммах взаимодействия?
5. Дайте определение понятию класс, объект класса.
6. Кем и для чего может быть использована диаграмма размещения?

Приложение А

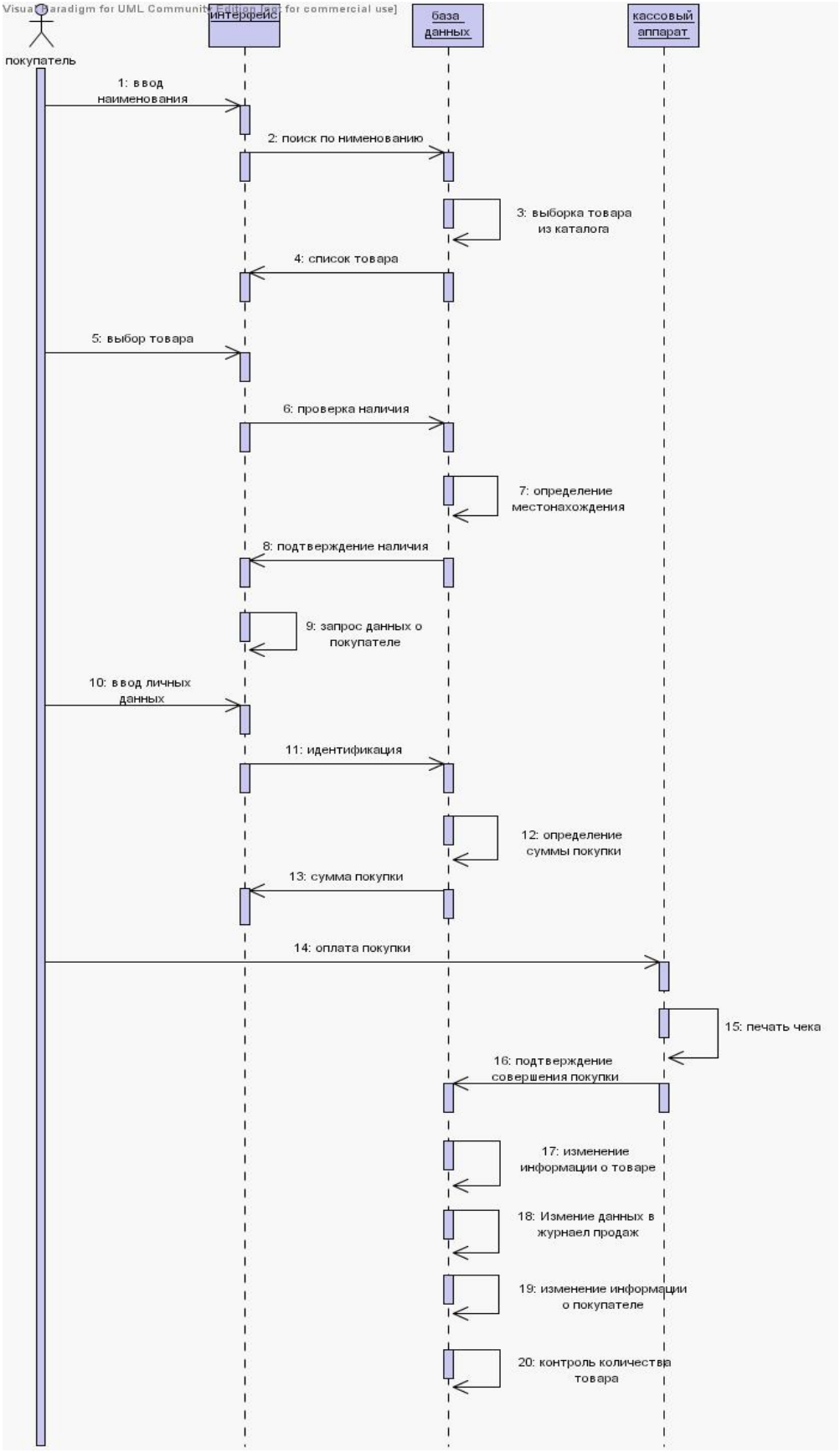
Пример построения объектной модели информационной системы «Обеспечение продаж магазина».

Диаграмма Вариантов использования

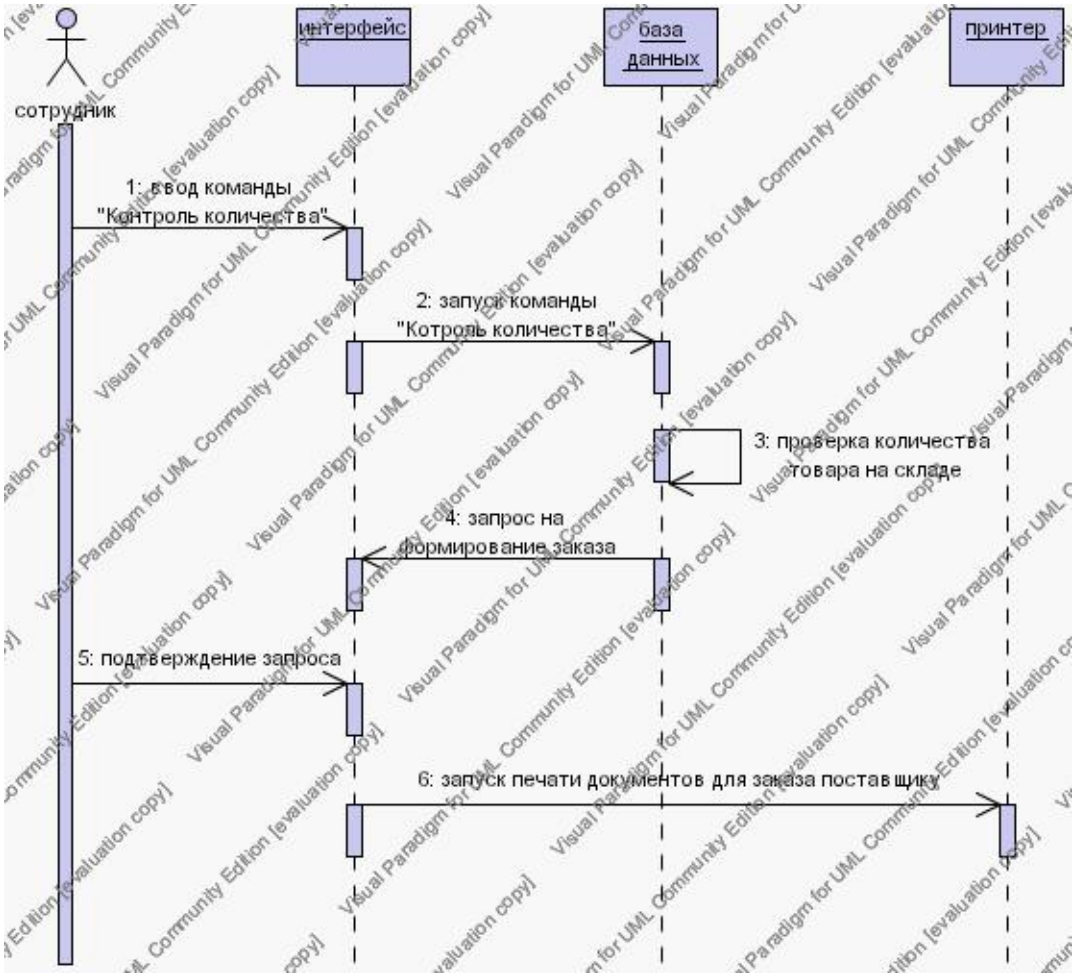


Данная диаграмма отражает взаимодействие между клиентом (постоянным клиентом и vip- клиентом) и системой, а также взаимодействие между сотрудником организации и системой. Клиент инициализирует процесс покупки товара. В этот процесс обычно включены такие процессы, как поиск товара, выбор товара, оплата товара. Сотрудник инициализирует запуск процессов «статистика продаж» и «контроль количества товара». В их результате запускается процесс «изменение уровня заказов».

Диаграмма Последовательностей



Данная диаграмма отражает поток событий, происходящий в рамках первого варианта использования (клиент и система).



Данная диаграмма отражает поток событий, происходящий в рамках второго варианта использования (сотрудник и система).

Диаграмма Классов

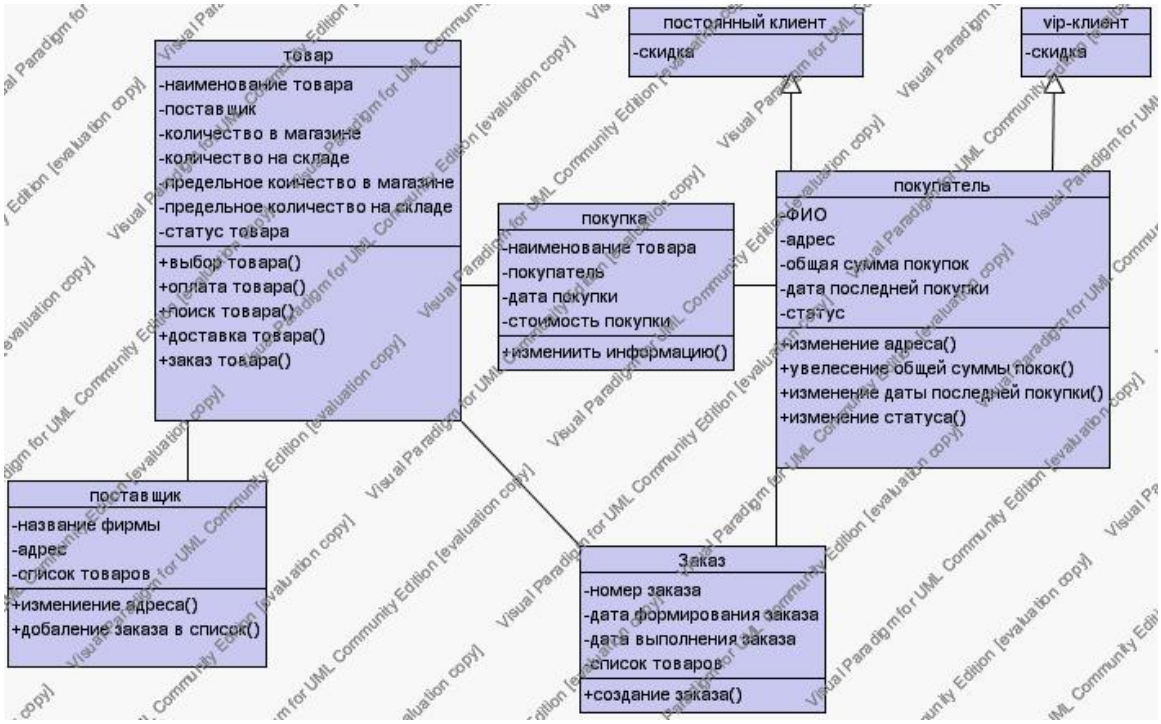
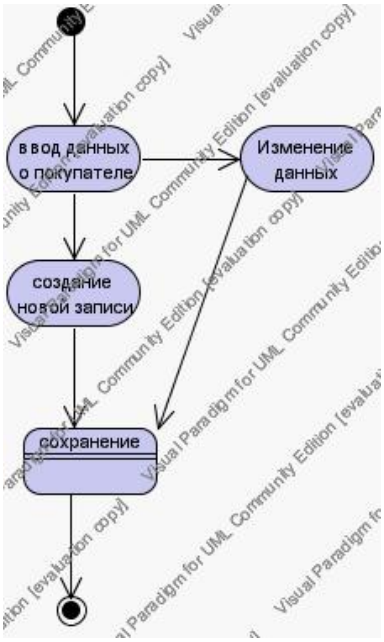


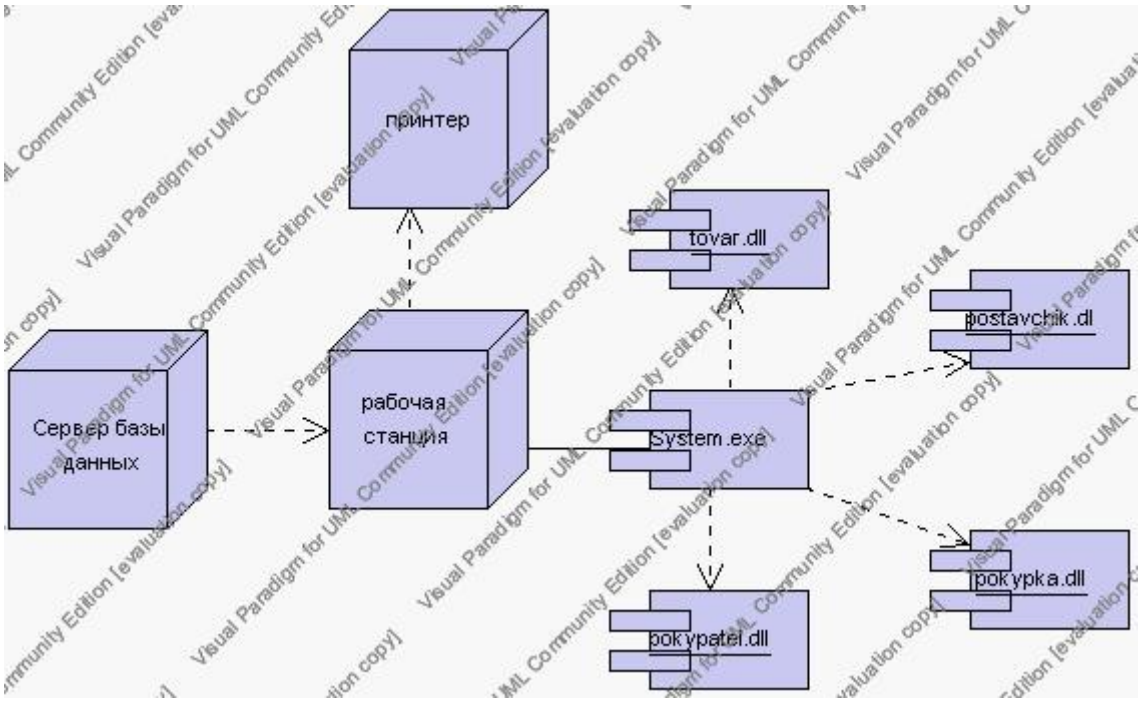
Диаграмма классов реализует полный функционал системы.

Диаграмма Состояний



Данная диаграмма описывает ситуацию, когда при вводе новых данных объект либо уже существует, либо еще не создан. Объект класса ведет себя иначе в различных состояниях, то может потребоваться такая диаграмма.

Диаграмма компонентов и размещений



Лабораторная работа № 5

«Методология управление проектами»

1. Цель работы:

Изучение методологии управления проектами. Получение навыков по применению данных методологий для планирования проекта.

2. Методические указания

Лабораторная работа направлена на ознакомление с основными понятиями методологии управления проектами, получение навыков по применению данных понятий при построении плана проекта, построения графика работ, распределения исполнителей, управления рисками.

Требования к результатам выполнения лабораторного практикума:

– Построить модель управления проектом, включающую:
  - определение всех этапов проекта, зависимых этапов, определение длительности этапов;
  - построение на основе полученных данных сетевой и временной диаграмм;
  - построение диаграммы распределения работников по этапам;
– при определении этапа указывается его название – отражающее суть этапа (например, определение пользовательских требований, проектирование интерфейса и т.д.);
– этапов должно быть не менее 7, срок реализации проекта – с 1.09 по 31.12;
– в проекте задействовано 3 человек персонала (группа разработчиков).

При составлении и оформлении отчета следует придерживаться рекомендаций, представленных на странице http://unesco.kemsu.ru/student/rule/rule.html.

3. Теоретический материал

Основные понятия

Проблемы управления программными проектами впервые проявились в 60-х — начале 70-х годов. Руководители программных проектов выполняют такую же работу, что и руководители технических проектов. Вместе с тем процесс разработки ПО существенно отличается от процессов реализации технических проектов, что порождает определенные сложности в управлении программными проектами. Ниже приведён краткий список этих отличий.

1. *Программный продукт нематериален.* Менеджер технического проекта видит результаты выполнения своего проекта. Если реализация проекта отстает от графика, это также видно воочию. В противоположность этому программное обеспечение нематериально. Его нельзя увидеть или потрогать. Менеджер программного проекта не видит процесс "роста" разрабатываемого ПО. Он может полагаться только на документацию, которая фиксирует процесс разработки программного продукта.

2. *Не существует стандартных процессов разработки ПО.* На сегодняшний день не существует четкой зависимости между процессом создания ПО и типом создаваемого программного продукта. Другие технические дисциплины имеют длительную историю, процессы разработки технических изделий многократно опробованы и проверены. Процессы создания большинства технических систем хорошо изучены. Изучением же процессов создания ПО специалисты занимаются только несколько последних лет. Поэтому пока нельзя точно предсказать, на каком этапе процесса разработки ПО могут возникнуть проблемы, угрожающие всему программному проекту.

3. *Большие программные проекты - это часто "одноразовые" проекты.* Большие программные проекты, как правило, значительно отличаются от проектов, реализованных ранее. Поэтому, чтобы уменьшить неопределенность в планировании проекта, руководители проектов должны обладать очень большим практическим опытом. Но постоянные технологические изменения в компьютерной технике и коммуникационном оборудовании обесценивают предыдущий опыт. Знания и навыки, накопленные опытом, могут не востребоваться в новом проекте.

Перечисленное выше может привести к тому, что реализация проекта выйдет из временного графика или превысит бюджетные ассигнования. Программные системы зачастую оказываются новинками как в "идеологическом", так и в техническом плане. Технические проекты, которые являются инновационными (например, новая транспортная система), также часто нарушают временные графики работ. Поэтому, предвидя возможные проблемы в реализации программного проекта, следует всегда помнить, что многим из них свойственно выходить за рамки временных и бюджетных ограничений.

Планирование проекта

Эффективное управление программным проектом напрямую зависит от правильного планирования работ, необходимых для его выполнения. План помогает менеджеру предвидеть проблемы, которые могут возникнуть на каких-либо этапах создания ПО, и разработать превентивные меры для их предупреждения или решения. План, разработанный на начальном этапе проекта, рассматривается всеми его участниками как руководящий документ, выполнение которого должно привести к успешному завершению проекта. Этот первоначальный план должен максимально подробно описывать все этапы реализации проекта.

Кроме разработки плана проекта, на менеджера ложится обязанность разработки других видов планов. Эти виды планов кратко описаны в табл. 1.

Таблица 1. Виды планов

План	**Описание**
План качества	Описывает стандарты и мероприятия по поддержке качества разрабатываемого ПО
План аттестации	Описывает способы, ресурсы и перечень работ, необходимых для аттестации программной системы
План управления конфигурацией	Описывает структуру и процессы управления конфигурацией
План	Предлагает план мероприятий, требующихся для

сопровождения ПО	сопровождения ПО в процессе его эксплуатации, а также расчет стоимости сопровождения и необходимые для этого ресурсы
План по управлению персоналом	Описывает мероприятия, направленные на повышение квалификации членов команды разработчиков

В листинге 1 показан процесс планирования создания ПО в виде псевдокода. Здесь сделан акцент на том, что планирование — это итерационный процесс. Поскольку в процессе выполнения проекта постоянно поступает новая информация, план должен регулярно пересматриваться. Важными факторами, которые должны учитываться при разработке плана, являются финансовые и деловые обязательства организации. Если они изменяются, эти изменения также должны найти отражение в плане работ.

Листинг 1. Процесс планирования проекта

```
Определение проектных ограничений
Первоначальная оценка параметров проекта
Определение этапов выполнения проекта и контрольных отметок
while пока проект не завершится или не будет остановлен loop
    Составление графика работ
    Начало выполнения работ
    Ожидание окончания очередного этапа работ
    Отслеживание хода выполнения работ
    Пересмотр оценок параметров проекта
    Изменение графика работ
    Пересмотр проектных ограничений
    if (возникла проблема) then
    Пересмотр технических или организационных параметров проекта
    end if
end loop
```

Процесс планирования начинается с определения проектных ограничений (временные ограничения, возможности наличного персонала, бюджетные ограничения и т.д.). Эти ограничения должны определяться параллельно с оцениванием проектных параметров, таких как структура и размер проекта, а также распределением функций среди исполнителей. Затем определяются этапы разработки и то, какие результаты документация, прототипы, подсистемы или версии программного продукта) должны быть получены по окончании этих этапов. Далее начинается циклическая часть планирования. Сначала разрабатывается график работ по выполнению проекта или дается разрешение на продолжение использования ранее созданного графика. После этого (обычно через 2-3

недели) проводится контроль выполнения работ и отмечаются расхождения между реальным и плановым ходом работ.

Далее, по мере поступления новой информации о ходе выполнения проекта, возможен пересмотр первоначальных оценок параметров проекта. Это, в свою очередь, может привести к изменению графика работ. Если в результате этих изменений нарушаются сроки завершения проекта, должны быть пересмотрены (и согласованы с заказчиком ПО) проектные ограничения.

Конечно, большинство менеджеров проектов не думают, что реализация их проектов пройдет гладко, без всяких проблем. Желательно описать возможные проблемы еще до того, как они проявят себя в ходе выполнения проекта. Поэтому лучше составлять "пессимистические" графики работ, чем "оптимистические". Но, конечно, невозможно построить план, учитывающий все, в том числе случайные, проблемы и задержки выполнения проекта, поэтому и возникает необходимость периодического пересмотра проектных ограничений и этапов создания программного продукта.

План проекта

План проекта должен четко показать ресурсы, необходимые для реализации проекта, разделение работ на этапы и временной график выполнения этих этапов. В некоторых организациях план проекта составляется как единый документ, содержащий все виды планов, описанных выше. В других случаях план проекта описывает только технологический процесс создания ПО. В таком плане обязательно присутствуют ссылки на планы других видов, но они разрабатываются отдельно от плана проекта.

План, представленны ниже, принадлежит именно к последнему типу планов. Детализация планов проектов очень разнится в зависимости от типа разрабатываемого программного продукта и организации-разработчика. Но в любом случае большинство планов содержат следующие разделы.

1. *Введение.* Краткое описание целей проекта и проектных ограничений (бюджетных, временны*х* и т.д.), которые важны для управления проектом.
2. *Организация выполнения проекта.* Описание способа подбора команды разработчиков и распределение обязанностей между членами команды.
3. *Анализ рисков.* Описание возможных проектных рисков, вероятности их проявления и стратегий, направленных на их уменьшение. Тема управления рисками рассмотрена ниже.
4. *Аппаратные и программные ресурсы, необходимые для реализации проекта.* Перечень аппаратных средств и программного обеспечения, необходимого для разработки программного продукта. Если аппаратные средства требуется закупать, приводится их стоимость совместно с графиком закупки и поставки.
5. *Разбиение работ на этапы.* Процесс реализации проекта разбивается на отдельные процессы, определяются этапы выполнения проекта, приводится описание результатов ("выходов") каждого этапа и контрольные отметки. Эта тема представлена ниже.
6. *График работ.* В этом графике отображаются зависимости между отдельными процессами (этапами) разработки ПО, оценки времени их выполнения и распределение членов команды разработчиков по отдельным этапам.
7. *Механизмы мониторинга и контроля за ходом выполнения проекта.* Описываются предоставляемые менеджером отчеты о ходе выполнения работ, сроки их предоставления, а также механизмы мониторинга всего проекта.

План должен регулярно пересматриваться в процессе реализации проекта. Одни части плана, например график работ, изменяются часто, другие более стабильны. Для внесения изменений в план требуется специальная организация документопотока, позволяющая отслеживать эти изменения.

Контрольные отметки этапов работ

Менеджеру для организации процесса создания ПО и управления им необходима информация. Поскольку само программное обеспечение неосязаемо, эта управленческая информация может быть получена только в виде документов, отображающих выполнение очередного этапа разработки программного продукта. Без этой информации нельзя судить о степени готовности создаваемого продукта, невозможно оценить произведенные затраты или изменить график работ.

При планировании процесса определяются контрольные отметки— вехи, отмечающие окончание определенного этапа работ. Для каждой контрольной отметки создается отчет, который предоставляется руководству проекта. Эти отчеты не должны быть большими объемными документами; они должны подводить краткие итоги окончания отдельного логически завершенного этапа проекта. Этапом не может быть, например, "Написание 80% кода программ", поскольку невозможно проверить завершение такого "этапа"; кроме того, подобная информация практически бесполезна для управления, поскольку здесь не отображается связь этого "этапа" с другими этапами создания ПО.

Обычно по завершении основных больших этапов, таких как разработка спецификации, проектирование и т.п., заказчику ПО предоставляются результаты их выполнения, так называемые контрольные проектные элементы. Это может быть документация, прототип программного продукта, законченные подсистемы ПО и т.д. Контрольные проектные элементы, предоставляемые заказчику ПО, могут совпадать с контрольными отметками (точнее, с результатами выполнения какого-либо этапа). Но обратное утверждение неверно. Контрольные отметки — это внутренние проектные результаты, которые используются для контроля за ходом выполнения проекта, и они, как правило, не предоставляются заказчику ПО.

Для определения контрольных отметок весь процесс создания ПО должен быть разбит на отдельные этапы с указанным "выходом" (результатом) каждого этапа. Например, на рис. 1 показаны этапы разработки спецификации требований в случае, когда для ее проверки используется прототип системы, а также представлены выходные результаты (контрольные отметки) каждого этапа. Здесь контрольными проектными элементами являются требования и спецификация требований.

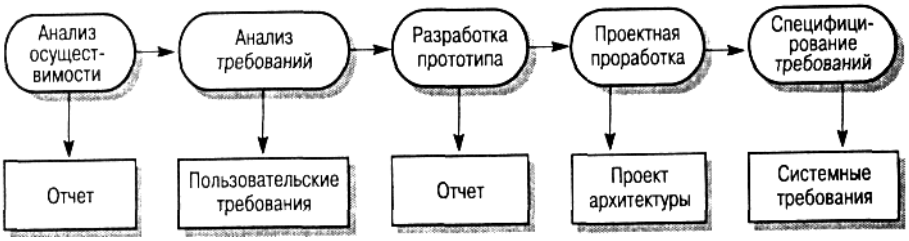


Рис. 1. Этапы процесса разработки спецификации

График работ

Составление графика - одна из самых ответственных работ, выполняемых менеджером проекта. Здесь менеджер оценивает длительность проекта, определяет ресурсы, необходимые для реализации отдельных этапов работ, и представляет их (этапы) в виде согласованной последовательности. Если данный проект подобен ранее реализованному, то график работ последнего проекта можно взять за основу для данного проекта. Но затем следует учесть, что на отдельных этапах нового проекта могут использоваться методы и подходы, отличные от использованных ранее.

Если проект является инновационным, первоначальные оценки длительности и требуемых ресурсов наверняка будут слишком оптимистичными, даже если менеджер попытается предусмотреть все возможные неожиданности. С этой точки зрения проекты создания ПО не отличаются от больших инновационных технических проектов. Новые аэропорты, мосты и даже новые автомобили, как правило, появляются позже первоначально объявленных сроков их сдачи или поступления на рынок, чему причиной являются неожиданно возникшие проблемы и трудности. Именно поэтому графики работ необходимо постоянно обновлять по мере поступления новой информации о ходе выполнения проекта.

В процессе составления графика (рис. 2) весь массив работ, необходимых для реализации проекта, разбивается на отдельные этапы и оценивается время, требующееся для выполнения каждого этапа. Обычно многие этапы выполняются параллельно. График работ должен предусматривать это и распределять производственные ресурсы между ними наиболее оптимальным образом. Нехватка ресурсов для выполнения какого-либо критического этапа - частая причина задержки выполнения всего проекта.

Длительность этапов обычно должна быть не меньше недели. Если она будет меньше, то окажется ниже точности временных оценок этапов, что может привести к частому пересмотру графика работ. Также целесообразно (в аспекте управления проектом) установить максимальную длительность этапов, не превышающую 8 или 10 недель. Если есть этапы, имеющие большую длительность, их следует разбить на этапы меньшей длительности.

При расчете длительности этапов менеджер должен учитывать, что выполнение любого этапа не обойдется без больших или маленьких проблем и задержек. Разработчики могут допускать ошибки или задерживать свою работу, техника может выйти из строя либо аппаратные или программные средства поддержки процесса разработки могут поступить с опозданием. Если проект инновационный и технически сложный, это становится дополнительным фактором появления непредвиденных проблем и увеличения длительности реализации проекта по сравнению с первоначальными оценками.

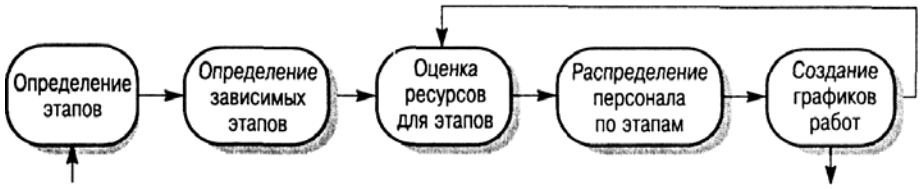


Рис. 2.- Процесс составления графика работ

Кроме временных затрат, менеджер должен рассчитать другие ресурсы, необходимые для успешного выполнения каждого этапа. Особый вид ресурсов — это команда разработчиков, привлеченная к выполнению проекта. Другими видами ресурсов могут быть необходимое свободное дисковое пространство на сервере, время использования какого-либо специального оборудования и бюджетные средства на командировочные расходы персонала, работающего над проектом.

Существует хорошее эмпирическое правило: оценивать временные затраты так, как будто ничего непредвиденного и "плохого" не может случиться, затем увеличить эти оценки для учета возможных проблем. Возможные, но трудно прогнозируемые проблемы существенно зависят от типа и параметров проекта, а также от квалификации и опыта членов команды разработчиков. К исходным расчетным оценкам рекомендуется добавлять 30% на возможные проблемы и затем еще 20%, чтобы быть готовым к тому, что невозможно предвидеть.

График работ по проекту обычно представляется в виде набора диаграмм и графиков, показывающих разбиение проектных работ на этапы, зависимости между этапами и распределение разработчиков по этапам.

Временны*е* и сетевые диаграммы

Временные и сетевые диаграммы полезны для представления графика работ. Временная диаграмма показывает время начала и окончания каждого этапа и его длительность. Сетевая диаграмма отображает зависимости между различными этапами проекта. Эти диаграммы можно создать автоматически с помощью программных средств поддержки управления на основе информации, заложенной в базе данных проекта.

Рассмотрим этапы некоего проекта, представленные в табл. 2, из которой, в частности, видно, что этап Т3 зависит от этапа Т1. Это значит, что этап Т1 должен завершиться прежде, чем начнется этап Т3. Например, на этапе Т1 проводится компонентный анализ создаваемого программного продукта, а на этапе Т3 — проектирование системы.

На основе приведенных значений длительности этапов и зависимости между ними строится сетевой график последовательности этапов (рис. 3). На этом графике видно, какие работы могут выполняться параллельно, а какие должны выполняться последовательно друг за другом. Этапы обозначены прямоугольниками. Контрольные отметки и контрольные проектные элементы показаны в виде овалов и обозначены (как и в табл. 2) буквой М с соответствующим номером. Даты на данной диаграмме соответствуют началу выполнения этапов. Сетевую диаграмму следует читать слева направо и сверху вниз.

Таблица 2. Этапы проекта

Этап	Длительность (дни)	Зависимость
Т1	8	
Т2	15	
Т3	15	Т1 (М1)
Т4	10	
Т5	10	Т2, Т4 (М2)

T6	5	T1, T2 (M3)
T7	20	T1 (M1)
T8	25	T4 (M5)
T9	15	T3, T6 (M4)
T10	15	T5, T7 (M7)
T11	7	T9 (M6)
T12	10	T11 (M8)

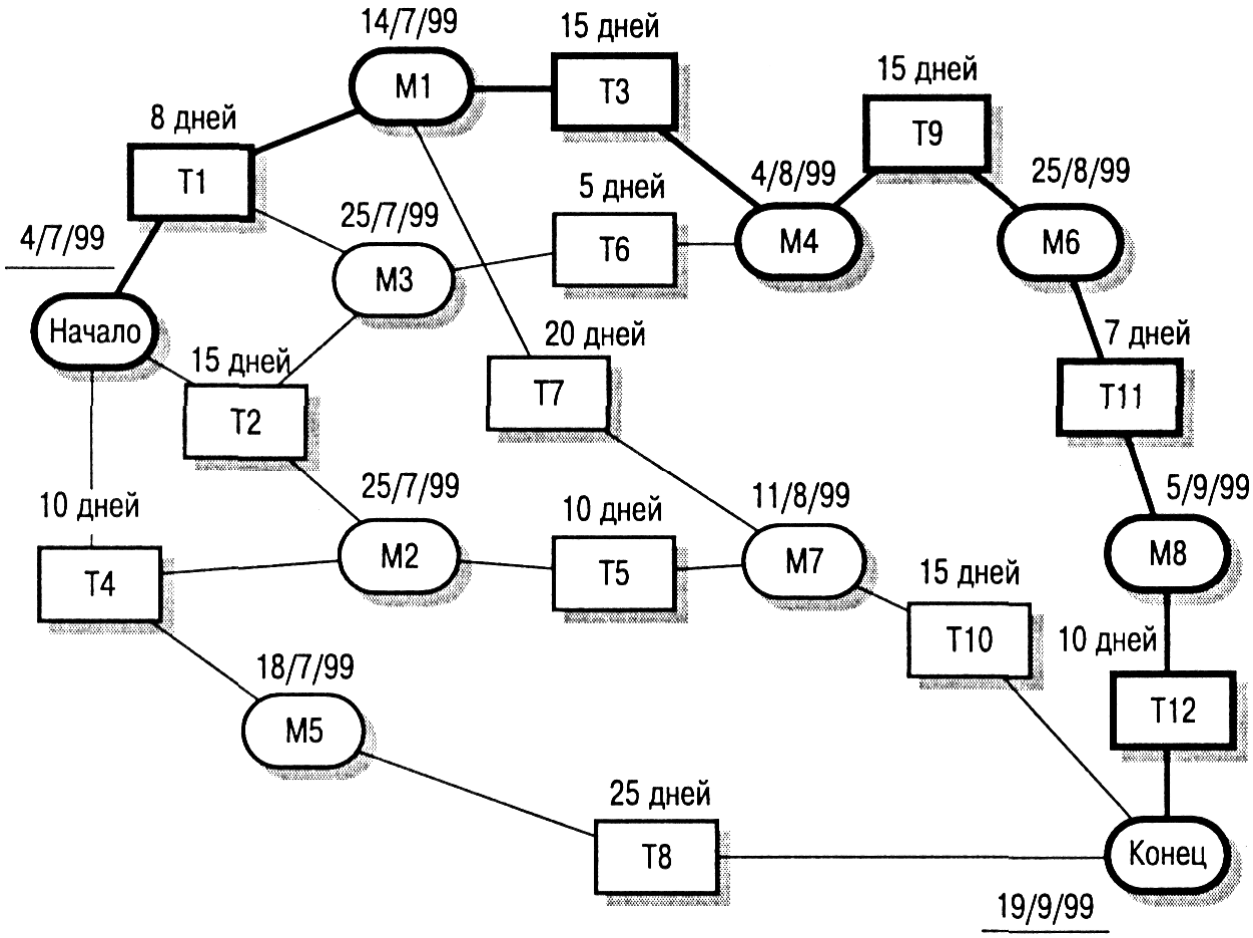


Рис. 3. Сетевая диаграмма этапов

Если для создания сетевой диаграммы используются программные средства поддержки управления проектом, каждый этап должен заканчиваться контрольной отметкой. Очередной этап может начаться только тогда, когда будет получена контрольная отметка (которая может зависеть от нескольких предшествующих этапов). Поэтому в третьем столбце табл. 2 приведены контрольные отметки; они будут достигнуты только тогда, когда будет завершен этап, в строке которого помещена соответствующая контрольная отметка.

Любой этап не может начаться, пока не выполнены все этапы на всех путях, ведущих от начала проекта к данному этапу. Например, этап Т9 не может начаться, пока не будут завершены этапы Т3 и Т6. Отметим, что в данном случае достижение контрольной отметки М4 говорит о том, что эти этапы завершены.

Минимальное время выполнения всего проекта можно рассчитать, просуммировав в сетевой диаграмме длительности этапов на самом длинном пути (длина пути здесь измеряется не количеством этапов на пути, а суммарной длительностью этих этапов) от начала проекта до его окончания (это так называемый критический путь). В нашем случае продолжительность проекта составляет 11 недель или 55 рабочих дней. На рис. 3 критический путь показан более толстыми линиями, чем остальные пути. Таким образом, общая продолжительность реализации проекта зависит от этапов работ, находящихся на критическом пути. Любая задержка в завершении любого этапа на критическом пути приведет к задержке всего проекта.

Задержка в завершении этапов, не входящих в критический путь, не влияет на продолжительность всего проекта до тех пор, пока суммарная длительность этих этапов (с учтом задержек) на каком-нибудь пути не превысит продолжительности работ на критическом пути. Например, задержка этапа Т8 на срок, меньший 20 дней, никак не влияет на общую продолжительность проекта. На рис. 4 представлена временная диаграмма, на которой показаны возможные задержки на каждом этапе.

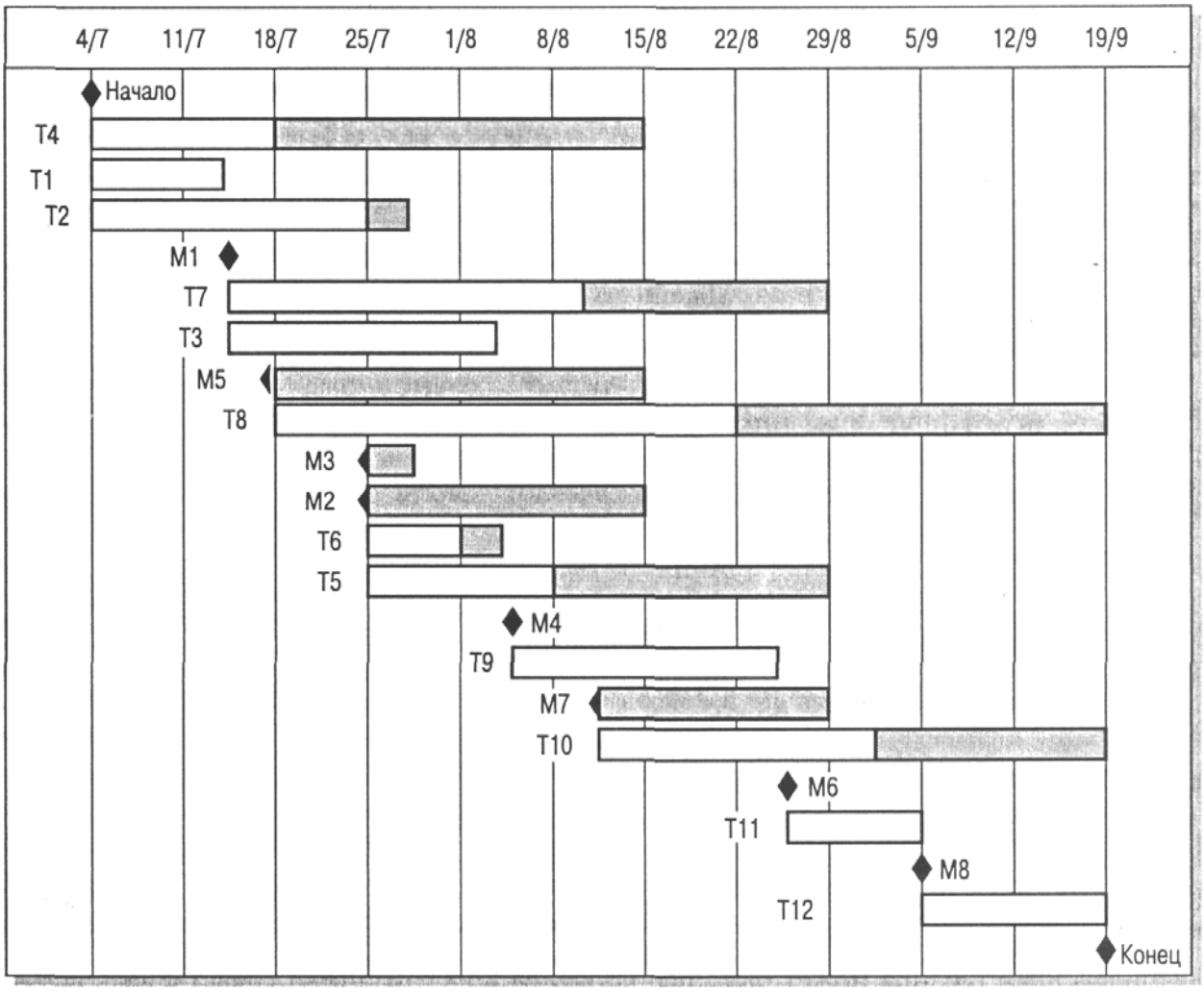


Рис. 4. Временная диаграмма длительности этапов

Сетевая диаграмма позволяет увидеть в зависимости этапов значимость того или иного этапа для реализации всего проекта. Внимание к этапам критического пути часто позволяет найти способы их изменения с тем, чтобы сократить длительность всего проекта. Менеджеры используют сетевую диаграмму для распределения работ.

На рис. 4 показано другое представление графика работ. Это временная диаграмма (иногда называемая по имени ее изобретателя диаграммой Гантта) может быть построена программными средствами поддержки процесса управления. Она показывает длительность выполнения каждого этапа и возможные их задержки (показаны затененными прямоугольниками), а также даты начала и окончания каждого этапа. Этапы критического пути не имеют затененных прямоугольников; это означает, что задержка с завершением данных этапов приведет к увеличению длительности всего проекта.

Подобно распределению времени выполнения этапов, менеджер должен рассчитать распределение ресурсов по этапам, в частности назначить исполнителей на каждый этап. В табл. 3 приведено распределение разработчиков на каждый этап, представленный на рис. 4.

Таблица 3. Распределение исполнителей по этапам

Этап	Исполнитель
T1	Джейн
T2	Анна
T3	Джейн
T4	Фред
T5	Мэри
T6	Анна
T7	Джим
T8	Фред
T9	Джейн
T10	Анна
T11	Фред
T12	Фред

Приведенная таблица может быть использована программными средствами поддержки процесса управления для построения временной диаграммы занятости сотрудников на определенных этапах работ (рис. 5). Персонал не занят в работе над проектом все время его реализации. В течение периода незанятости сотрудники могут быть в отпуске, работать над другими проектами, проходить обучение и т.д.

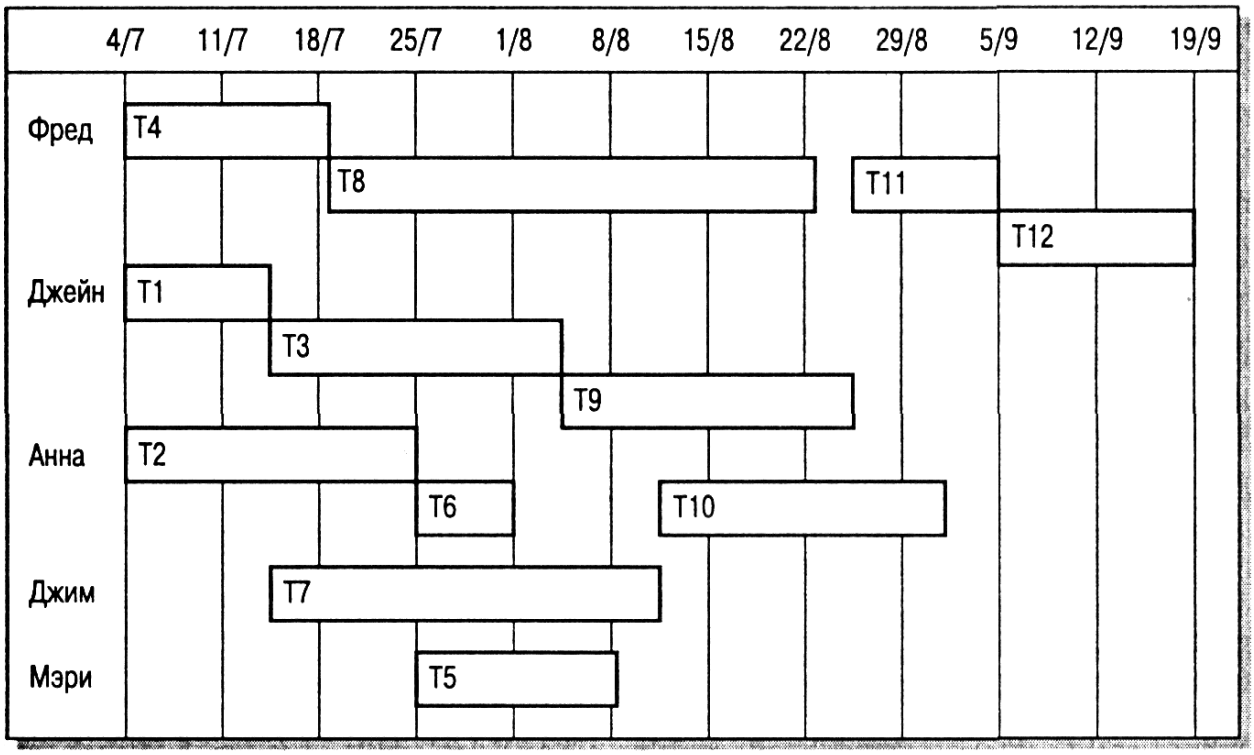


Рис. 5. Временная диаграмма распределения работников по этапам

В больших организациях обычно работает много специалистов, которые задействуются в проекте по мере необходимости. Конечно, такой подход может создать определенные проблемы для менеджеров проектов. Например, если специалист занят в проекте, который задерживается, это может создать прямые сложности для других проектов, где он также должен участвовать.

Первоначальный график работ неизбежно содержит какие-нибудь ошибки или недоработки. По мере реализации проекта рассчитанные оценки длительности выполнения этапов работ должны сравниваться с реальными сроками выполнения этих этапов. Результаты сравнения должны использоваться в качестве основы для пересмотра графика работ еще не реализованных этапов проекта, в частности для того, чтобы попытаться уменьшить длительность этапов критического пути.

Управление рисками

Важной частью работы менеджера проекта является оценка рисков, которые могут повлиять на график работ или на качество создаваемого программного продукта, и разработка мероприятий по предотвращению рисков. Результаты анализа рисков должны быть отражены в плане проекта. Определение рисков и разработка мероприятий по уменьшению их влияния на ход выполнения проекта называется управлением рисками.

Упрощенно риск можно понимать как вероятность проявления каких-либо неблагоприятных обстоятельств, негативно влияющих на реализацию проекта. Риски могут угрожать проекту в целом, создаваемому программному продукту или организации-разработчику. Можно выделить три типа рисков.

1. *Риски для проекта,* которые влияют на график работ или ресурсы, необходимые для выполнения проекта.
2. *Риски для разрабатываемого продукта,* влияющие на качество или производительность разрабатываемого программного продукта.
3. *Бизнес-риски,* относящиеся к организации-разработчику или поставщикам.

Конечно, эти типы рисков могут пересекаться. Например, если опытный программист покидает проект, это будет риском для проекта (поскольку задерживается срок сдачи готового продукта), риском для продукта (так как новый программист, заменивший ушедшего, может оказаться не слишком опытным и сделать ошибки в программе) и бизнес-риском (поскольку задержка данного проекта может негативно повлиять на будущие деловые контакты между заказчиком и организацией-разработчиком).

Конкретные типы рисков, которые могут оказать влияние на данный проект, зависят от вида создаваемого программного продукта и от организационного окружения, где реализуется программный проект. Вместе с тем многие типы рисков способны повлиять на любые программные проекты, эти риски приведены в табл. 4.

Таблица 4. Возможные риски программных проектов

Риск	Типы риска	Описание риска
Текучесть разработчиков	Риск для проекта	Опытные разработчики покидают проект до его завершения
Изменение в управлении организацией	Риск для проекта	Организация меняет свои приоритеты в управлении проектом
Неготовность аппаратных средств	Риск для проекта	Аппаратные средства, которые необходимы для проекта, не поступили вовремя или не готовы к эксплуатации
Изменение требований	Риск для проекта и для разрабатываемого продукта	Появление большого количества непредвиденных изменений в требованиях, предъявляемых к разрабатываемому ПО
Задержка в разработке спецификации	Риск для проекта и для разрабатываемого продукта	Спецификации основных интерфейсов подсистем не поступили к разработчикам в соответствии с графиком работ
Недооценка размера разрабатываемой системы	Риск для проекта и для разрабатываемого продукта	Размер системы значительно превысил первоначальную оценку
Недостаточная эффективность CASE-средств	Риск для разрабатываемого продукта	CASE-средства, предназначенные для поддержки проекта, оказались менее эффективными, чем ожидалось
Изменения в технологии разработки ПО	Бизнес-риск	Основные технологии построения программной системы заменяются новыми
Появление конкурирующего программного продукта	Бизнес-риск	На рынке программных продуктов до окончания проекта появилась конкурирующая программная система

Схема процесса управления рисками показана на рис. 6. Этот процесс состоит из четырех стадий.

1. *Определение рисков.* Определяются возможные риски для проекта, для разрабатываемого продукта и бизнес-риски.
2. *Анализ рисков.* Оценивается вероятность и последовательность появления рисковых ситуаций.
3. *Планирование рисков.* Планируются мероприятия по предотвращению рисков или минимизации их воздействия на проект.
4. *Мониторинг рисков.* Постоянное оценивание вероятностей рисков и выполнение мероприятий по смягчению последствий проявления рисковых ситуаций.

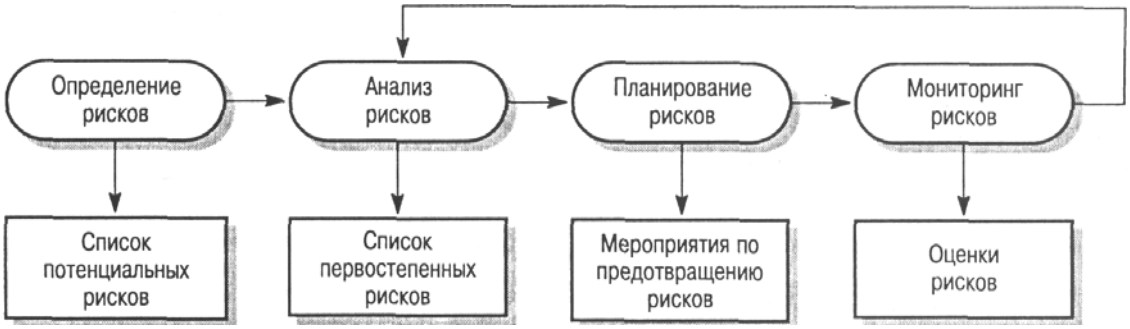


Рис. 6. Процесс управления рисками

Процесс управления рисками, как и другие процессы планирования, является итерационным, выполняемым в течение всего срока реализации проекта. Сначала разрабатываются планы управления рисками, затем постоянно отслеживается ситуация вокруг реализации проекта. При поступлении новой информации о возможных рисках заново проводится анализ рисков и первостепенное внимание уделяется новым рискам. По мере поступления новой информации также изменяются планы мероприятий по предотвращению и смягчению рисков.

Результаты процесса управления рисками документируются в виде планов управления рисками. Они должны включать описание возможных проектных рисков, их анализ и перечень мероприятий, необходимых для управления рисками.

Определение рисков

Определение рисков — первая стадия процесса управления рисками. На этой стадии описываются риски, которые могут проявиться при реализации проекта. В принципе на этой стадии не должна оцениваться вероятность и значимость рисков, но на практике маловероятные риски с незначительными последствиями обычно отбрасываются сразу.

Определение рисков может выполняться в режиме командной работы с использованием подхода "мозговой штурм" либо основываться на опыте менеджера. При определении рисков может помочь приведенный ниже список возможных категорий рисков.

1. *Технологические риски.* Проистекают из программных и аппаратных технологий, на основе которых разрабатывается система.
2. *Риски, связанные с персоналом.* Связаны с членами команды разработчиков.
3. *Организационные риски.* Проистекают из организационного окружения, в котором выполняется проект.
4. *Инструментальные риски.* Связаны с используемыми CASE-средствами и другими средствами поддержки процесса создания ПО.
5. *Риски, связанные с системными требованиями.* Проявляются при изменении требований, предъявляемых к разрабатываемой системе.
6. *Риски оценивания.* Связаны с оцениванием характеристик программной системы и

ресурсов, необходимых для реализации проекта.

В табл. 5 представлены некоторые примеры, относящиеся к каждой из описанных категорий рисков. Результатом этапа определения рисков будет длинный список возможных рисков, которые могут повлиять на разрабатываемый программный продукт, проект или организацию-разработчика.

Таблица 5. Категории рисков

Категория рисков	Примеры рисков
Технологические риски	База данных, которая используется в программной системе, не обеспечивает обработку ожидаемого объема транзакций.
	Программные компоненты, которые используются повторно, имеют дефекты, ограничивающие их функциональные возможности
Риски, связанные с персоналом	Невозможно подобрать работников с требуемым профессиональным уровнем.
	Ведущий разработчик заболел в самое критическое время.
	Невозможно организовать необходимое обучение персонала
Организационные риски	В организации, выполняющей разработку ПО, произошла реорганизация, в результате чего изменились приоритеты в управлении проектом.
	Финансовые затруднения в организации привели к уменьшению бюджета проекта
Инструментальные риски	Программный код, генерируемый CASE-средствами, не эффективен.
	CASE-средства невозможно интегрировать с другими средствами поддержки проекта
Риски, связанные с системными требованиями	Изменения требований приводят к значительным повторным темными требованиями работам по проектированию системы.
	Первоначальная нечеткая формулировка пользовательских требований привела к значительным изменениям системных требований, проявившихся на поздних стадиях разработки проекта
Риски оценивания	Недооценки времени выполнения проекта.

	Скорость выявления дефектов в системе ниже ранее запланированной.
	Размер системы значительно превышает первоначально рассчитанный

Анализ рисков

При анализе для каждого определенного риска подсчитывается вероятность его проявления и ущерб, который он может нанести. Не существует простых методов выполнения анализа рисков — в значительной мере он основан на мнении и опыте менеджера. Можно привести следующую шкалу вероятностей рисков и их последствий.

1. Вероятность риска считается очень низкой, если она имеет значение менее 10%; низкой, если ее значение от 10 до 25 %; средней при значениях от 25 до 50%; высокой, если значение колеблется от 50 до 75%; очень высокой при значениях более 75%.
2. Возможный ущерб от рисковых ситуаций можно подразделить на катастрофический, серьезный, терпимый и незначительный.

Результаты анализа рисков должны быть представлены в виде таблицы рисков, упорядоченных по степени возможного ущерба. В табл. 6 приведен упорядоченный список рисков, описанных в табл. 5; там же указаны вероятности этих рисков. Здесь вероятности рисков и степень ущерба от них указаны произвольно. На практике для их определения необходима подробная информация о проекте, технологии создания ПО, команде разработчиков и о самой организации.

Таблица 6. Список рисков после проведения их анализа

Риск	Вероятность	Степень ущерба
Финансовые затруднения в организации привели к уменьшению бюджета проекта	Низкая	Катастрофическая
Невозможно подобрать работников с требующимся профессиональным уровнем	Высокая	Катастрофическая
Ведущий разработчик заболел в самое критическое время	Средняя	Серьезная
Программные компоненты, используемые повторно, имеют дефекты, ограничивающие их функциональные возможности	Средняя	Серьезная
Изменения требований приводят к значительным повторным работам по проектированию системы	Средняя	Серьезная
В организации, выполняющей разработку ПО, произошла реорганизация, в результате чего изменились приоритеты в управлении проектом	Высокая	Серьезная
База данных, которая используется в программной системе, не обеспечивает обработку ожидаемого объема транзакций	Средняя	Серьезная

Недооценки времени выполнения проекта	Высокая	Серьезная
CASE-средства невозможно интегрировать с другими средствами поддержки проекта	Высокая	Терпимая
Первоначальная нечеткая формулировка пользовательских требований привела к значительным изменениям системных требований, проявившихся на поздних стадиях разработки проекта	Средняя	Терпимая
Невозможно организовать необходимое обучение персонала	Средняя	Терпимая
Скорость выявления дефектов в системе ниже ранее спланированной	Средняя	Терпимая
Размер системы значительно превышает первоначально рассчитанный	Высокая	Терпимая
Программный код, генерируемый CASE-средствами, неэффективен	Средняя	Незначительная

Конечно, как вероятность рисков, так и возможный ущерб от них должны пересматриваться при поступлении дополнительной информации об этих рисках и по мере реализации мероприятий по управлению ими. Поэтому подобные таблицы рисков должны переделываться на каждой итерации процесса управления рисками.

После проведения анализа рисков определяются наиболее значимые риски, которые затем отслеживаются на протяжении всего срока выполнения проекта. Определение этих значимых рисков зависит от их вероятностей и возможного ущерба. В общем случае всегда отслеживаются риски с катастрофическими последствиями, а также риски с серьезным ущербом, значение вероятности которых выше среднего.

В некоторых статьях рекомендуется определить и отслеживать "10 верхних" рисков, но это не всегда обоснованная рекомендация. Количество рисков, которые необходимо отслеживать, зависит от конкретного проекта. Это может быть пять рисков, а может — пятнадцать. Но, конечно, количество рисков, по которым проводится мониторинг, должно быть обозримым. Большое количество отслеживаемых рисков потребует огромного количества собираемой информации. Из списка рисков, представленных в табл. 6, для мониторинга следует отобрать те риски, которые могут привести к катастрофическим и серьезным последствиям для вашего проекта.

Планирование рисков

Планирование заключается в определении стратегии управления каждым значимым риском, отобранным для мониторинга после анализа рисков. Здесь также не существует общепринятых подходов для разработки таких стратегий — многое основывается на "чутье" и опыте менеджера проекта. В табл. 7 показаны возможные стратегии управления основными рисками, приведенными в табл. 6.

Таблица 7. Стратегии управления рисками

Риск	Стратегия
Финансовые	Подготовить краткий документ для руководства

проблемы организации	организации, показывающий важность данного проекта для достижения финансовых целей организации
Проблемы неквалифицированного персонала	Предупредить заказчика о потенциальных трудностях и возможной задержке проекта, рассмотреть вопрос о покупке компонентов системы
Болезни персонала	Реорганизовать работу команды разработчиков таким образом, чтобы обязанности и работа членов команды перекрывали друг друга, вследствие этого разработчики будут знать и понимать задачи, выполняемые другими сотрудниками
Дефектные системные компоненты	Заменить потенциально дефектные системные компоненты покупными компонентами, гарантирующими качество работы
Изменения требований	Попытаться определить требования, наиболее вероятно подверженные изменениям; в структуре системы не отображать детальную информацию
Реорганизация компании-разработчика	Подготовить краткий документ для руководства компании, показывающий важность данного проекта для достижения финансовых целей компании
Недостаточная производительность базы данных	Рассмотреть возможность покупки более производительной базы данных
Недооценки времени выполнения проекта	Рассмотреть вопрос о покупке системных компонентов, исследовать возможность использования генератора программного кода

Существует три категории стратегий управления рисками.

1. *Стратегии предотвращения рисков.* Согласно этим стратегиям следует проводить мероприятия, снижающие вероятность проявления рисков. Примером может служить стратегия исключения потенциально дефектных компонентов, описанная в табл. 7.
2. *Минимизационные стратегии.* Направлены на уменьшение возможного ущерба от рисков. Примером служит стратегия уменьшения ущерба от болезни членов команды разработчиков (см. табл. 7).
3. *Планирование "аварийных" ситуаций.* Согласно этим стратегиям необходимо иметь план мероприятий, которые следует выполнить в случае проявления рисковой ситуации. В табл. 7 это стратегия поведения при возникновении финансовых проблем у организации-разработчика.

Мониторинг рисков

Мониторинг рисков заключается в регулярном пересчете вероятностей рисков и ущерба, который они могут нанести. Для этого необходимо постоянно отслеживать факторы, которые влияют на вероятность рисков и возможный ущерб. Эти факторы зависят от типов риска. В табл. 8 приведены признаки, которые помогают определить тип риска.

Таблица 8. Признаки рисков

Тип риска	**Признаки**

Технологические риски	Задержки в поставке оборудования или программных средств поддержки процесса создания ПО, многочисленные документированные технологические проблемы
Риски, связанные с персоналом	Низкое моральное состояние персонала, натянутые отношения между членами команды разработчиков, низкое качество выполненной работы
Организационные риски	Разговоры среди персонала о пассивности и недостаточной компетентности высшего руководства организации
Инструментальные риски	Нежелание разработчиков использовать программные средства поддержки, неодобрительные отзывы о CASE-средствах, запросы на более мощные инструментальные средства
Риски, связанные с системными требованиями	Необходимость пересмотра многих системных требований, недовольство заказчика ПО
Риски оценивания	Изменения графика работ, многочисленные отчеты о нарушении графика работ

Мониторинг рисков должен быть непрерывным процессом, отслеживающим ход выполнения мероприятий по управлению рисками, при этом каждый основной риск должен рассматриваться отдельно.

4. Порядок выполнения работы

1. Изучить предлагаемый теоретический материал.
2. Построить временную и сетевую диаграммы для выбранного проекта.
3. Построить диаграмму распределения участников группы по этапам.
4. Построить список возможных рисков с указанием названия риска, его описание и типа.
5. Провести анализ рисков.
6. Описать стратегию планирования рисков.
7. Построить отчёт, включающий все полученные диаграммы и описание стратегии планирования рисков.

5. Содержание отчета

В отчете следует указать:

- Цель работы
- Введение
- Программно-аппаратные средства, используемые при выполнении работы.
- Основную часть (описание самой работы), выполненную согласно требованиям к результатам выполнения лабораторного практикума (п.2).
- Заключение (выводы)
- Список используемой литературы

6. Литература:

1. Буч Г., Рамбо Дж., Джекобсон А. Язык UML. Руководство пользователя. – С-П.: Издательство «Питер», 2003. – 432 с.
2. Соммервиль Иан. Инженерия программного обеспечения, 6-е издание. : Пер. с англ. – М.: Издательский дом “Вильямс”, 2002. – 624 с.
3. Константайн Л., Локвуд Л. Разработка программного обеспечения. – СПб.:Питер, 2004. – 592 с.