Документ подписан простой электронной подписью
Информация о владельце:
ФИО: Шебзухова Татьяна Александровна
Должность: Директор Пятигорского института (филиал) Северо-Кавказского федерального университета
Дата подписания: 21.05.2025 11:46:46
Уникальный программный ключ:
d74ce93cd40e39275c3ba2f58486412a1c8ef96f

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ ПО ДИСЦИПЛИНЕ

ТЕХНОЛОГИИ СОЗДАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2025

СОДЕРЖАНИЕ

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью освоения дисциплины «Технология создания информационных систем» является формирование набора профессиональных компетенций будущего магистра по направлению подготовки 09.04.02 «Информационные системы и технологии», для решения прикладных задач в рамках направленности (профиля) «Технологии работы с данными и знаниями, анализ информации».

Задачи освоения дисциплины: изучение знаний, освоение и умение применять на практике методы и способы создания информационных систем и технологий, умение профессионально использовать современное оборудование, освоение современных инструментальных средств для обработки данных и анализа информации.

2. НАИМЕНОВАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1. Разработка описания и анализ информационной системы
Лабораторная работа 2. Разработка технического задания для создания информационной системы
Лабораторная работа 3. Разработка требований к информационной системе
Лабораторная работа 4. Методология функционального моделирования.
Лабораторная работа 5. Методология объектно-ориентированного моделирования

3. СОДЕРЖАНИЕ ЛАБОРАТОРНЫХ РАБОТ

Лабораторная работа 1. Разработка описания и анализ информационной системы

Цель работы: описать и проанализировать информационную систему, распределить роли в группе разработчиков.

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:

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

На выполнение лабораторной работы предусмотрено 3 часа.

Теоретические сведения

Общие сведения о разработке информационной системы

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 ИС, 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 создании ИС.

Имеется существенная разница между профессиональной разработкой ИС 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 разрабатываемой ИС.

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 готовой ИС, в превышении сметной стоимости проекта и в несоответствии готового ПО спецификации требований.

Процесс создания ИС 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. 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 информационной системы

Невозможно описать и стандартизировать все работы, выполняемые в проекте по созданию ИС. Эти работы весьма существенно зависят от организации, где выполняется разработка ИС, 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.

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.

Выделяются следующие роли в группе по созданию ИС:

- Руководитель – общее руководство проектом, написание документации,

общение с заказчиком ИС

- 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 жизненным циклом информационной системы 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.

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.

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.

Содержание отчета

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, 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

Вопросы для обсуждения:

Общие сведения о разработке информационной системы

Процесс управления разработкой информационной системы

Планирование проекта разработки программного обеспечения

Общие сведения о требованиях к информационным системам

Первые шаги по разработке требований к информационным системам анализ осуществимости

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Оценочные средства: отчет к лабораторной работе (См.: Фонд оценочных средств)

Лабораторная работа 2. Разработка технического задания для создания информационной системы

Цель работы: составить и оформить техническое задание на разработку программного обеспечения.

Лабораторная работа направлена на ознакомление с процессом разработки технического задания на разработку программного обеспечения.

Порядок выполнения работы

4. Изучить предлагаемый теоретический материал.
5. Составить подробное описание информационной системы.
6. 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.

7. Распределить роли в группе (руководитель проекта-разработчик, системный аналитик-разработчик, тестер-разработчик).
8. 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.

9. Составить отчет о проделанной работе.

Содержание отчета

9. Цель работы

10. Введение. Краткое описание целей проекта и проектных ограничений (бюджетных, временных и т.д.), которые важны для управления проектом

11. 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

12. Анализ осуществимости, указать возможные проблемы и пути их решения.
13. Роли участников группы разработки ПО.
14. Программно-аппаратные средства, используемые при выполнении работы.
15. Заключение (выводы)
16. Список используемой литературы

Вопросы для обсуждения:

Общие сведения о разработке информационной системы

Процесс управления разработкой информационной системы

Планирование проекта разработки программного обеспечения

Общие сведения о требованиях к информационным системам

Первые шаги по разработке требований к информационным системам анализ осуществимости

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Оценочные средства: отчет к лабораторной работе (См.: Фонд оценочных средств)

Лабораторная работа 3. Разработка требований к информационной системе

Цель работы: составить и проанализировать требования к информационной системе, оформить техническое задание на разработку программного обеспечения.

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 лабораторной работы:

- наличие диаграммы идентификации точек зрения и диаграммы иерархии точек зрения;
- наличие сценариев событий (последовательности действий);
- наличие пользовательских требований, четко описывающих будущий функционал системы;
- 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.);
- наличие составленного технического задания.

На выполнение лабораторной работы предусмотрено 3 часа.

Теоретические сведения

Общие сведения о требованиях к информационным системам

Проблемы, которые приходится решать специалистам в процессе создания информационных систем 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.

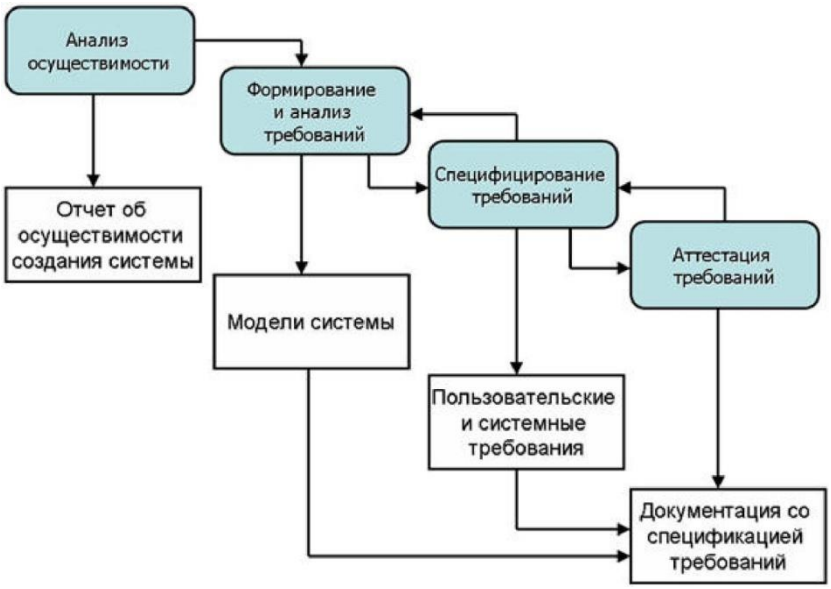

Рис. 1. Процесс разработки требований

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.

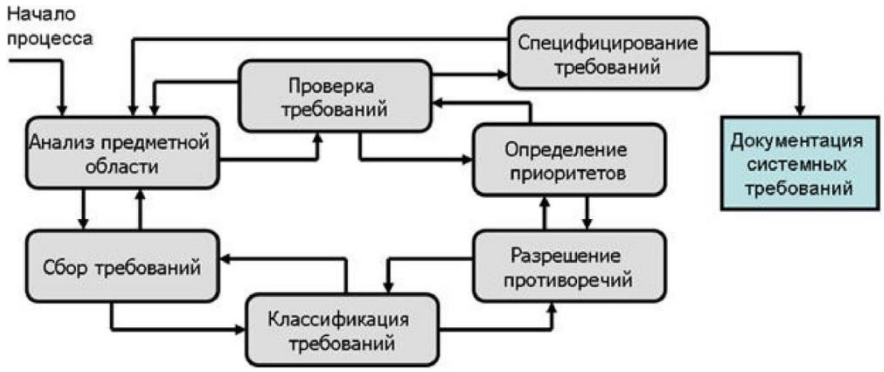

Рис. 2. Процесс формирования и анализа требований

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 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.

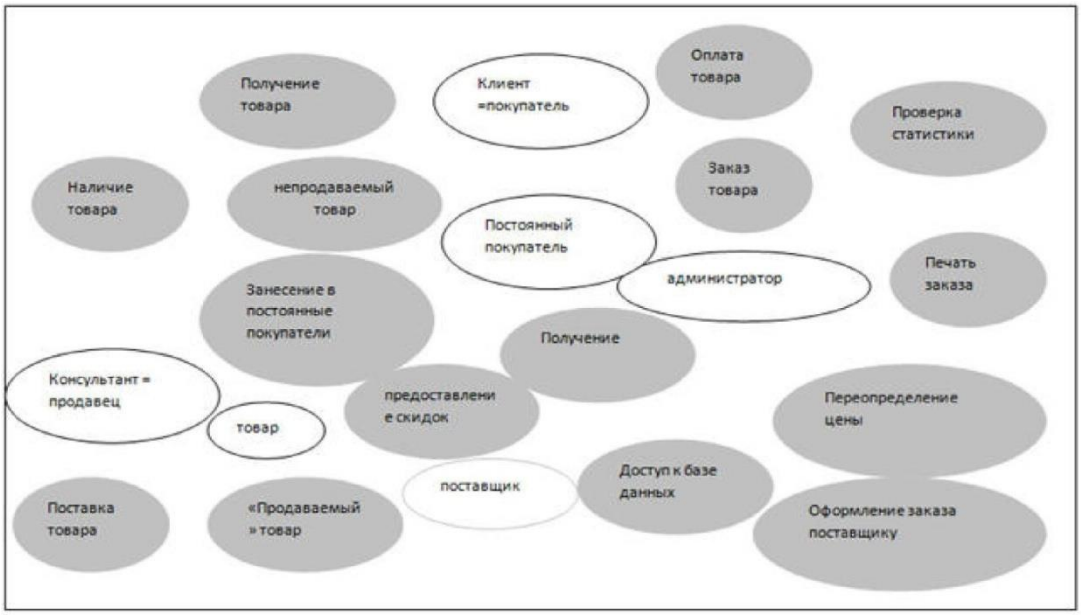

Рис. 4. Диаграмма идентификации точек зрения

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
Получени е чека			Назначение цены		Доступ к каталогу	Переопре деление цены

Заказ товара			Переопределение цены		Проверка наличия товара	Оформление заказа поставщику
Занесение покупателя и суммы покупки в базу данных			«Покупаемый» или «непокупаемый» 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.

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 ИС. ТЗ должно содержать основные разделы, описанные в ГОСТ 34.602-89.
6. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

Содержание отчета

1. Цель работы
2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основная часть (описание самой работы), выполненная согласно требованиям к результатам выполнения лабораторной работы.
5. Заключение (выводы)
6. Список используемой литературы

Вопросы для обсуждения:

Общие сведения о требованиях к информационным системам
Разработка требований. Процесс разработки требований
Формирование и анализ требований
Опорные точки зрения
Аттестация требований
Пользовательские и системные требования

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Оценочные средства: отчет к лабораторной работе(См.: Фонд оценочных средств)

Лабораторная работа 4. Методология функционального моделирования

Цель работы: изучить методологии функционального моделирования IDEF0 и IDEF3.

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.

На выполнение лабораторной работы предусмотрено 3 часа.

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.

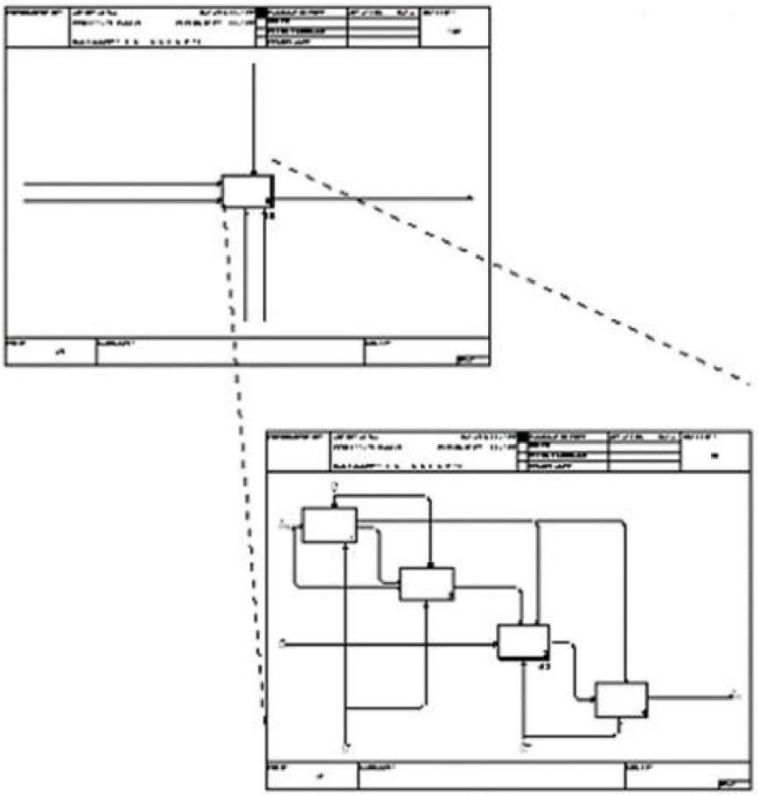

Рис. 2. Декомпозиция функционального блока Пример.

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

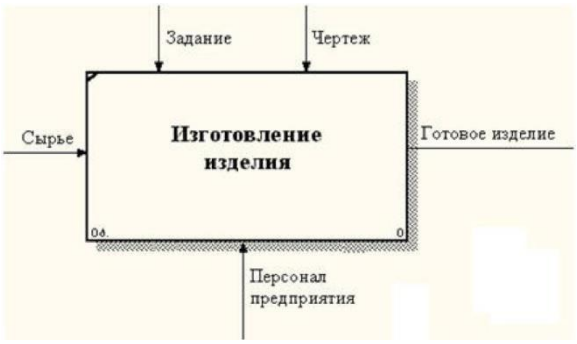

Рис. 3. Контекстная диаграмма

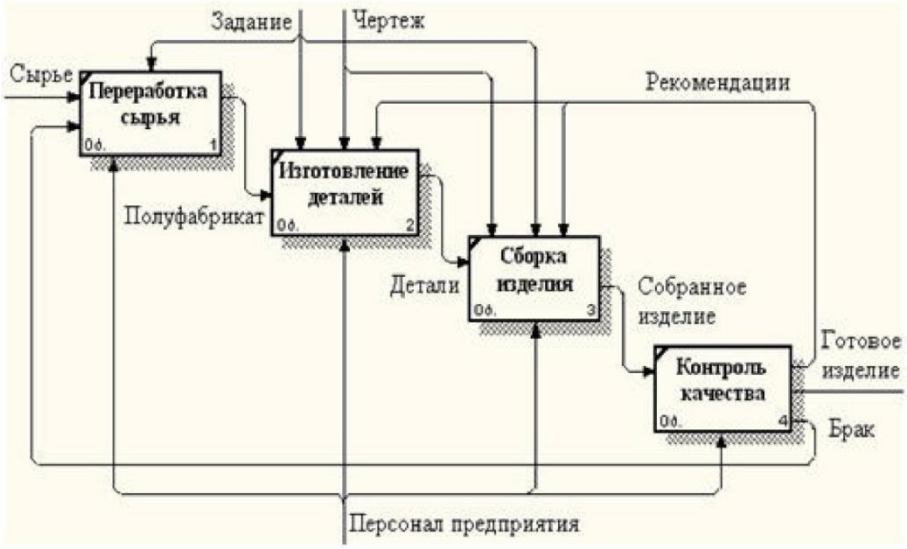

Рис.4. Диаграмма первого уровня декомпозиции

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
O	Asynchronous OR	Один или несколько предшествующих процессов должны быть завершены	Один или несколько следующих процессов должны быть запущены
O	Synchronous OR	Один или несколько предшествующих процессов завершены одновременно	Один или несколько следующих процессов запускаются одновременно
X	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)

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
   - 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.

Содержание отчета

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 задания.
5. Заключение (выводы)
6. Список используемой литературы

Вопросы для обсуждения:

IDEF0. Основные понятия IDEF0
Принципы моделирования в IDEF0
Применение IDEF0
IDEF3. Метод описания процессов IDEF3

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Оценочные средства: отчет к лабораторной работе (См.: Фонд оценочных средств)

Лабораторная работа 5. Методология объектно-ориентированного моделирования

Цель работы: ознакомление с основными элементами определения, представления, проектирования и моделирования программных систем с помощью языка UML.

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.

На выполнение лабораторной работы предусмотрено 4,5 часа.

Теоретические сведения

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.

Язык UML

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

Стандарт UML предлагает следующий набор диаграмм для моделирования:

1. диаграммы вариантов использования (use case diagrams) – для моделирования бизнес-процессов организации и требований к создаваемой системе);
2. диаграммы классов (class diagrams) –

для моделирования статической структуры классов системы и связей между ними;

3. 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;

4. 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 при дал 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.

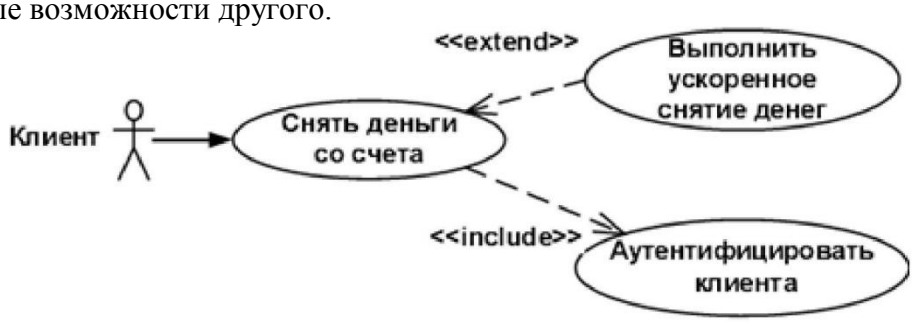

Рис.5. Пример связи включения и расширения

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

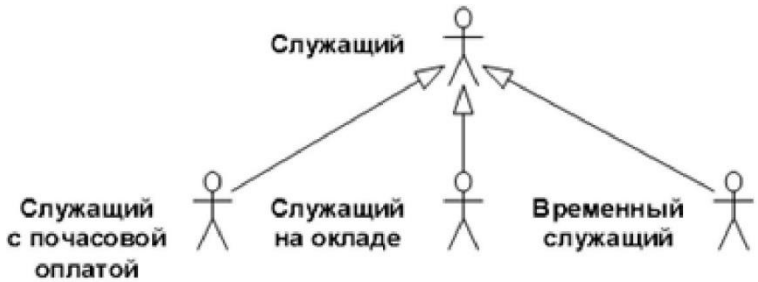

Рис.6. Пример связи обобщения

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 последовательно сти (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.

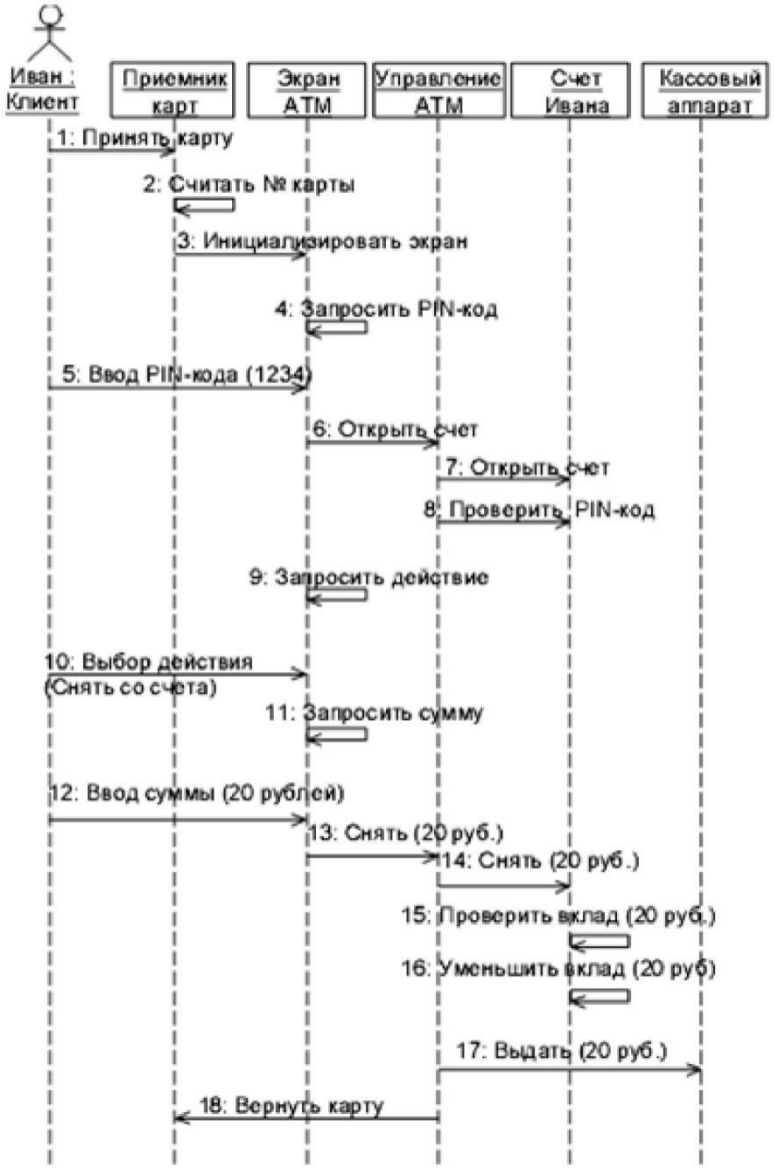

Рис. 7. Пример диаграммы последовательности

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 последовательно сти, стрелки обозначают сообщения, обмен которыми осуществляется в рамках данного варианта использования. Их временная последовательность указывается путем нумерации сообщений.

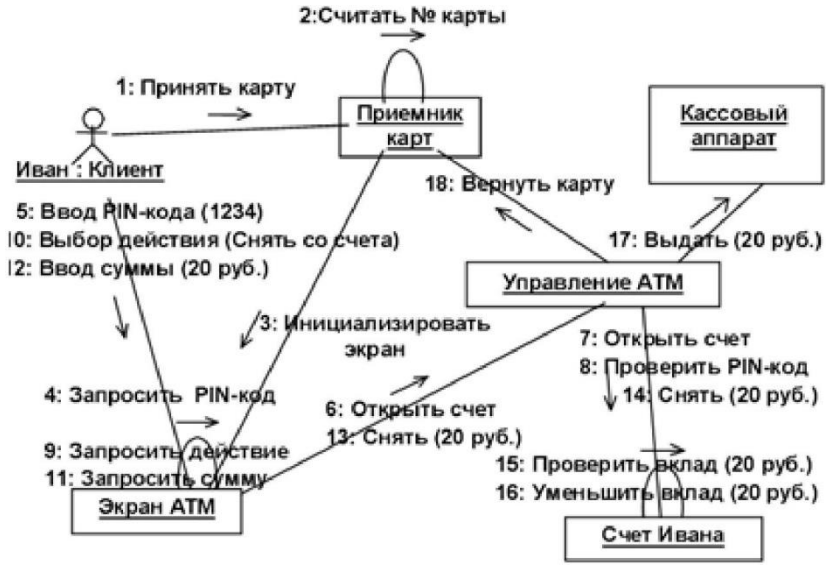

Рис. 8. Пример диаграммы кооперации

Диаграммы классов

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).

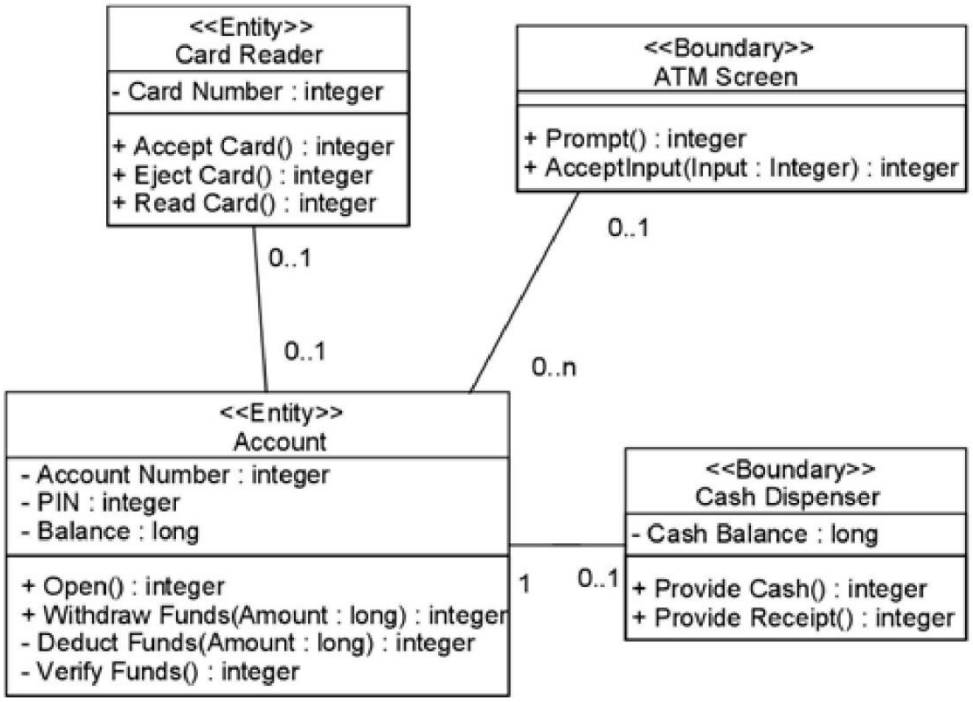

Рис. 9. Пример диаграммы классов

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 А использует методы класса В. Тогда при изменении класса В необходимо произвести соответствующие изменения в классе А.

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.

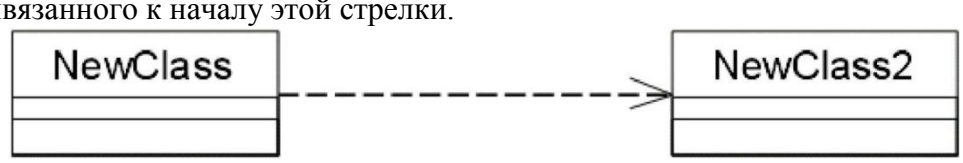

Рис. 11. Связь зависимость

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.

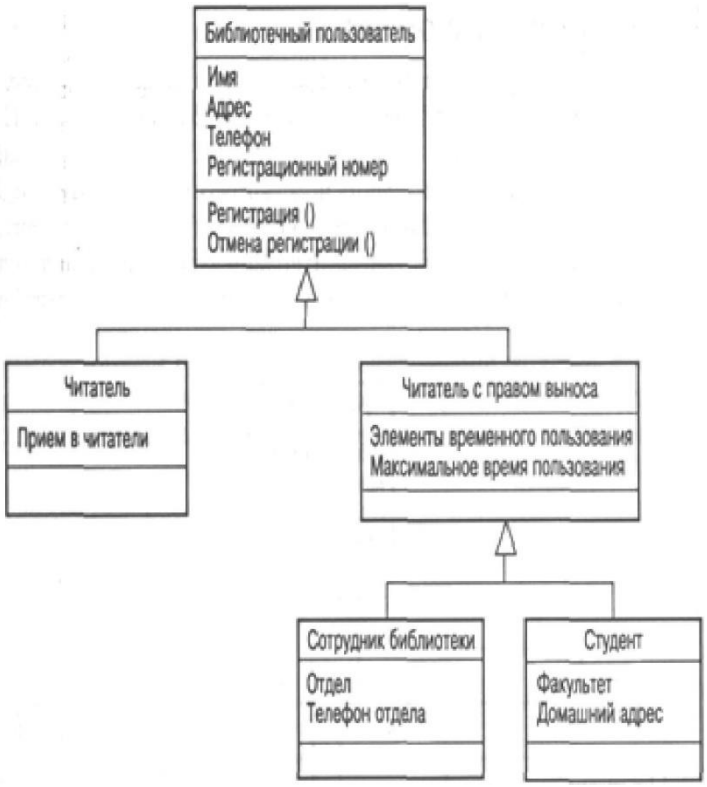

Рис. 12. Пример связи наследование

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):

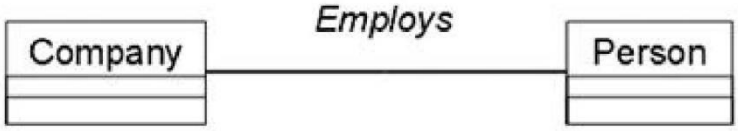

Рис. 13. Пример имен связей

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:

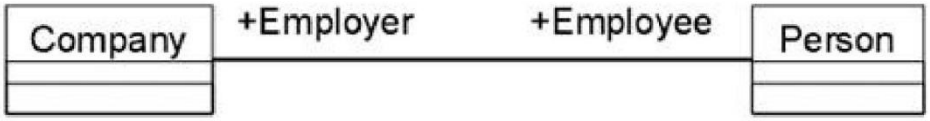

Рис. 14. Пример ролей связей

Пакет. Механизм пакетов

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

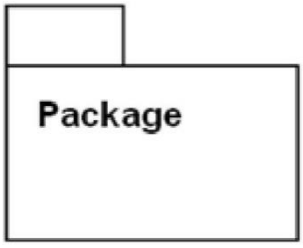

Рис. 15. Обозначение пакета в UML

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.

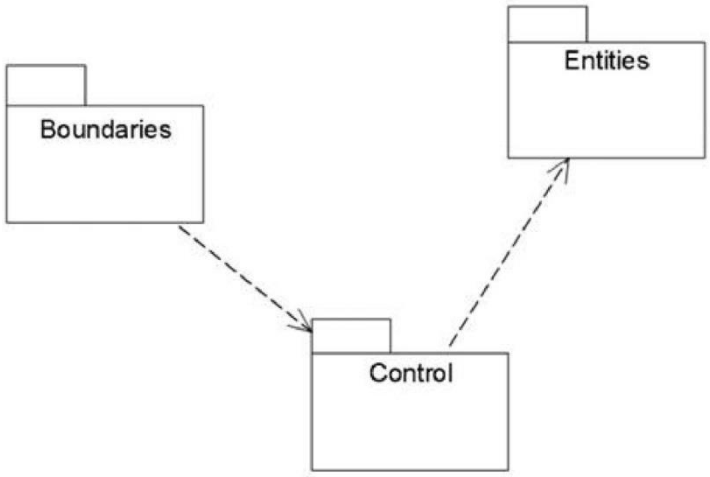

Рис. 16. Пример диаграммы пакетов

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 « ^ ».

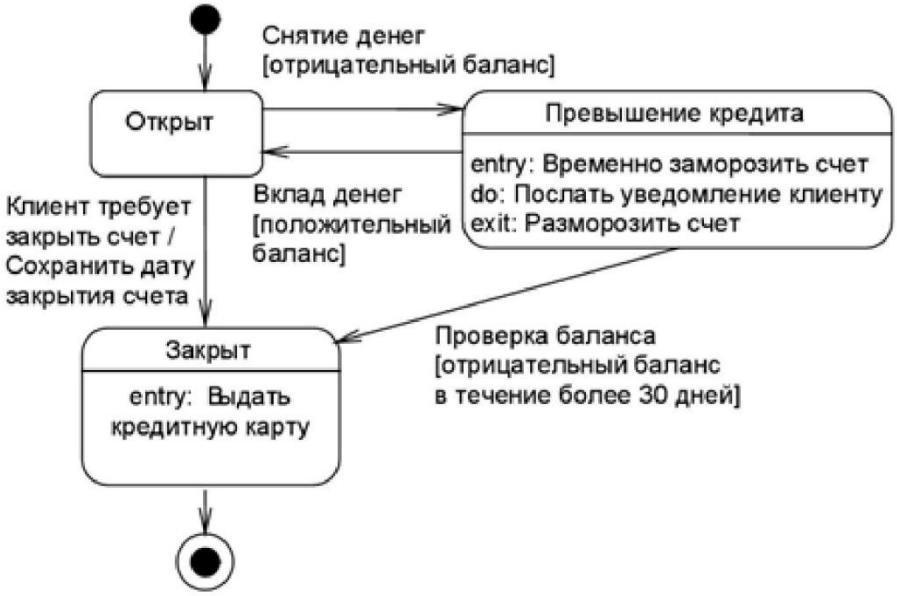

Рис. 17. Пример диаграммы состояний

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.

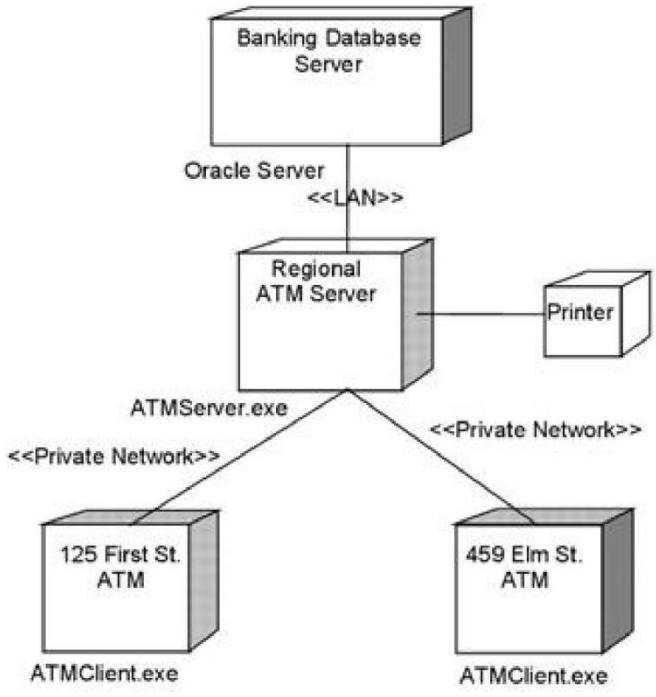

Рис. 19. Пример диаграммы размещения

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.

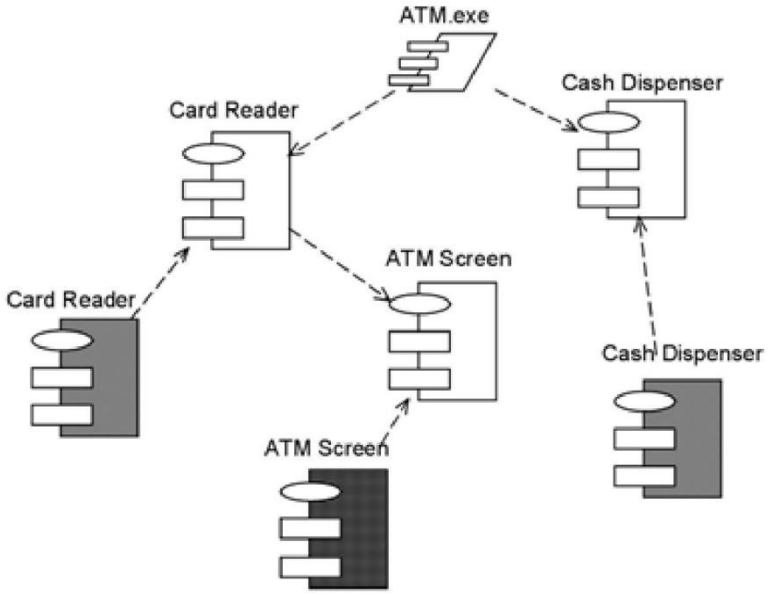

Рис. 18. Пример диаграммы компонентов

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.

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.

Содержание отчета

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 лабораторной работы.
5. Заключение (выводы)
6. Список используемой литературы

Вопросы для обсуждения:

Общие сведения об объектном моделировании ИС

Унифицированный язык объектно-ориентированного моделирования Unified Modeling Language (UML)

Диаграммы вариантов использования. Связи между вариантами использования и действующими лицами.

Диаграммы взаимодействия (interaction diagrams)

Диаграмма последовательности (sequence diagrams)

Диаграмма кооперации (collaboration diagram)

Диаграммы классов

Диаграммы размещения

Диаграммы компонентов

Объединение диаграмм компонентов и развертывания

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Оценочные средства: отчет к лабораторной работе (См.: Фонд оценочных средств)

4. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя;

недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

5. МЕТОДИЧЕСКИЕ МАТЕРИАЛЫ, ОПРЕДЕЛЯЮЩИЕ ПРОЦЕДУРЫ ОЦЕНИВАНИЯ ЗНАНИЙ, УМЕНИЙ, НАВЫКОВ И (ИЛИ) ОПЫТА ДЕЯТЕЛЬНОСТИ, ХАРАКТЕРИЗУЮЩИХ ЭТАПЫ ФОРМИРОВАНИЯ КОМПЕТЕНЦИЙ

Текущая аттестация студентов проводится преподавателями, ведущими лабораторные занятия по дисциплине, в следующих формах: отчет письменный по заданию преподавателя, контрольная работа.

Допуск к лабораторным работам происходит при наличии у студентов печатного варианта отчета. Защита отчета проходит в форме доклада студента по выполненной работе и ответов на вопросы преподавателя.

Отчет включает в себя следующие разделы: титульный лист с названием работы; цель работы; краткие теоретические сведения; описание результатов лабораторной работы (скриншоты); вывод из работы, включающий в себя описание проделанной работы.

Оценку «отлично» студент получает, если оформление отчета соответсвует установленным требованиям, правильно отвечает на предложенные преподавателем контрольные вопросы, правильно отвечает на дополнительные вопросы по теме лабораторной работы.

Оценку «хорошо» студент получает, если оформление отчета соответсвует установленным требованиям, правильно отвечает на предложенные преподавателем контрольные вопросы.

Оценку «удовлетворительно» студент получает без беседы с преподавателем, если оформление отчета соответсвует установленным требованиям.

Отчет может быть отправлен на доработку в следующих случаях:

- польностью не соовествует установленным требованиям;
- не раскрыта суть работы.

6. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

6.1. Рекомендуемая литература

6.1.1. Основная литература

1. Избачков Ю.С., Петров В.Н., Васильев А.А., Телина И. С. Информационные системы: Учебник для ВУЗов. 3-е изд. – СПб.: Питер, 2011
2. Информационные системы в экономике / Под ред. Д. Дика – М. 2010.

6.1.2. Дополнительная литература

1. Благодатских В.А. Стандартизация разработки программных средств: учеб.пособие / В.А. Благодатских, В.А. Волнин, К.Ф. Поскакалов; под ред. Проф. О.С.Разумова, 2010 г. - 288 с.

6.1.3. Методическая литература

1. Методические указания по выполнению лабораторных работ по дисциплине «Технология создания информационных систем».
2. Методические указания по выполнению контрольной работы по дисциплине «Технология создания информационных систем».
3. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Технология создания информационных систем».

6.1.4. Интернет-ресурсы

1. http://www.intuit.ru – сайт дистанционного образования в области информационных технологий
2. http://window.edu.ru – образовательные ресурсы ведущих вузов

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ СТУДЕНТОВ ПО ОРГАНИЗАЦИИ САМОСТОЯТЕЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ ТЕХНОЛОГИИ СОЗДАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Под самостоятельной работой понимается планируемая учебная, учебно-исследовательская, а также научно-исследовательская работа, которая выполняется во внеаудиторное время по инициативе магистранта или по заданию и при методическом руководстве преподавателя, но без его непосредственного участия.

Целью самостоятельной работы является изучение тем, не рассмотренных в течение аудиторных занятий

Задачи самостоятельной работы:

формирование целостного представления о современных информационных технологиях, применяемых при обработке результатов научных исследований, сборе, хранении, обработке и передачи информации, и их роли в развитии общества;

умение использовать инструментарий компьютерных технологий в профессиональной деятельности;

свободное владение базовыми понятиями, концепциями и методами информатизации науки и образования при проведении самостоятельных научных исследований и в обучении;

приобретение навыков использования методов и приемов решения задач науки и образования на базе компьютерных технологий;

формирование способности самостоятельно приобретать и применять новые знания и умения;

обеспечение гармоничного развития магистранта и подготовки его к эффективной работе в условиях массового внедрения вычислительной техники во все сферы человеческой деятельности.

Выполнение заданий по самостоятельной работе позволяет закрепить знания и приобрести практические навыки в области технологий создания информационных систем и технологий.

1. ЦЕЛЬ И ЗАДАЧИ ОСВОЕНИЯ ДИСЦИПЛИНЫ

Целью освоения дисциплины «Технология создания информационных систем» является формирование набора профессиональных компетенций будущего магистра по направлению подготовки 09.04.02 «Информационные системы и технологии», для решения прикладных задач в рамках направленности (профиля) «Технологии работы с данными и знаниями, анализ информации».

Задачи освоения дисциплины: изучение знаний, освоение и умение применять на практике методы и способы создания информационных систем и технологий, умение профессионально использовать современное оборудование, освоение современных инструментальных средств для обработки данных и анализа информации.

3. ТЕХНОЛОГИЧЕСКАЯ КАРТА САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТА

Для студентов заочной формы обучения:

Коды реализуемых компетенций, индикатора (ов)	Вид деятельности студентов	Средства и технологии оценки	Объем часов, в том числе		
			СРС	Контактная работа с преподавателем	Всего

2 семестр					
ПК-1 (ИД 1 ПК-1, ИД 2 ПК-1, ИД 3 ПК-1) ПК-6 (ИД 1 ПК-6, ИД 2 ПК-6, ИД 3 ПК-6)	Подготовка к лекциям	Собеседование	0,36	0,04	0,4
ПК-10 (ИД 1 ПК-10, ИД 2 ПК-10, ИД 3 ПК-10) ПК-12 (ИД 1 ПК-12, ИД 2 ПК-12, ИД 3 ПК-12), ПК-13 (ИД 1 ПК-13, ИД 2 ПК-13, ИД 3 ПК-13)	Самостоятельное изучение литературы	Собеседование	102,6	11,4	114
ПК-7 (ИД 1 ПК-7, ИД 2 ПК-7, ИД 3 ПК-7), ПК-8 (ИД 1 ПК-8, ИД 2 ПК-8, ИД 3 ПК-8), ПК-9 (ИД 1 ПК-9, ИД 2 ПК-9, ИД 3 ПК-9),	Подготовка и выполнение лабораторных работ	Отчет письменный	3,24	0,36	3,6
ПК-7 (ИД 1 ПК-7, ИД 2 ПК-7, ИД 3 ПК-7), ПК-8 (ИД 1 ПК-8, ИД 2 ПК-8, ИД 3 ПК-8), ПК-9 (ИД 1 ПК-9, ИД 2 ПК-9, ИД 3 ПК-9),	Выполнение контрольной работы	Контрольная работа	9	1	10
Итого за 2 семестр			115,2	12,8	128

3. СОДЕРЖАНИЕ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Тема самостоятельного изучения № 1. Концепции, методы и принципы создания ИС.

Вид деятельности студентов: подготовка к лекциям, самостоятельное изучение литературы, подготовка и выполнение лабораторных работ, выполнение контрольной работы

Итоговый продукт самостоятельной работы: конспект

Средства и технологии оценки: отчет

Краткие теоретические сведения

Концепции и методы создания ИС – информационных систем

Разработка сложных ИС предприятий, невозможна без тщательно обдуманного методологического подхода. Какие этапы необходимо пройти, какие методы и средства использовать, как организовать контроль за продвижением проекта и качеством выполнения работ – эти и другие вопросы решаются методологиями программной инженерии.

В настоящее время существует ряд общих методологий разработки ИС. Главное в них – единая дисциплина работы на всех этапах жизненного цикла системы, учет критических задач и контроль их решения, применение развитых инструментальных средств поддержки процессов анализа, проектирования и реализации ИС. Для успешной реализации проекта объект проектирования (ИС) должен быть, прежде всего, адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели ИС. Накопленный к настоящему времени опыт проектирования ИС показывает, что это логически сложная, трудоемкая и длительная по времени работа, требующая высокой квалификации участвующих в ней специалистов. Однако до недавнего времени проектирование ИС выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования ИС. Кроме того, в процессе создания и функционирования ИС информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

Для различных классов систем используются разные методы разработки, определяемые типом создаваемой системы и средствами реализации. Спецификации этих систем, в большинстве случаев, состоят из двух основных компонентов – функционального и информационного. Современные методы создания ИС разного назначения базируются в основном, на трех подходах: объектно-ориентированная технология, основанная на знаниях **(интеллектуальная) технология** и **CASE-технология** (см. рисунок 1.1):

Рисунок 1.1 – Современные концепции создания ИС

В области создания систем САПР доминируют структурные подходы, так как они максимально приспособлены для взаимодействия с пользователями, не являющимися специалистами в области ИТ. Адекватными инструментальными средствами, поддерживающими структурный подход к созданию ИС, являются CASE-системы автоматизации проектирования.

Принципы и методы создания ИС

Еще в 60-е годы прошлого столетия были сформулированы шесть основополагающих принципов, на которые необходимо опираться в процессе создания ИС: новых задач; системного подхода; первого руководителя; разумной типизации проектных решений; непрерывного развития системы; минимизации ввода-вывода информации. Развитие технической основы создания компьютеров и ИТ привело к переформулированию этих принципов и в ГОСТ РД 50-680-88 к ним отнесены следующие: системность, развитие (открытость), совместимость, стандартизация (унификация) и эффективность.

Разработка сложных ИС - информационных систем - предприятий, невозможна без тщательно обдуманного методологического подхода. Какие этапы необходимо пройти, какие методы и средства использовать, как организовать контроль за продвижением проекта и качеством выполнения работ – эти и другие вопросы решаются методологиями программной инженерии.

В настоящее время существует ряд общих методологий разработки ИС. Главное в них – единая дисциплина работы на всех этапах жизненного цикла системы, учет критических задач и контроль их решения, применение развитых инструментальных средств поддержки процессов анализа, проектирования и реализации ИС.

Принципы создания ИС

Принцип системности.

По принципу системности ИС следует рассматривать как системы, структура которых определяется функциональным назначением. (ИС рассматривается как система, что позволяет выявить многообразные типы связей между структурными элементами).

Системный подход предполагает учет всех этих взаимосвязей, анализ отдельных частей системы как ее самостоятельных структурных составляющих и параллельно - выявление роли каждой из них в функционировании всей системы в целом. Таким образом, реализуются процессы анализа и синтеза, фундаментальный смысл которых -разложение целого на составные части и воссоединение целого из частей.

Принцип системности заключается в том, что при декомпозиции должны быть

установлены такие связи между структурными компонентами системы, которые обеспечивают цельность корпоративной системы и ее взаимодействие с другими системами.

Нельзя разрабатывать какую-либо задачу автономно от других и реализовывать только отдельные ее аспекты. Задача должна рассматриваться комплексно со всеми возможными информационными связями.

Пример:

Отбор персонала на вакантные рабочие места. Ее решение должно осуществляться с учетом следующих моментов:

– использования результатов периодически проводимого профессионального и психофизиологического тестирования работников;

– анализа результатов периодически проводимой аттестации рабочих мест;

– анализа показателей трудовой дисциплины персонала;

– разработки общих и дополнительных критериев отбора (при наличии нескольких претендентов на одно рабочее место);

– использования банка данных претендентов, сформированного ранее;

– индивидуального собеседования;

– анализа анкетных данных и резюме (если претендент не является членом трудового коллектива).

Принцип развития (открытости)

Принцип новых задач (развития) - возможность постоянного пополнения и обновления ИС. Число решаемых задач постоянно увеличивается и меняется методика их решения.

Заключается в том, что внесение изменении в систему, обусловленных самыми различными причинами (внедрением новых информационных технологии, изменением законодательства, организационной перестройкой внутри фирмы и т. п.), должно осуществляться только путем дополнения системы без переделки уже созданного, т. е. не нарушать ее функционирования. Реализовать данный принцип на практике достаточно сложно, так как он требует очень глубокой аналитической предпроектной работы. Необходимо разделить решаемые задачи на определенные группы и для каждой из них предусмотреть возможные направления развития (например, выход в глобальные сети, применение средств для сканирования документов, шифрование информации).

В любой фирме на протяжении ряда лет применяются традиционно сложившиеся методы и приемы управления. Но ситуация в компьютерном мире и в сфере экономики изменяется постоянно: модифицируется элементная база компьютеров, что делает их более мощными; появляются новые средства передачи и хранения данных; расширяются границы доступа к данным; вступают в силу новые законы и т.д. Все это необходимо учитывать как при решении традиционных задач (корректировании технологии решения, методов ввода, вывода и передачи информации), так и при постановке новых задач, принципиальное решение которых оказывается возможным только в условиях новых технологий.

Если не отслеживать эти изменения и, тем более, не поспевать за ними, можно отстать от остальных пользователей и тем самым перс крыть доступ к общению с ними, а это абсолютно недопустимо, поскольку информационная изоляция имеет только негативные последствия.

Принцип совместимости

Принцип совместимости – способность взаимодействия ИС различных видов, уровней в процессе их совместного функционирования.

Заключается в том, что при создании системы должны быть реализованы информационные интерфейсы, благодаря которым она может взаимодействовать с другими системами согласно установленным правилам. В современных условиях это особенно касается сетевых связей локального и глобального уровней.

Если в локальных сетях относительно несложно установить и соблюдать стандарты "общения" отдельных бизнес-процессов между собой и со смежными системами, то выход в глобальные сети требует:

- дополнительных ужесточенных мер по защите информации;
- знания и соблюдения различного рода протоколов, регламентирующих все пилы информационных обменов;
- знание сетевого этикета, предусматривающего такие правила, как:
  - регулярная проверка своей электронной почты;
  - периодическая чистка своего почтового ящика;
  - корректность в составлении сообщений;
  - указание координат для обратной связи и т.п.

Принцип стандартизации (унификации)

Принцип стандартизации и унификации – необходимость применения типовых, унифицированных и стандартизованных элементов функционирования ИС.

При создании системы должны быть рационально использованы типовые, унифицированные и стандартизованные элементы, проектные решения, пакеты прикладных программ, комплексы, компоненты.

Задачи необходимо разрабатывать таким образом, чтобы они подходили к возможно более широкому кругу объектов. Игнорирование именно этого принципа привело в свое время к тому, что подсистема УК, несмотря на традиционный перечень задач и алгоритмов их решения, разрабатывалась на каждом предприятии самостоятельно, что привело к совершенно неоправданному расходу трудовых, материальных, финансовых и временных ресурсов.

В современных разработках пакетов прикладных программ (ППП) рассматриваемый принцип задействован. Однако при знакомстве с конкретным ППП необходимо обращать внимание на сущность реализации типовых решений, поскольку каждый разработчик по-своему "видит" такие решения. Например, во многих пакетах по управлению кадрами присутствует задача "Отбор кадров". Однако в пакете фирмы Infin она реализована достаточно оригинально. Решение ее заключается в следующем. Экран разделен на две половины. Слева выводится достаточно большой список мужских и женских имен, по которому перемещается курсор. Если интересующее имя отмечено, то для него с правой стороны экрана приводится текст, и котором сообщается о том, кого обозначает имя и какими чертами характера обладает человек, имеющий его. Относиться к подобному подходу можно по-разному. Но можно сказать определенно -такого рода информации явно недостаточно для решения задачи и ограничиваться только ею нельзя.

Принцип эффективности

Принцип эффективности – достижение рационального соотношения между затратами на создание ИС и целевым эффектом, получаемым при ее функционировании. (Эффективность ИС следует рассматривать как интегральный (обобщенный) показатель уровня реализации приведенных выше принципов, отнесенного к затратам по созданию и эксплуатации системы).

Предусматривает достижение рационального соотношения между затратами на создание системы и целевыми эффектами, включая конечные результаты, отражающиеся на прибыльности и получаемые по окончании внедрения автоматизации в управленческие процессы.

Перечень рассмотренных принципов создания корпоративных систем взят из ГОСТ. Однако к их числу с полным правом можно отнести еще один из тех, которые были сформулированы в 60-е годы и по сей день не потеряли своей актуальности. Это -Принцип первого руководителя. Чрезвычайно важный принцип, распространяющийся на все сферы управленческой деятельности. Уровень компетентности руководителя любого уровня в производственных, административных, психологических и других вопросах определяет

общие тенденции развития фирмы или се подразделении и социально-психологический климат в коллективе. Известно, что устойчивое бесконфликтное взаимопонимание среди персонала способствует росту творческих начал и эффективной повседневной деятельности. И именно руководитель и первую очередь должен обеспечивать все элементы стабильности. Сформировать такой коллектив достаточно сложно и далеко не каждый руководитель способен это сделать. Напротив, негативное отношение руководителя к каким-либо нововведениям является тормозом в развитии творческой и профессиональной инициативы работников всех категорий.

Принцип первого руководителя – на предприятии должен быть человек, одинаково заинтересованный в развитии автоматизации всех сторон деятельности предприятия.

Принцип устойчивости заключается в том, что ИС должна выполнять основные функции независимо от воздействия на нее внутренних и внешних возможных факторов. Это значит, что неполадки в отдельных ее частях должны быть легко устранимы, а работоспособность системы – быстро восстановима.

Принцип гибкости означает приспособляемость системы к возможным перестройкам благодаря модульности построения всех подсистем и стандартизации их элементов.

План конспекта:

Основополагающие принципы создания ИС. Принцип системности. Принцип развития (открытости). Принцип современности. Принцип стандартизации (унификации). Принцип эффективности.

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Тема самостоятельного изучения № 2. Этапы создания ИС

Вид деятельности студентов: подготовка к лекциям, самостоятельное изучение литературы, подготовка и выполнение лабораторных работ, выполнение контрольной работы

Итоговый продукт самостоятельной работы: конспект

Средства и технологии оценки: отчет

Краткие теоретические сведения

Этапы создания информационных систем (ИС)

Выделяются несколько этапов создания ИС:

I этап – предпроектный (обследование, составление отчета, технико-экономического обоснования и технического задания);

II этап – проектный (составление технического и рабочего проектов);

III этап – внедрение (подготовка к внедрению, проведение опытных испытаний и сдача в промышленную эксплуатацию);

IV этап – сопровождение и анализ функционирования (выявление проблем, внесение изменений в проектные решения и существующие ИС).

Содержание документации на каждой стадии определяется составом и спецификой работ. Стадии детализируются и включают следующие этапы:

Предпроектная стадия:

- обследование объекта и обоснование необходимости создания ИС;
- формирование требований пользователя к ИС;

– оформление отчета о выполненной работе и заявки на разработку ИС;
– разработка и утверждение технического задания ИС.

Проектная часть:

– разработка проектных решений по системе и ее частям;
– разработка документации на ИС;
– разработка и оформление документации на поставку изделий для комплектования ИС;
– разработка рабочей документации на систему или ее части;
– разработка или адаптация программ.

Стадия внедрения:

– подготовка объекта автоматизации к вводу в действие;
– подготовка персонала, проводится обучение персонала;
– строительно-монтажные работы, в том случае, если строится специализированное здание;
– проведение предварительных испытаний;
– проведение опытной эксплуатации;
– проведение опытных испытаний;
– введение в промышленную эксплуатацию.

Анализ функционирования:

– гарантийное и послегарантийное обслуживание;
– внесение изменений в проектные решения.

Основными участниками процесса создания ИС являются предприятие-заказчик, для которого она создается и предприятие-разработчик, выполняющий работы по проектированию ИС. Юридические и организационные взаимоотношения конкретно заказчиков и разработчиков регулируются заключенными между ними договорами.

Заказчик обязан заключить договор на создание ИС, приобрести технические средства, подготовить задание на строительство или реконструкцию помещения, если необходимо, совместно с разработчиком выполнить работы предпроектной стадии, в необходимые сроки подготовить помещение, приобрести и установить технические средства, разработать и осуществить мероприятия по совершенствованию организации управления и производства. На стадии проектирования необходимо обеспечить обучение персонала, обеспечить запись необходимой информации на машинные носители и ее контроль, обеспечить уточнение исходных данных по составу и структуре информационной базы, завершить ее формирование, подготовить контрольные примеры, организовать поэтапную приемку рабочих программ с проверкой на контрольных примерах. При подготовке объекта к внедрению заказчик выполняет следующие работы: внедряет локальные и общегосударственные классификаторы, унифицированные формы документов, проводит в намеченные сроки мероприятия по подготовке объекта к внедрению ИС. При вводе системы в действие заказчик завершает ввод в эксплуатацию технических средств, завершает опытную эксплуатацию комплекса задач и принимает в промышленную эксплуатацию. Разрабатывает и согласовывает с разработчиком программу приема сдаточных испытаний и организуют работу приемочной комиссии по проведению испытаний системы.

Основная цель разработчика – создание ИС. На предпроектной стадии проводит обследование объекта, обрабатывает материалы обследования, определяет задачи, комплексы задач, подлежащие автоматизации, определяет экономическую эффективность. На стадии ТП разрабатывает документацию, в соответствии с утвержденным ТЗ осуществляет методическое руководство работами по созданию классификаторов, внедрению унифицированных систем документации, разрабатывает структуру информационной базы, принимает участие в обучении персонала заказчика. На стадии рабочей документации осуществляет разработку программного обеспечения,

генерацию рабочих программ, участвует в разработке должностных инструкций управленческого персонала, технологических инструкций пользователя. При вводе системы в действие разработчик осуществляет методическое руководство, вносит корректировки в проекты, принимает участие в сдаче задач и комплексов задач в промышленную эксплуатацию и участвует в работе комиссии по приемке системы в промышленную эксплуатацию.

План конспекта:

Предпроектный этап создания ИС (обследование, составление отчета, технико-экономического обоснования и технического задания). Проектный этап создания ИС (составление технического и рабочего проектов). Внедрение ИС (подготовка к внедрению, проведение опытных испытаний и сдача в программную эксплуатацию). Анализ функционирования ИС (выявление проблем, внесение изменений в проектные решения и существующие АИС и АИТ).

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Тема самостоятельного изучения № 3. Требования, предъявляемые к технологии создания ИС

Вид деятельности студентов: подготовка к лекциям, самостоятельное изучение литературы, подготовка и выполнение лабораторных работ, выполнение контрольной работы

Итоговый продукт самостоятельной работы: конспект

Средства и технологии оценки: отчет

Краткие теоретические сведения

Требования, предъявляемые к технологии создания ИС

Информационная система должна соответствовать требованиям гибкости, надежности, эффективности и безопасности.

Гибкость

Гибкость, способность к адаптации и дальнейшему развитию подразумевают возможность приспособления информационной системы к новым условиям, новым потребностям предприятия. Выполнение этих условий возможно, если на этапе разработки информационной системы использовались общепринятые средства и методы документирования, так что по прошествии определенного времени сохранится возможность разобраться в структуре системы и внести в нее соответствующие изменения, даже если все разработчики или их часть по каким-либо причинам не смогут продолжить работу.

Любая информационная система рано или поздно морально устареет, и станет вопрос о ее модернизации или полной замене. Разработчики информационных систем, как правило, не являются специалистами в прикладной области, для которой разрабатывается система. Участие в модернизации или создании новой системы той же группы проектировщиков существенно сократит сроки модернизации.

Вместе с тем возникает риск применения устаревших решений при модернизации системы. Рекомендация в таком случае одна — внимательнее относиться к подбору разработчиков информационных систем.

Надежность

Надежность информационной системы подразумевает ее функционирование без искажения информации, потери данных по «техническим причинам».

Требование надежности обеспечивается созданием резервных копий хранимой информации, выполнения операций протоколирования, поддержанием качества каналов связи и физических носителей информации, использованием современных программных и аппаратных средств. Сюда же следует отнести защиту от случайных потерь информации в силу недостаточной квалификации персонала.

Эффективность

Система является эффективной, если с учетом выделенных ей ресурсов она позволяет решать возложенные на нее задачи в минимальные сроки.

В любом случае оценка эффективности будет производиться заказчиком, исходя из вложенных в разработку средств и соответствия представленной информационной системы его ожиданиям.

Негативной оценки эффективности информационной системы со стороны заказчика можно избежать, если представители заказчика будут привлекаться к проектированию системы на всех его стадиях. Такой подход позволяет многим конечным пользователям уже на этапе проектирования адаптироваться к изменениям условий работы, которые иначе были бы приняты враждебно.

Активное сотрудничество с заказчиком с ранних этапов проектирования позволяет уточнить потребности заказчика. Часто встречается ситуация, когда заказчик чего-то хочет, но сам не знает чего именно. Чем раньше будут учтены дополнения заказчика, тем с меньшими затратами и в более короткие сроки система будет создана.

Кроме того, заказчик, не являясь специалистом в области разработки информационных систем, может не знать о новых информационных технологиях. Контакты с заказчиком во время разработки для него информационной системы могут подтолкнуть заказчика к модернизации его аппаратных средств, применению новых методов ведения бизнеса, что отвечает потребностям как заказчика, так и проектировщика. Заказчик получает рост эффективности своего предприятия, проектировщик — расширение возможностей, применяемых при проектировании информационной системы.

Эффективность системы обеспечивается оптимизацией данных и методов их обработки, применением оригинальных разработок, идей, методов проектирования (в частности, спиральной модели проектирования информационной системы, о которой речь пойдет в следующих главах).

Не следует забывать и о том, что работать с системой придется обычным людям, являющимся специалистами в своей предметной области, но зачастую обладающим весьма средними навыками в работе с компьютерами. Интерфейс информационных систем должен быть им интуитивно понятен. В свою очередь, разработчик-программист должен понимать характер выполняемых конечным пользователем операций. Рекомендациями в этом случае могут служить повышение эффективности управления разработкой информационных систем, улучшение информированности разработчиков о предметной области.

Безопасность

Под безопасностью, прежде всего, подразумевается свойство системы, в силу которого посторонние лица не имеют доступа к информационным ресурсам организации, кроме тех, которые для них предназначены. Защита информации от постороннего доступа обеспечивается управлением доступом к ресурсам системы, использованием современных программных средств защиты информации. В крупных организациях целесообразно создавать подразделения, основным направлением деятельности которых было бы обеспечение информационной безопасности, в менее крупных организациях назначать сотрудника, ответственного за данный участок работы.

Система, не отвечающая требованиям безопасности, может причинить ущерб интересам заказчика, прежде всего имущественным.

Помимо злого умысла, при обеспечении безопасности информационных систем

приходится сталкиваться еще с несколькими факторами. В частности, современные информационные системы являются достаточно сложными программными продуктами. При их проектировании с высокой вероятностью возможны ошибки, вызванные большим объемом программного кода, несовершенством компиляторов, человеческим фактором, несовместимостью с используемыми программами сторонних разработчиков в случае модификации этих программ и т. п. Поэтому за фазой разработки информационной системы неизбежно следует фаза ее сопровождения в процессе эксплуатации, в которой происходит выявление скрытых ошибок и их исправление.

Требование безопасности обеспечивается современными средствами разработки информационных систем, современной аппаратурой, методами защиты информации, применением паролей и протоколированием, постоянным мониторингом состояния безопасности операционных систем и средств их защиты. И, наконец, самый важный фактор, влияющий на процесс разработки, — знания и опыт коллектива разработчиков информационных систем.

План конспекта:

Требования к стандарту проектирования:

- набор необходимых моделей и степень их детализации
- правила фиксации проектных решений на диаграммах
- требования к конфигурации рабочих мест разработчиков
- механизм обеспечения совместной работы над проектом

Требования к стандарту оформления проектной документации:

- комплектность, состав и структуру документации на каждой стадии проектирования
- требования к её оформлению
- правила подготовки, рассмотрения, согласования и утверждения документации
- требования к настройке издательской системы
- требования к настройке CASE-средств

Требования к стандарту интерфейса пользователя:

- правила оформления экранов, состав и расположение окон и элементов управления
- правила использования клавиатуры и мыши
- правила оформления текстов помощи
- перечень стандартных сообщений
- правила обработки реакции пользователя

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Тема самостоятельного изучения № 4. Выбор технологии создания ИС. Внедрение технологий создания ИС.

Вид деятельности студентов: подготовка к лекциям, самостоятельное изучение литературы, подготовка и выполнение лабораторных работ, выполнение контрольной работы

Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: отчет
Краткие теоретические сведения
Выбор технологии создания ИС

В жизненном цикле современных информационных систем (ИС) можно выделить

два этапа: этап создания и этап эксплуатации. На первом из них осуществляется постановка задачи, проектирование системы и ее реализация. Иначе говоря, происходит декомпозиция задачи, т. е. переход от ее постановки к реализации отдельных функций, а затем объединение отдельных функций в приложения для конечных пользователей.

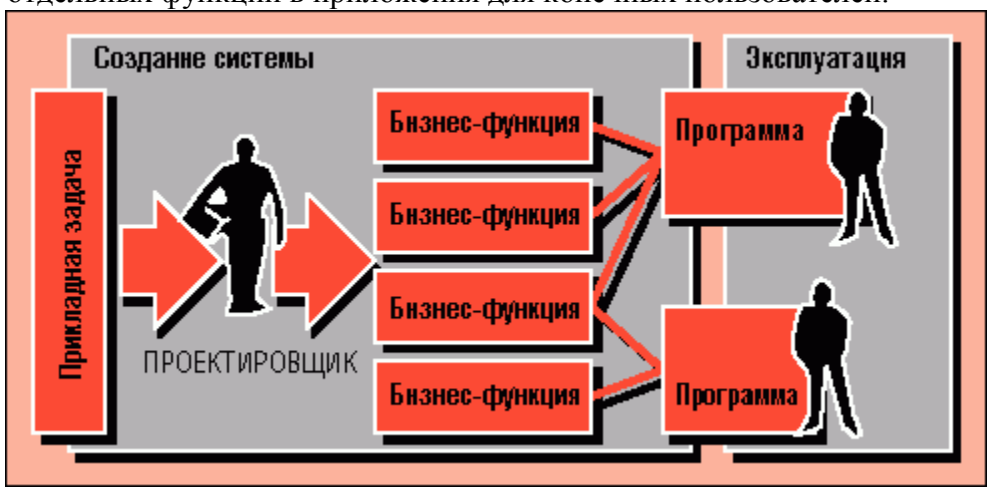

Рисунок 1 - Стандартная схема создания информационных систем

На этапе эксплуатации ИС появляется потребность в развитии ее функциональных возможностей (функционала). Само существование системы показывает, что знания о предметной области систематизированы. На основе этих знаний возникают новые, приводящие к выявлению новых прикладных задач и, как следствие, к формулированию новых требований к системе. Эти требования зарождаются на рабочих местах в процессе использования системы. Их накопление обычно ведет к созданию ее следующей версии.

Таким образом, этап эксплуатации системы подразумевает ее поддержание (администрирование) и накопление требований для следующей версии.

Не будем углубляться в процесс поддержания готовой системы, а сосредоточимся на вопросе накопления требований. Они могут касаться как изменения имеющегося, так и появления нового функционала.

Для реализации прикладной задачи в процессе эксплуатации системы используется заранее заложенная в нее композиция отдельных функций. В настоящее время проблема большинства проектов состоит в том, что способ композиции (объединения) этих функций определяется на этапе построения системы, поскольку, согласно существующим методологиям создания ИС, функционал выделяется в привязке к пользователю.

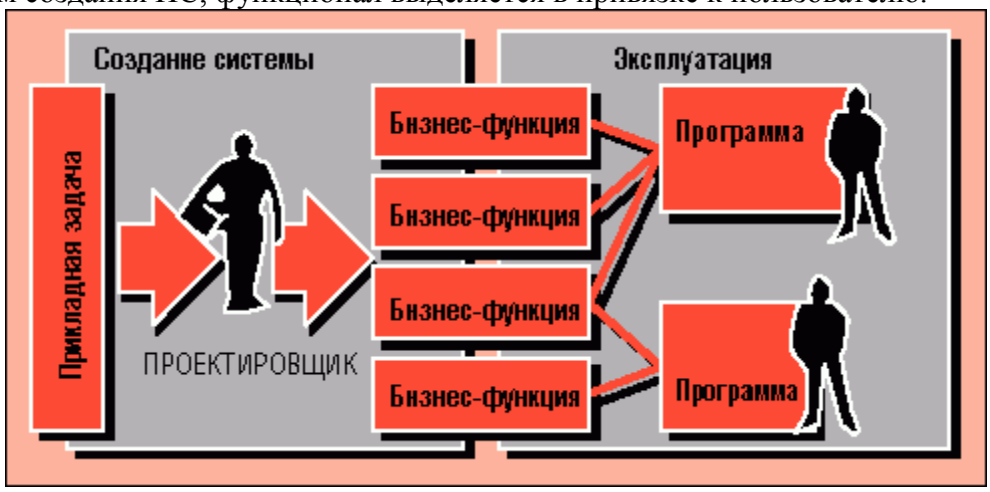

Рисунок 2 - Предлагаемая схема создания информационных систем

Проблемы, которые порождает существующий подход, очевидны:

- невозможно быстро изменять функционал приложения при изменении требований к прикладной задаче;
- с появлением новых типов пользователей нельзя создать приложение,

использующее имеющиеся функции системы (не возвращаясь на этап создания);

- накопленный функционал остается в лучшем случае только на бумаге, отлаженный код не переходит из одной версии системы в другую.

Решение проблем

Логично несколько изменить сами этапы жизненного цикла информационных систем, не отрицая при этом существующих методологий. В частности, ограничить этап создания ИС стадией разработки бизнес-функций и перенести процесс композиции отдельных функций на этап эксплуатации.

При этом становятся возможными следующие вещи:

- Сборка приложения для конкретного пользователя с произвольной функциональностью. Следуя предложенной методике, можно собрать приложение, использующее уже реализованный функционал.

- Вовлечение старых работников самих предприятий в создание и изменение функциональности новых приложений. Эти люди лучше других знают, что нужно их бизнесу.

- Накопление библиотеки готового функционала (элементов бизнеса) и порождение из него произвольного набора бизнес-функций (сервисов).

Идея подобной разработки информационных систем далеко не нова, но конкретная реализация всегда упиралась в неразрешимые проблемы. Для рассмотрения деталей необходимо определиться со словарем терминов, употребляемых в данной предметной области. Авторами предлагается следующая терминология:

- бизнес - совокупность сервисов, описываемых информационной системой предприятия;

- сервис - набор бизнес-функций, направленных на решение определенной проблемы предприятия;

- бизнес-функция - составная часть сервиса, отвечает за решение определенной задачи. Законченное действие пользователя;

- логическая функция - составная часть бизнес-функции;

- компонент - механизм реализации логических функций. Один компонент может участвовать в реализации нескольких логических функций.

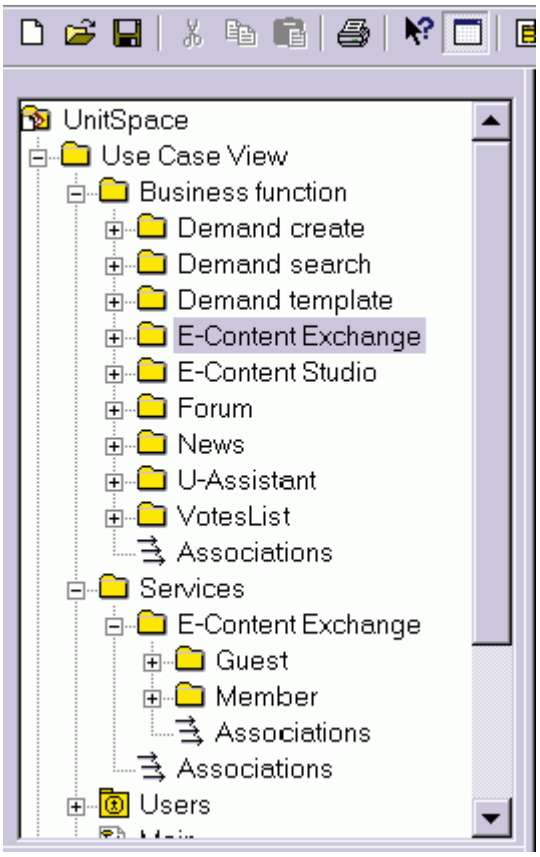

Рисунок 3 - Список бизнес-функций

Опираясь на этот словарь терминов, можно перейти к более подробному рассмотрению встречающихся проблем.

- Отсутствие универсального механизма для связывания набора полученных компонентов. Механизм, объединяющий компоненты, должен быть достаточно гибким, позволяющим взаимодействовать распределенным компонентам. Кроме того, необходимо, чтобы он был открытым для сторонних разработчиков, использовал промышленные стандарты, принятые в отрасли, и мог работать в гетерогенных средах.

- Реализация крупных систем подразумевает применение нескольких серверов, на которых устанавливаются элементы системы. И здесь остро встает вопрос создания единой службы имен. Система должна предоставлять пользователю возможность прозрачной навигации внутри самой себя.

- Стандартизация уровней построения системы, т. е. разграничение ответственности между проектировщиками, программистами и теми, кто эксплуатирует систему. Системы, использующие исходный код для построения нового функционала, стоят дорого. Мы предлагаем более дешевое решение - сборку приложений из готовых компонентов.

- Отсутствие простых механизмов, позволяющих работникам предприятий формировать новые сервисы. Для решения этой проблемы необходимо создать систему хранения информации обо всех бизнес-процессах, протекающих на предприятии, с одновременной связкой их с выделенными сервисами и реализацией.

Предлагаемая технология создания информационных систем частично обходит эти проблемы, а частично дает инструмент для их решения. Она базируется на использовании собственного программного обеспечения, классических методологий и CASE-средств фирмы Rational. Основная идея технологии - дать возможность формировать сервисы на этапе эксплуатации (т. е. исключить из этапа создания системы объединение бизнес-функций в сервисы).

Предлагаемая технология обладает множеством достоинств. Ниже перечислены некоторые из них.

1. Для обеспечения связи между компонентами разработанное ПО поддерживает

промышленные стандарты отрасли - технологии CORBA, JMS, Oracle AQ.

2. Единая служба имен и справочник сервисов реализованы с использованием LDAP и JNDI. Это позволяет решить все вопросы с аутентификацией пользователя на различных серверах и платформах.

3. Проектировщики и разработчики системы разрабатывают бизнес-функции, а эксплуатационники объединяют их в сервисы и выдают конечным пользователям. Стоимость внедрения и эксплуатации подобной системы значительно снижается за счет того, что каждый член команды работает в привычной ему среде.

4. Использование CASE-средств для ведения проектов позволяет хранить всю информацию в одном месте на этапах создания, проектирования и эксплуатации, при этом исключаются дублирование и разночтения между различными аспектами системы. Одновременно это позволяет бизнес-менеджерам и специалистам в конкретных предметных областях создавать необходимые сервисы методом “перетащи и оставь”.

5. Применение CASE-средств подразумевает разработку информационной системы в несколько этапов и позволяет использовать бизнес-менеджеров не только для корректировки направления развития системы, но и для ее эксплуатации. Менеджеры могут определять функциональность сервисов на этапе эксплуатации.

6. Использование классических методологий разработки софтверных проектов и, как следствие, документированность и прогнозируемость развития проекта. Получение документированного проекта позволяет компании-клиенту пользоваться услугами нескольких фирм-разработчиков.

7. Удешевление внедрения за счет разделения ответственности между проектировщиками и разработчиками, с одной стороны, и бизнес-пользователями - с другой.

Дополнительным достоинством данного подхода является его применимость при создании сложных, многоаспектных систем. На сегодняшний день не существует рекомендаций или методик для работы с подобными системами. И все бремя ответственности ложится на плечи проектировщиков и аналитиков. Рассмотрение функционала предприятия без привязки к пользователю позволяет реализовать набор бизнес-функций, из которых на этапе эксплуатации возможно построение произвольного набора сервисов в рамках данного бизнеса.

Примеры применения

Проектировщик, проводя декомпозицию предметной области, связанной с управлением электронным контентом, выделяет существующие бизнес-функции. Описание выполняется совместно со специалистом в предметной области и оформляется в виде текста, содержащего название бизнес-функции, описание последовательности действий пользователя для ее реализации и список нефункциональных требований (качество, надежность, масштабируемость и т. д.)

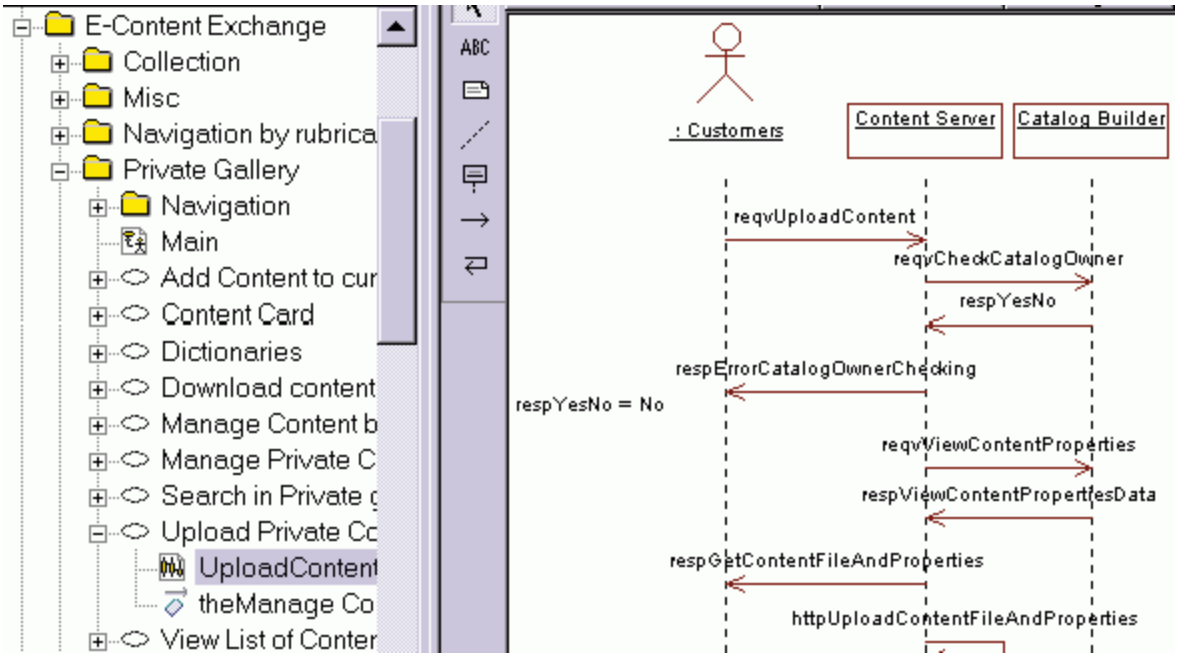

Рисунок 4 - Описание механизма вызова компонентов

Проектируя механизм реализации бизнес-функций, проектировщик выполняет декомпозицию до уровня компонентов. Вся последовательность шагов документируется в Rational Rose, что позволяет бизнес-менеджерам других отделов контролировать процесс разработки и, если нужно, влиять на него.

Проектировщик выделяет из бизнес-функции логические функции, проектирует компоненты, т. е. определяет наборы логических функций и спецификации реализующих их компонентов. Для каждой логической функции прописывается сценарий ее выполнения, включая описание передаваемых данных, последовательности действий и список используемых компонентов.

Далее для каждого спроектированного компонента необходимо получить задание для программиста. С использованием CASE-средств это можно сделать автоматически, так как информация была заложена проектировщиком и доступна в проекте. Задание для программиста содержит набор диаграмм, показывающих последовательность использования компонентов для реализации бизнес-функций, а также спецификацию интерфейса по каждому компоненту. Отдельно прилагается документ, содержащий набор нефункциональных требований.

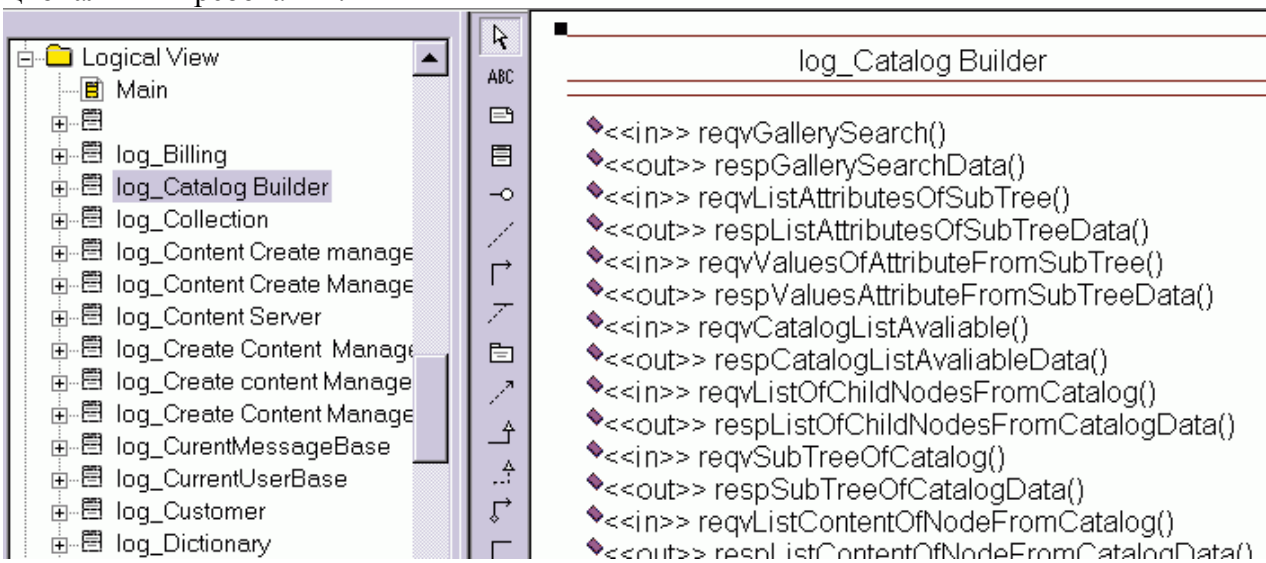

Рисунок 5 - Описание интерфейса компонентов

В процессе работы над системой список готовых компонентов растет, и для реализации новых логических функций уже не надо формировать задание для программиста. Новые бизнес-функции реализуются готовыми компонентами. Использование же готовых компонентов не представляет сложности, так как все они содержатся в проекте и к тому же документированы.

После завершения проектирования и реализации набора компонентов менеджер отдела маркетинга определяет группы пользователей, для которых будет доступна реализованная бизнес-функция.

Вся работа выполняется бизнес-менеджером методом “перетащи и оставь” в Rational Rose.

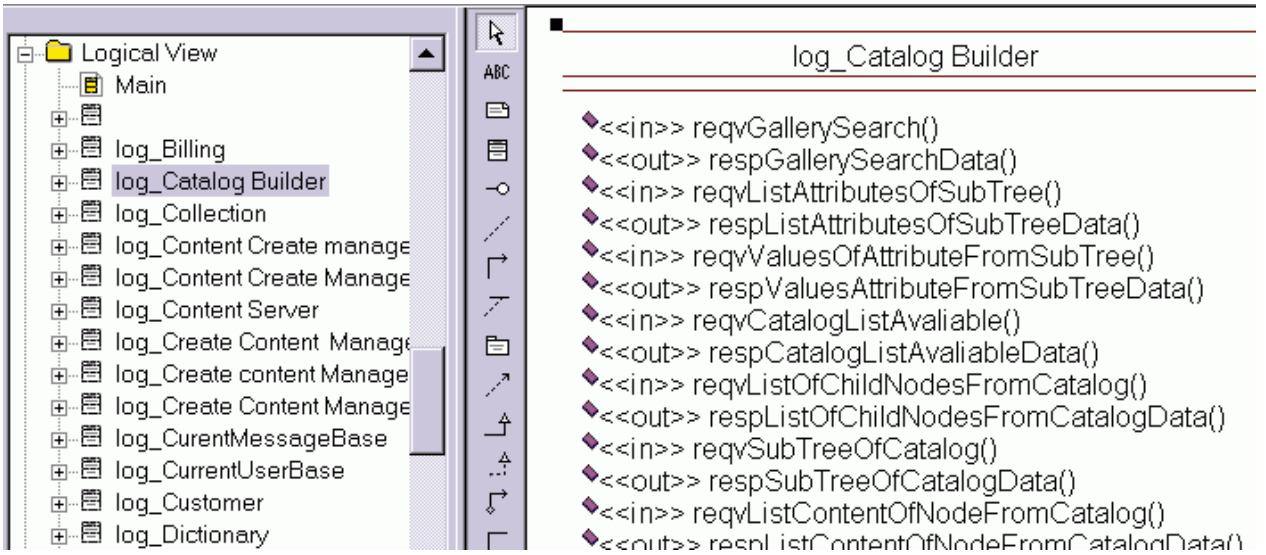

Рисунок 6 - Определение прав пользователей

Следующими шагами построения информационной системы (или Web-ресурса) является размещение информации о пользователях и сервисах в LDAP-сервере и настройка системы взаимодействия компонентов. Эти операции также производятся автоматически с использованием информации из проекта, созданного в CASE-средстве.

Таким образом, процесс создания бизнес-функционала тесно интегрируется с современными CASE-средствами фирмы Rational, а также использует передовые методологии объектно-ориентированного анализа и проектирования (RUP).

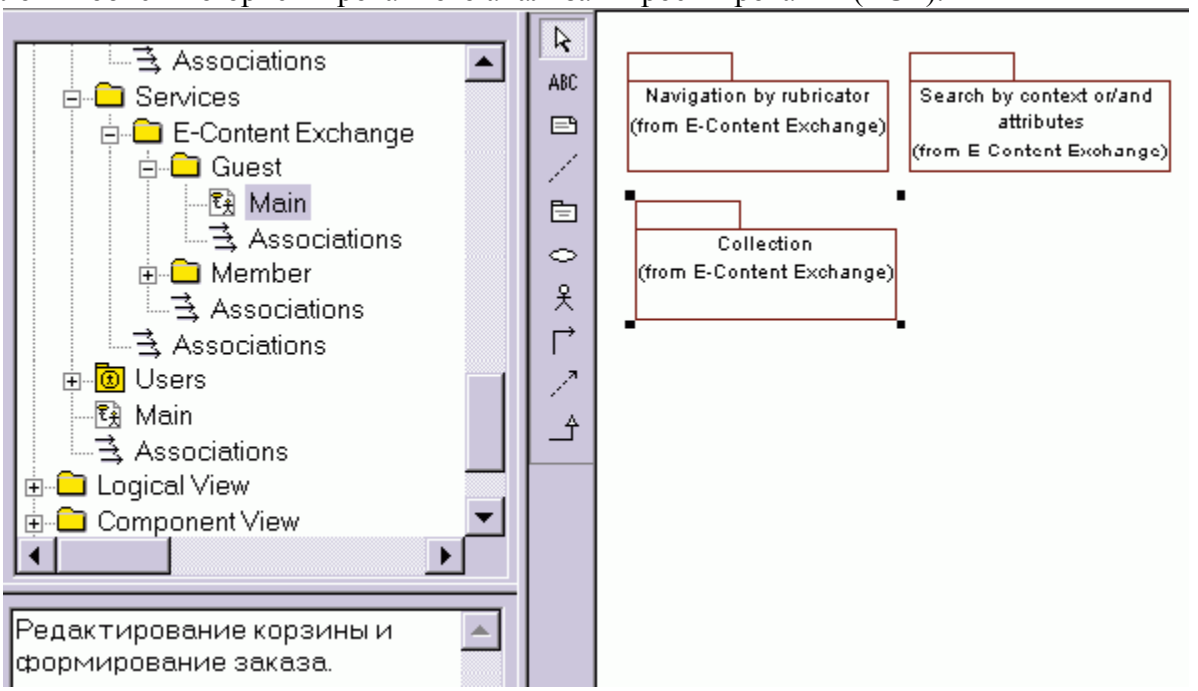

Рисунок 7 - Структура LDAP-хранилища

В заключение можно сказать, что предлагаемая методика применялась фирмой UnitSpace при создании нескольких проектов на американском и российском рынках электронного бизнеса. Более конкретно это можно рассмотреть на примере проектов “Восторг” и E-content Exchange.

Проект “Восторг” (www.vostorg.ru) - это портал, предоставляющий своим пользователям весь спектр решений в области электронной коммерции - от создания простейших витрин магазинов до крупных супермаркетов, которые обеспечивают работу со многими поставщиками, а также отдельных магазинов и групп магазинов, предоставляющих покупателю общую корзину. Обслуживающему персоналу каждого магазина предлагается мощный back-office.

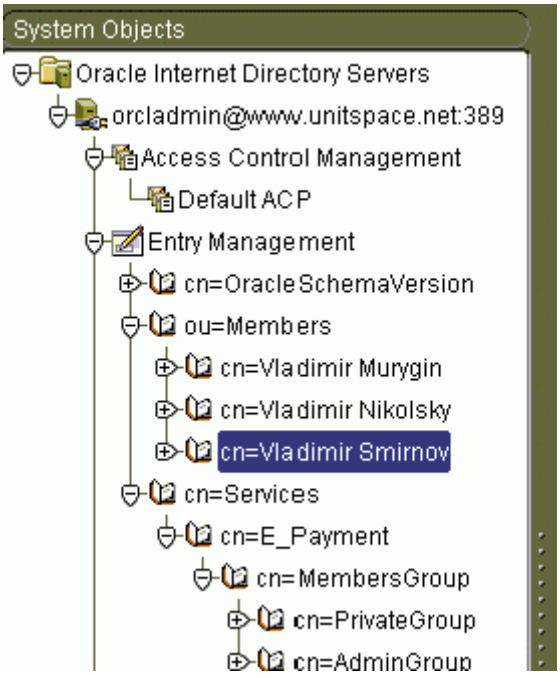

Рисунок 8 - Примеры реализации моделей проектов

Проект E-content Exchange (www.unitspace.net) - это портал компании UnitSpace. Он содержит богатую библиотеку э-контента (фотографии товаров, текстовые описания, презентации и пр.) и сервисы, предназначенные для создания виртуальных промостраниц, промосайтов и каталогов товаров и интеграции их на страницы сайта или электронного магазина, что поможет повысить эффективность представления продукции в Интернете.

Испытание описанной технологии в компании UnitSpace подтверждает ее эффективность.

Во-первых, разработка проектов стала более структурированной, модель проекта в CASE-средстве содержит описание предметной области, наработанного функционала, а также сервисов, предлагаемых каждой информационной системой. Это позволяет в любой момент автоматически получить необходимую документацию по любому этапу проекта.

Во-вторых, специалистам UnitSpace удалось сократить время разработки за счет использования готовых компонентов. Информация о готовых компонентах из модели проекта существенно сокращает стоимость этапа проектирования.

Далее, менеджеры отдела маркетинга получили возможность создавать сервисы, предоставляемые системой, используя наработанный функционал бизнес-функций. Это сократило время, необходимое для разработки новых сервисов, и снизило стоимость их создания, что в свою очередь придало системе динамичность и повысило ее привлекательность для пользователя.

Использование собственных разработок и обобщенной спецификации для написания компонентов, объединенной с адаптацией объектно-ориентированной методологии под собственные технологии, а также творческий подход к применению CASE-средств позволили фирме UnitSpace разработать законченную методику создания информационных

систем произвольной сложности на промышленной основе.

Внедрение технологий создания ИС

Внедрение информационной системы управления предприятием, как и любое серьезное преобразование на предприятии, является сложным и зачастую болезненным процессом. Тем не менее, некоторые проблемы, возникающие при внедрении системы, достаточно хорошо изучены, формализованы и имеют эффективные методологии решения. Заблаговременное изучение этих проблем и подготовка к ним значительно облегчают процесс внедрения и повышают эффективность дальнейшего использования системы. Первейшим этапом создания системы должно быть проведение работ по предпроектному обследованию (так называемый консалтинг). Пока не описаны и не проанализированы все бизнес-процессы предприятия, не построена модель предприятия «как есть сегодня», не сформулированы обоснованные требования к новой системе, не построена модель будущей системы «как должно быть», не разработано техническое задание не может быть и речи о покупке или начале разработки системы. Цель этой предпроектной работы заключается в том, чтобы разработать представление о будущей системе, описать функционально-информационную модель будущей системы и защитить ее перед заказчиком. Только после этого можно вкладывать деньги в покупку или разработку системы.

Подготовка предприятия к реализации ИС

– *Подготовка нормативно-справочной информации.*

– Разработка методик подготовки и ведения нормативно-справочной информации.

– Разработка классификации объектов нормативно-справочной информации, их определение и детальное описание их свойств. Подготовка образцов описания данных объектов.

Базовый состав объектов нормативно-справочной информации включает:

- производственную структуру предприятия (рабочие центры и их группировки, их идентификация и классификация);
- территориальную структуру предприятия (площадки и места хранения запасов и их группировки, их идентификация и классификация);
- финансовую структуру предприятия (центры финансовой ответственности и их группировки, их идентификация и классификация);
- номенклатурные позиции, их классификация и группировки;
- спецификации номенклатурных позиций (структуры продуктов);
- технологические маршруты (в том числе учетные точки в нём для построения системы производственного учета);
- другие данные.

– Формирование рекомендаций по устранению выявленного дефицита данных об объектах нормативно-справочной информации в существующей информационной системе.

– Аудит процесса подготовки и ведения справочников нормативно-справочной информации на предмет соответствия задачам предприятия и принципам формирования ИС.

– Выделение категорий затрат, изучение и определение методик расчета себестоимости продукции (в части прямых затрат и переменных косвенных затрат).

– *Подготовка бизнес-процессов.*

– Анализ и формирование рекомендаций по совершенствованию бизнес-процессов планирования операционной деятельности, ее исполнения, а также ведения нормативных данных для поддержки операционной деятельности.

– Анализ и формирование рекомендаций по достижению соответствия бизнес-процессов рекомендациям методики ИС

– Разработка моделей бизнес-процессов сбыта, производства, закупок, планирования и других, в соответствии с предметной областью проекта, на различных

уровнях иерархии плановых решений, необходимых предприятию Заказчика бизнес-процессов, которые будут поддерживаться системой

– *Выбор программной системы автоматизации планирования и учета на производстве.*

– Анализ рынка программного обеспечения.

– Разработка системы аналитической отчетности, которую необходимо будет получать средствами системы.

– Разработка требований к информационной системе.

– Подготовка технического задания на выбор и внедрение информационной системы.

– Организация проведения конкурса по выбору программного обеспечения для информационной системы.

Необходимо учитывать уровень подготовки специалистов, которым предстоит работать с приложением, а также назначение приложения. Если пользователи имеют большой опыт работы с программными приложениями, то можно использовать многооконный интерфейс, выпадающие меню и т. д.

Если же речь идет о сотрудниках, для которых сложно “двумя руками три кнопки нажать”, то интерфейс системы должен быть как можно более простым, а последовательность действий - очевидной. Аналогично, если в режиме использования критичен быстрый ввод данных, то на первое место выходит удобство интерфейса. Имеет смысл еще до сдачи информационной системы в эксплуатацию предоставить разработчикам возможность попробовать себя в роли конечных пользователей.

План конспекта:

Требования к технологии создания информационных систем. Основные поддерживаемые процессы:

- управление требованиями
- анализ и проектирование ИС
- разработка ИС
- эксплуатация
- сопровождение
- документирование
- управление конфигурацией и изменениями
- тестирование
- управление проектом

Основные этапы процесса внедрения ТС ИС:

- определение потребностей в ТС ИС
- определение требований, предъявляемых к ТС ИС
- оценка вариантов ТС ИС
- выбор ТС ИС
- адаптация ТС ИС к условиям применения

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

Тема самостоятельного изучения № 5. Обзор инструментария для создания ИС

Вид деятельности студентов: подготовка к лекциям, самостоятельное изучение литературы, подготовка и выполнение лабораторных работ, выполнение контрольной

работы

Итоговый продукт самостоятельной работы: конспект
Средства и технологии оценки: отчет
Краткие теоретические сведения
Обзор технологий создания ИС

Под средствами проектирования информационных систем (СП ИС) будем понимать комплекс инструментальных средств, обеспечивающих в рамках выбранной методологии проектирования поддержку полного жизненного цикла (ЖЦ) ИС, который включает в себя:

- Планирование и анализ.
- Проектирование.
- Реализацию и внедрение.
- Эксплуатацию.

Понятие «цикл» связано с тем, что как правило, начало эксплуатации ИС совпадает с планированием и анализом обстановки для создания следующей ИС (результаты опытной эксплуатации являются исходными данными для обоснования и выбора параметров новой ИС).

Каждый этап характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. При анализе СП их следует рассматривать не локально, а в комплексе, что позволяет реально охарактеризовать их достоинства, недостатки и место в общем технологическом цикле создания ИС. В общем случае стратегия выбора СП для конкретного применения зависит от следующих факторов:

- характеристик моделируемой предметной области;
- целей, потребностей и ограничений будущего проекта ИС, включая квалификацию участвующих в процессе проектирования специалистов;
- используемой методологии проектирования.

Тенденции развития современных информационных технологий приводят к постоянному возрастанию сложности ИС, создаваемых в различных областях экономики. Современные сложные ИС и проекты, обеспечивающие их создание, характеризуются, как правило, следующими особенностями:

- сложность предметной области (достаточно большое количество функций, объектов, атрибутов и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;
- наличие совокупности тесно взаимодействующих компонентов - подсистем, имеющих свои локальные задачи и цели функционирования;
- иерархическую структуру взаимосвязей компонентов, обеспечивающую устойчивость функционирования системы;
- иерархическую совокупность критериев качества функционирования компонентов и ИС в целом, обеспечивающих достижение главной цели - создания и последующего применения системы;
- отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;
- необходимость достаточно длительного сосуществования старых приложений и вновь разрабатываемых БД и приложений;
- наличие потребности как в традиционных приложениях, связанных с обработкой транзакций и решением регламентных задач, так и в приложениях аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема;
- поддержка одновременной работы достаточно большого количества локальных сетей, связываемых в глобальную сеть масштаба предприятия, и территориально удаленных пользователей;

– функционирование в неоднородной операционной среде на нескольких вычислительных платформах;

– разобщенность и разнородность отдельных микроколлективов разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

– существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика и различной степенью готовности отдельных ее подразделений к внедрению ИС.

На выбор СП могут существенно повлиять следующие особенности методологии проектирования:

– ориентация на создание уникального или типового проекта;

– итерационный характер процесса проектирования;

– возможность декомпозиции проекта на составные части, разрабатываемые группами исполнителей ограниченной численности с последующей интеграцией составных частей;

– жесткая дисциплина проектирования и разработки при их коллективном характере;

– необходимость отчуждения проекта от разработчиков и его последующего централизованного сопровождения.

Критерии выбора средств проектирования

Традиционно при обсуждении проблемы выбора СП (в особенности CASE-средств) большое внимание уделялось особенностям реализации той или иной методологии анализа предметной области (IDEF0, IDEF1X, Гана/Сарсона, Джордана, Баркера и др.). Безусловно, богатство изобразительных и описательных средств дает возможность на этапах стратегического планирования и анализа построить наиболее полную и адекватную модель предметной области. С другой стороны, если говорить о конечных результатах - базах данных и приложениях, то обнаруживается, что часть описаний в них практически не отражается, оставаясь чисто декларативной (на выходе мы в любом случае получим описание БД в табличном представлении). Опытные аналитики и проектировщики всегда с большими или меньшими трудозатратами придут к нужному конечному результату независимо от того, какая конкретно методология или ее разновидность реализована в данном инструменте. Это, конечно, не означает, что методология не важна, напротив, отсутствие или неполнота описательных средств могут с самого начала значительно затруднить работу над проектом. Однако, зачастую на первом плане оказываются другие критерии, невыполнение которых может породить гораздо большие трудности.

Может создаться впечатление, что если можно сформировать необходимую аппаратную платформу из компонентов различных фирм-производителей, то так же просто можно выбрать и скомплексировать разные инструментальные средства, каждое из которых является одним из мировых лидеров в своем классе. Однако в случае инструментальных средств в настоящее время, в отличие от оборудования, отсутствуют международные стандарты на основные свойства конечных продуктов (программ, баз данных и их сопряжение). Поскольку составные части проекта должны быть интегрированы в единый продукт имеет смысл рассматривать только сопряженные инструментальные средства, которые в принципе могут быть ориентированы - даже внутри одного класса - на разные методологии; при этом необходимо отбирать в состав комплекса СП средства, поддерживающие по крайней мере близкие методологии, если не одну и ту же. Исходя из перечисленных выше соображений, примем в качестве основных свойств (показателей, критериев выбора) СП следующие критерии:

Поддержка полного ЖЦ ИС с обеспечением эволюционности ее развития.

Полный жизненный цикл ИС должен поддерживаться "сквозной" технологической цепочкой средств разработчика, обеспечивающей решение следующих задач:

– обследование и получения формализованных знаний о предметной области (последовательный и логически связный переход от формализованного описания предметной области к ее моделям);

– декомпозиция проекта на составные части и интеграция составных частей;

– проектирование моделей приложений (логики приложений и пользовательских интерфейсов);

– прототипирование приложений;

– проектирование баз данных;

– коллективная, территориально распределенная разработка приложений с использованием различных инструментальных средств (включая их интеграцию, тестирование и отладку);

– разработка распределенных баз данных (с выбором оптимальных вариантов распределения);

– разработка проектной документации с учетом требований проектных стандартов;

– адаптация к различным системно-техническим платформам и СУБД;

– тестирование и испытания;

– сопровождение, внесение изменений и управление версиями и конфигурацией ИС;

– интеграция с существующими разработками (включая реинжиниринг приложений, конвертирование БД);

– администрирование ИС (оптимизация эксплуатационных характеристик);

– управление разработкой и сопровождением ИС (планирование, координация и контроль за ресурсами и ходом выполнения работ);

– прогнозирование и оценка трудоемкости, сроков и стоимости разработки.

Для существующих ИС должен обеспечиваться плавный переход из старой среды эксплуатации в новую с минимальными переделками и поддержкой эксплуатируемых баз данных и приложений, внедренных до начала работ по созданию новой системы.

Обеспечение целостности проекта и контроля за его состоянием.

Данное требование означает наличие единой технологической среды создания, сопровождения и развития ИС, а также целостность базы проектных данных (репозитория). Единая технологическая среда должна обеспечиваться за счет использования единственной CASE-системы для поддержки моделей ИС, а также за счет наличия программно-технологических интерфейсов между отдельными инструментальными средствами, сертифицированных и поддерживаемых фирмами- разработчиками соответствующих средств. В частности, интерфейс между CASE-системой и средствами разработки приложений должен выполнять две основные функции:

– непосредственный переход в рамках единой среды от описания логики приложения, реализованного CASE-системой, к разработке пользовательского интерфейса (экранных форм);

– перенос описания БД из репозитория CASE-системы в репозиторий средства разработки приложений и обратно.

Вся информация о проекте должна автоматически помещаться в базу проектных данных, при этом должны поддерживаться согласованность, непротиворечивость, полнота и минимальная избыточность проекта, а также корректность операций его редактирования. Это может быть достигнуто при условии исключения или существенного ограничения возможности актуализации репозитория различными средствами. Должны также обеспечиваться возможности для централизованного сбора, хранения и распределения информации между различными этапами проекта, группами разработчиков и выполняемыми операциями. Поддержка базы проектных данных может быть реализована собственными средствами СП или средствами целевой СУБД (второй вариант предпочтительнее, поскольку упрощается технология ведения репозитория).

Невыполнение требования целостности в условиях разобщенности разработчиков и временной протяженности крупного проекта может означать утрату контроля за его состоянием.

Независимость от программно-аппаратной платформы и СУБД.

Требование определяется неоднородностью среды функционирования ИС. Такая независимость может иметь две составляющих: независимость среды разработки и независимость среды эксплуатации приложений. Она обеспечивается за счет наличия совместимых версий СП для различных платформ и драйверов соответствующих сетевых протоколов, менеджеров транзакций и СУБД. Один из дополнительных факторов, который при этом следует учитывать - это способ взаимодействия с СУБД (прямой или через ODBC), поскольку использование ODBC может заметно ухудшить производительность и надежность интерфейса.

Поддержка одновременной работы групп разработчиков.

Развитые СП должны обладать возможностями разделения полномочий персонала разработчиков и объединения отдельных работ в общий проект. Должна обеспечиваться одновременная работа проектировщиков БД и разработчиков приложений (разработчики приложений в такой ситуации могут начинать работу с базой данных, не дожидаясь полного завершения ее проектирования CASE-средствами). При этом все группы специалистов должны быть обеспечены адекватным инструментарием, а внесение изменений в проект различными разработчиками должно быть согласованным и корректным. Каждый разработчик должен иметь возможность работы со своим личным репозиторием, являющимся фрагментом или копией общего репозитория. Должны обеспечиваться содержательная интеграция всех изменений, вносимых разработчиками, в общем репозитории, одновременная доступность для разработчика общего и личного репозиториев и простота переноса объектов между ними. Помимо перечисленных основных критериев, предварительный анализ при выборе СП должен учитывать следующие аспекты:

Возможность разработки приложений "клиент-сервер"

Подразумевается сочетание наличия развитой графической среды разработки приложений (многооконность, разнообразие стандартных графических объектов, разнообразие используемых шрифтов и т.д.) с возможностью декомпозиции приложения на "клиентскую" часть, реализующую пользовательский экранный интерфейс и "серверную" часть. При этом должна обеспечиваться возможность перемещения отдельных компонентов приложения между "клиентом" и "сервером" на наиболее подходящую платформу, обеспечивающую максимальную эффективность функционирования приложения в целом, а также возможность использования сервера приложений (менеджера транзакций).

Открытая архитектура и возможности экспорта/импорта.

Открытая и общедоступная информация об используемых форматах данных и прикладных программных интерфейсах должна позволять интегрировать инструментальные средства третьих фирм и относительно безболезненно переходить от одной системы к другой. Возможности экспорта/импорта означают, что спецификации, полученные на этапах анализа, проектирования и реализации для одной ИС, могут быть использованы для проектирования другой ИС. Повторное проектирование и реализация могут быть обеспечены при помощи средств экспорта/импорта спецификаций в различные СП.

Качество технической поддержки, простота использования

Имеется в виду наличие квалифицированных дистрибьюторов и консультантов, быстрота обслуживания пользователей, высокое качество технической поддержки и обучения продукту и методологии его применения для больших коллективов разработчиков (наличие сведений о практике использования системы, качество документации, укомплектованность примерами и обучающими курсами, наличие прототипных проектов).

Затраты на обучение новым технологиям значительны, однако потери от использования современных сложных технологий необученными специалистами могут оказаться значительно выше. Кроме того, фирмы-поставщики инструментальных средств должны быть устойчивыми, так как технология выбирается не на один год, а также должны обеспечивать хорошую поддержку на территории России (горячая линия, консультации, обучение, консалтинг), возможно, через дистрибьюторов. Что касается стоимости, следует учитывать возможность получения бесплатной пробной, стоимость лицензии на одно рабочее место СП, скидки, предоставляемые фирмой в случае приобретения большого количества лицензий, необходимость приобретения версий для эксплуатации приложений и т.д. В то же время стоимость продукта должна рассматриваться не сама по себе, а с учетом ее соответствия возможностям продукта.

Учитываются следующие характеристики: доступность пользовательского интерфейса; время, необходимое для обучения; простота инсталяции; качество документации.

Обеспечение качества проектной документации.

Это требование относится к возможностям СП анализировать и проверять описания и документацию на полноту и непротиворечивость, а также на соответствие принятым в данной методологии стандартам и правилам (включая ГОСТ, ЕСПД). В результате анализа должна формироваться информация, указывающая на имеющиеся противоречия или неполноту в проектной документации. Должна быть также обеспечена возможность создавать новые формы документов, определяемые пользователями.

Для того, чтобы проект мог выполняться разными коллективами разработчиков, необходимо использование стандартных методов моделирования и стандартных нотаций, которые должны быть оформлены в виде нормативов до начала процесса проектирования. Несоблюдение данного требования ставит разработчиков в зависимость от фирмы-производителя данного средства, делает затруднительным формальный контроль корректности используемых нотаций и снижает возможности привлечения дополнительных коллективов разработчиков, поскольку число специалистов, знакомых с данным методом (нотацией) может быть ограниченным.

В идеальном случае окончательный выбор может быть произведен по результатам тестирования в соответствии с заданным планом, которое должно включать имитацию проектирования реальной БД и разработки приложений и состоять из следующих шагов:

- *установка и конфигурирование*(ясность и точность инструкций по установке, наличие подсказок в процессе установки, возможность установки по выбору и задания многопользовательской конфигурации);
- *разработка концептуальной схемы БД*(понятность и простота построения, модификации и документирования различных элементов диаграмм "сущность-связь", отображение ограничений ссылочной целостности и бизнес- правил, управление режимом отображения);
- *формирование отчета о концептуальной схеме*(список сущностей с определениями и атрибутами, включая указание ключей, список атрибутов, сгруппированных по сущностям, список связей между сущностями, возможность форматирования отчета, составления отчета по выделенной части схемы, передачи отчета, например, в другие приложения (текстовые процессоры));
- *разработка графической схемы БД для конкретной СУБД*с учетом специфичных для нее структур данных и ограничений (выбор целевой СУБД и реализация элементов схемы - ввод и модификация имен таблиц и столбцов, определение типов данных, доменов, индексов, значений по умолчанию и неопределенных значений, порядка индексирования, а также задание ограничений ссылочной целостности и дополнительных бизнес-правил, характеризующих предметную область, управление триггерами и хранимыми процедурами);
- *формирование отчета о схеме БД*(печать диаграммы схемы, списка таблиц с

соответствующими столбцами, первичными ключами, индексами и т.д., возможность форматирования отчета, составления отчета по выделенной части схемы, передачи отчета в другие приложения);

– *генерация схемы БД*(трансформация схемы БД в файл DDL в текстовом формате или непосредственный интерфейс с целевой СУБД);

– *разработка простейшего приложения*(описание экранных форм, программирование или описание логики приложения и интерфейса с БД, загрузка БД тестовыми данными и тестирование приложения);

– *сопровождение схем БД*(внесение изменений - создание новых сущностей и атрибутов, изменение схемы БД, повторная генерация схемы, управление версиями, обеспечение сохранности данных, синхронизация концептуальной схемы и самой БД);

– *обратное проектирование* - *реинжиниринг*(полное и точное восстановление исходной концептуальной схемы по файлам DDL или непосредственно из словаря целевой СУБД).

В результате выполненного анализа может оказаться, что ни одно доступное средство не удовлетворяет в нужной мере всем основным критериям и не покрывает все потребности проекта. В этом случае может применяться набор средств, позволяющий построить на их базе единую технологическую среду.

Анализ средств проектирования информационных систем

Современные СП могут быть разделены на две большие категории.

Первую составляют CASE-системы (как независимые (upper CASE), так и интегрированные с СУБД), обеспечивающие проектирование БД и приложений в комплексе с интегрированными средствами разработки приложений "клиент-сервер" (например, Westmount I-CASE+Uniface, Designer/2000+Developer/2000). Их основное достоинство заключается в том, что они позволяют разрабатывать всю ИС целиком (функциональные спецификации, логику процессов, интерфейс с пользователем и базу данных), оставаясь в одной технологической среде. Инструменты этой категории, как правило, обладают существенной сложностью, широкой сферой применения и высокой гибкостью.

Вторую категорию составляют собственно средства проектирования БД, реализующие ту или иную методологию, как правило, "сущность-связь" и рассматриваемые в комплексе со средствами разработки приложений. К средствам этой категории можно отнести такие, как SILVERRUN+JAM, ERwin/ERX + PowerBuilder и др.

Помимо указанных, СП можно классифицировать по следующим признакам:

– степени интегрированности: (отдельные локальные средства, набор частично интегрированных средств, охватывающих большинство этапов жизненного цикла ИС и полностью интегрированные средства, связанные общей базой проектных данных - репозиторием);

– применяемым методологиям и моделям систем и БД;

– степени интегрированности с СУБД;

– степени открытости;

– доступным платформам.

В разряд СП попадают как относительно дешевые системы для персональных компьютеров (ПК) с весьма ограниченными возможностями, так и дорогостоящие системы для неоднородных вычислительных платформ и операционных сред. Так, современный рынок программных средств насчитывает около 300 различных CASE-систем, наиболее мощные из которых так или иначе используются практически всеми ведущими западными фирмами. Применение СП требует от потенциальных пользователей специальной подготовки и обучения. Опыт показывает, что внедрение СП осуществляется медленно, однако по мере приобретения практических навыков и общей культуры проектирования эффективность применения этих средств резко возрастает, причем наибольшая потребность в использовании СП испытывается на начальных этапах разработки, а именно на этапах

анализа и спецификации требований. Это объясняется тем, что цена ошибок, допущенных на начальных этапах, на несколько порядков превышает цену ошибок, выявленных на более поздних этапах разработки. На сегодняшний день Российский рынок программного обеспечения располагает следующими наиболее развитыми СП:

• Westmount I-CASE;	• Uniface;
• SILVERRUN+JAM;	• ERwin/ERX+PowerBuilder.
• Designer/2000+Developer/2000(ORACLE	

Приведенный список не претендует на полноту. Кроме того, на рынке постоянно появляются как новые (для отечественных пользователей) системы, так и новые версии и модификации перечисленных систем (например, CASE/4/0, System Architect и т.д.). Некоторое представление о возможностях наиболее развитых СП может дать краткая характеристика следующих программных продуктов:

Westmount I-CASE 3.2 (CADRE Technologies Inc.)

Westmount I-CASE представляет собой интегрированный программный продукт, обеспечивающий выполнение следующих функций:

– графическое проектирование архитектуры системы (проектирование состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирование отношений типа "клиент- сервер", анализ использования мониторов транзакций и особенностей функционирования систем в реальном времени);

– проектирование диаграмм потоков данных, "сущность-связь", структур данных, структурных схем программ и последовательностей экранных форм;

– генерация кода программ на 4GL целевой СУБД с полным обеспечением программной среды и генерация SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;

– программирование на языке С со встроенным SQL;

– управление версиями и конфигурацией проекта;

– генерация проектной документации по стандартным и индивидуальным шаблонам;

– экспорт и импорт данных проекта в формате CDIF.

Westmount I-CASE можно использовать в конфигурации "клиент-сервер", при этом база проектных данных может располагаться на сервере, а рабочие места разработчиков могут быть клиентами. Westmount I-CASE функционирует на всех основных UNIX-платформах и VMS. В качестве целевой СУБД могут использоваться ORACLE, Informix, Sybase и Ingres. В качестве отдельного продукта поставляется интерфейс Westmount-Uniface Bridge, обеспечивающий совместное использование двух систем в рамках единой технологической среды проектирования (при этом схемы БД, структурные схемы программ и последовательности экранных форм непосредственно в режиме on-line, без создания каких-либо файлов экспорта- импорта, переносятся в репозиторий Uniface, и, наоборот, прикладные модели, сформированные средствами Uniface, могут быть перенесены в репозиторий Westmount I-CASE. Возможные рассогласования между репозиториями двух систем устраняются с помощью специальной утилиты). В рамках версии Westmount I-CASE 4.0 предполагается обеспечить возможность функционирования клиентской части в среде Windows 95, а серверной - в среде Windows NT.

Uniface (Compuware)

Uniface 6.1 представляет собой среду разработки крупномасштабных приложений "клиент-сервер" и имеет следующую компонентную архитектуру:

– *Application Objects Repository*(репозиторий объектов приложений) содержит метаданные, автоматически используемые всеми остальными компонентами на протяжении жизненного цикла ИС.

– *Application Model Manager*поддерживает прикладные модели, каждая из

которых представляет собой подмножество общей схемы БД с точки зрения данного приложения.

– *Rapid Application Builder*- средство быстрого создания экранных форм и отчетов на базе объектов прикладной модели. Оно включает графический редактор форм, средства прототипирования, отладки, тестирования и документирования. Реализован интерфейс с разнообразными типами оконных элементов управления (Open Widget Interface) для существующих графических систем - MS Windows (включая VBX), Motif, OS/2.

– *Developer Services*(службы разработчика) - используются для поддержки крупных проектов и реализуют контроль версий, права доступа, глобальные модификации и т.д. Это обеспечивает разработчиков средствами параллельного проекти-рования, входного и выходного контроля, поиска, просмотра, поддержки и выдачи отчетов по данным системы контроля версий.

– *Deployment Manager*(управление распространением приложений) - средства, позволяющие подготовить созданное приложение для распространения, установить и сопровождать его (при этом платформа пользователя может отличаться от платформы разработчика). В их состав входят сетевые драйверы и драйверы СУБД, сервер приложений (полисервер), средства распространения приложений и управления базами данных. Uniface поддерживает интерфейс практически со всеми известными программно- аппаратными платформами, СУБД, CASE-средствами, сетевыми протоколами и менеджерами транзакций.

– *Personal Series*(персональные средства) - используются для создания сложных запросов и отчетов в графической форме, а также для переноса данных в такие системы, как WinWord и Excel.

Анализ данных показывает, что из перечисленных СП только комплекс Westmount I-CASE+Uniface наиболее полно удовлетворяет всем критериям, принятым в качестве основных. Так, например, в комплексе Westmount I-CASE+Uniface целостность базы проектных данных и единая технология сквозного проектирования ИС обеспечивается за счет использования интерфейса Westmount-Uniface Bridge. Следует отметить, что каждый из двух продуктов сам по себе является одним из наиболее мощных в своем классе. Таким образом, наиболее развитыми средствами разработки крупномасштабных ИС на сегодняшний день является, по мнению автора, комплекс Westmount I-CASE+Uniface. С другой стороны, его применение не исключает использования в том же самом проекте таких средств, как PowerBuilder, для разработки сравнительно небольших прикладных систем в среде MS Windows.

CASE-технологии в создании информационных систем

Значительно лучше соответствуют большой размерности задачи иерархические CASE-модели. Аббревиатура CASE (Computer-Aided Software/System Engineering) означает проектирование программного обеспечения или системы на основе компьютерной поддержки.

CASE-технология — актуальное и интенсивно развивающееся направление создания САПР в области программных продуктов и систем обработки информации. Практически ни один крупный зарубежный программный продукт не создается в настоящее время без использования CASE-средств.

Среди отечественных систем, созданных с использованием CASE-средств, следует отметить систему БОСС-КОРПОРАЦИЯ фирмы АйТи. На всех стадиях создания этой системы использовались средства разработки, относящиеся к семейству Oracle 2000 (Designer/2000, Developer/200, Programmer/2000).

Область применения CASE-технологий относится к созданию, прежде всего, экономических информационных систем, что объясняется массовостью этих систем.

Следует отметить, что CASE-технологий применяются не только для создания автоматизированных систем управления, но и для разработки моделей систем, помогающих

в принятии решений в области стратегического планирования, управления финансами фирмы, обучения персонала и т.д. Это направление применения CASE-технологий получило свое собственное название — бизнес-анализ.

CASE-технологий применяются также там, где проблематика предметной области отличается большой сложностью, например, в разработке системного программного обеспечения.

Рассмотрим методологические основы CASE-технологий.

Основой CASE-методологии является моделирование. CASE-технология — это модельный метод автоматизации проектирования системы.

CASE-технология основана на парадигме: методология — метод — нотации — средства.

Методология определяет общие подходы к оценке и выбору варианта системы, последовательность стадий и этапов проектирования, подходы к выбору методов.

Метод конкретизирует порядок проектирования отдельных компонентов системы (например, известны методы проектирования потоков данных в системе, задания спецификаций (описаний) процессов, представления структур данных в хранилище и т.д.).

Нотации — это графические средства обозначения и правила, предназначенные для описания структуры системы, этапов обработки информации, структуры данных и т. д. Нотации включают графы, диаграммы, таблицы, блок-схемы, формальные и естественные языки.

Наконец, средства — это инструментарии, средства автоматизации проектирования в виде программных продуктов для обеспечения интерактивного режима проектирования (создание и редактирование графического проекта информационной системы) и кодогенерацни программ (автоматического создания кодов программ системы).

Методология проектирования на основе компьютерной поддержки, очевидно, требует построения формализованного описания информационной системы в виде информационной модели. Построение CASE-модели системы предусматривает декомпозицию системы и иерархическое упорядочивание декомпозированных подсистем.

Модель системы должна отражать:

- функциональную часть системы;
- отношения между данными;
- переходы состояний системы при работе в реальном времени. Для моделирования информационной системы в трех указанных аспектах используются три разновидности графических средств с определенными нотациями.

1. Диаграммы потоков данных — DFD (Data Flow Diagrams). Они используются совместно со словарями данных и спецификациями процессов.

2. Диаграммы „сущность-связь" — ERD (Entity Relationship Diagrams), показывающие отношения между данными.

3. Диаграммы переходов состояний — STD (State Transitign Diagrams) для отражения зависящего от времени поведения системы (в режиме реального времени).

Ведущая роль в моделировании принадлежит DFD.

DFD предназначена для отражения взаимосвязей источников и приемников данных (так называемых внешних сущностей по отношению к информационной системе), потоков данных, процессов обработки (вычислительных процессов, соответствующих функциям системы), хранилищ данных (накопителей).

Графическое представление диаграммы потоков данных на экране дисплея обеспечивает наглядность моделирования и удобство корректировки основных компонентов модели в интерактивном режиме.

Поскольку графического представления недостаточно для точного определения компонентов DFD, используются текстовые описания и другие средства конкретизации процессов обработки и структуры данных.

Так, потоки данных конкретизируются в части их структуры в словарях данных.

Каждый процесс (функция системы) может быть детализирована с помощью DFD нижнего уровня, где он разделяется на несколько процессов с одновременной детализацией потоков данных.

Детализация процессов заканчивается, когда описание каждого детализированного процесса может быть сделано с помощью выбранного метода написания алгоритма процесса. Спецификация процесса содержит номер и имя процесса, списки имен входных и выходных данных из словаря данных и алгоритм процесса, трансформирующий входные потоки данных во входные. В CASE-технологии используются такие методы задания алгоритмов процессов, как:

- текстовое описание;
- естественный структурированный язык;
- таблицы решений;
- деревья решений;
- визуальные языки;
- языки программирования.

Языки программирования (C, Cobol и др.) вызывают затруднения в написании алгоритмов применительно к DFD, поскольку требуют использования, помимо потоков данных, словарей данных, и требуют синхронной корректировки спецификаций процессов при корректировке DFD.

Структурированный естественный язык легко понимается не только проектировщиками и программистами, но и конечными пользователями. В этом его достоинство. Однако он не обеспечивает автоматической кодогенерации из-за наличия неоднозначностей.

Таблицы и деревья решений, наглядно отражая связь комбинации условий с необходимыми действиями, не обладают процедурными возможностями для кодогенерации программ.

Визуальные языки обеспечивают автоматическую кодогенерацию, но представленные с их помощью спецификации процессов сложно корректировать.

Содержимое каждого хранилища данных, представленного на диаграмме потока данных, описывается словарем данных и моделью данных ERD. В случае работы системы в реальном времени DFD дополняется STD.

Важным методологическим принципом CASE-технологии создания информационной системы является четкое разделение процесса создания системы на 4 стадии:

- предпроектную (стадию анализа, прототипирования, и построения модели требовании к системе);
- проектную, предполагающую логическое проектирование системы (без программирования);
- стадию программирования (включая проектирование физической базы данных);
- послепроектную, включающую в себя ввод в действие, эксплуатацию и сопровождение системы.

На предпроектной стадии строится модель требований к системе, т. е. подробное описание того, что она должна делать, без указания путей реализации требований.

На проектной стадии происходит уточнение модели требований (разработка подробной иерархической модели на основе DFD и спецификаций процессов) и расширение ее до модели реализации на логическом уровне. В заключение этой стадии происходит тщательный контроль проекта на уровне логической модели реализации.

На следующей стадии (программирования) осуществляется физическое проектирование системы. Эта стадия предусматривает автоматическую кодогенерацию по спецификациям процессов программного обеспечения системы и физическое

проектирование базы данных.

Заключительная послепроектная стадия начинается с приемосдаточных испытаний. Далее следуют ввод в постоянную эксплуатацию, сопровождение и развитие системы.

Рассмотрим факторы эффективности CASE-технологии.

1. Следует отметить, что CASE-технология создает возможность и предусматривает перенос центра тяжести в трудоемкости создания системы на предпроектную и проектную стадии. Тщательная проработка этих стадий в интерактивном режиме с компьютерной поддержкой уменьшает число возможных ошибок в проектировании, исправлять которые на последующих стадиях затруднительно.

2. Доступная для понимания пользователей-непрограммистов графическая форма представления модели позволяет осуществить принцип пользовательского проектирования, предусматривающий участие пользователей в создании системы. CASE-модель позволяет достичь взаимопонимания между всеми участниками создания системы (заказчиками, пользователями, проектировщиками, программистами).

3. Наличие формализованной модели системы на предпроектной стадии создает возможность для многовариантного анализа с прототипированием и ориентировочной оценкой эффективности вариантов. Анализ прототипа системы позволяет скорректировать будущую систему до того, как она будет реализована физически. Этот подход ускоряет и удешевляет создание системы.

4. Закрепление в формализированном виде требований к системе избавляет проектировщиков от необходимости многочисленных корректировок по новым требованиям пользователей.

5. Отделение проектирования системы от программирования создает устойчивость проектных решений для реализации на разных программно-технических платформах.

6. Наличие формализованной модели реализации системы и соответствующих средств автоматизации позволяет осуществить автоматическую кодогенерацию программного обеспечения системы и создать рациональную структуру базы данных.

7. На стадии эксплуатации системы появляется возможность внесения изменений на уровне модели, не обращаясь к текстам программ, возможно, силами специалистов отдела автоматизации фирмы.

8. Модель системы может использоваться не только как основа ее создания, но и в целях автоматизированного обучения персонала с использованием диаграмм.

9. На основе модели действующей системы может выполняться бизнес-анализ для поддержки управленческих решений и бизнес-реинжиниринг при изменении направления деятельности фирмы.

Рассмотрим программные средства, обеспечивающие CASE-технологию. В зависимости от функционального назначения они подразделяются на следующие классификационные группировки, обеспечивающие:

- анализ и проектирование информационной системы;
- проектирование баз данных;
- программирование;
- сопровождение и реинжиниринг;
- управление процессом проектирования.

Средства анализа и проектирования служат для построения CASE-модели как действующей, так и реализуемой системы управления. Они поддерживают графическое построение и контроль иерархической модели диаграмм потоков данных и описание ее компонентов. Эти средства позволяют аналитикам и проектировщикам получить доступ к базе данных проектируемой системы (репозитарию).

К таким средствам относятся: отечественный пакет CASE. Аналитик, Design/IDEF (Meta Software), The Developer (ASYST Technologies) и др.

Для согласования требований пользователей создаются прототипы пользовательских интерфейсов, включающих в себя меню, экранные формы и отчеты в

виде таблиц или графиков. Примером программного средства создания пользовательского интерфейса является Developer/2000 (Oracle).

Средства проектирования баз данных обеспечивают логическое моделирование данных, автоматическое преобразование моделей данных в третью нормальную форму и генерацию схем баз данных. Примерами таких средств является Designer/2000 фирмы Oracle, ERWin (Logic Works) и др.

Средства программирования поддерживают автоматическую кодогенерацию из спецификаций процессов, тестирование и документирование программы. К их числу относятся Programmer/2000 (Oracle), DECASE (DEC), APS (Sage Software) и др.

Средства сопровождения и реижиниринга позволяют вносить изменения в систему на уровне моделей при меняющихся условиях бизнеса (Adpac CASE Tools фирмы Adpac и др.).

Средства управления процессом проектирования поддерживают планирование и контроль выполнения комплекса проектных работ, а также взаимодействие аналитиков, проектировщиков и программистов на основе общей базы данных проекта (например, Project Workbench фирмы Applied Business Technology). Очевидна актуальность создания интегрированного пакета инструментальных средств поддержки CASE-технологии на всех этапах жизненного цикла информационной системы.

План конспекта:

Основные технологии создания ИС. Оценка и выбор технологии создания информационных систем:

- требования к ТС ИС
- цели и ограничения проекта
- данные о доступных технологиях

Способы оценки и накопление информации о технологиях:

- анализ технологий и документации поставщика
- опрос реальных пользователей
- анализ результатов проектов, использовавших данные технологии
- просмотр демонстраций и опрос демонстраторов
- выполнение тестовых примеров
- применение технологий в пилотных проектах
- анализ любых доступных результатов предыдущих оценок

Работа с литературой:

Рекомендуемые источники информации (№ источника)			
Основная	Дополнительная	Методическая	Интернет-ресурсы
1-2	1	1-3	1-2

4. КРИТЕРИИ ОЦЕНИВАНИЯ КОМПЕТЕНЦИЙ

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

5. УЧЕБНО-МЕТОДИЧЕСКОЕ И ИНФОРМАЦИОННОЕ ОБЕСПЕЧЕНИЕ ДИСЦИПЛИНЫ

5.1. Рекомендуемая литература

5.1.1. Основная литература

3. Избачков Ю.С., Петров В.Н., Васильев А.А., Телина И. С. Информационные системы: Учебник для ВУЗов. 3-е изд. – СПб.: Питер, 2011
4. Информационные системы в экономике / Под ред. Д. Дика – М. 2010.

5.1.2. Дополнительная литература

2. Благодатских В.А. Стандартизация разработки программных средств: учеб.пособие / В.А. Благодатских, В.А. Волнин, К.Ф. Поскакалов; под ред. Проф. О.С.Разумова, 2010 г. - 288 с.

5.1.3. Методическая литература

1. Методические указания по выполнению лабораторных работ по дисциплине «Технология создания информационных систем».
2. Методические указания по выполнению контрольной работы по дисциплине «Технология создания информационных систем».
3. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Технология создания информационных систем».

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Пятигорский институт (филиал) СКФУ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ ПО ДИСЦИПЛИНЕ ТЕХНОЛОГИИ СОЗДАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Направление подготовки	**09.04.02 Информационные системы и технологии**
Направленность (профиль)	**«Технологии работы с данными и знаниями, анализ информации»**
Квалификация выпускника	Магистр

Пятигорск, 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Методические указания содержат перечень вариантов заданий для контрольных работ, требования к оформлению контрольных работ и пример выполнения задания. Теоретической основой подготовки специалиста являются знания в области информатики, вычислительной систем.

1. ЦЕЛЬ, ЗАДАЧИ И РЕАЛИЗУЕМЫЕ КОМПЕТЕНЦИИ

Методические указания составлены с учетом требований стандарта высшего образования по дисциплине: «Технология создания информационных систем». Целью освоение дисциплины «Технология создания информационных систем» является формирование набора общекультурных и профессиональных компетенций будущего магистра по направлению подготовки 09.04.02 «Информационные системы и технологии», для решения прикладных задач в рамках направленности (профиля) «Технологии работы с данными и знаниями, анализ информации».

Задачами курса является изучение знаний, освоение и умение применять на практике методы и способы создания информационных систем и технологий, умение профессионально использовать современное оборудование, освоение современных инструментальных средств для обработки данных и анализа информации.

Индекс	Формулировка:
ПК-1	способность осуществлять управление, развитием баз данных, включая развертывание, сопровождение, оптимизацию функционирования баз данных, являющихся частью различных информационных систем
ПК-6	способность проводить организационное сопровождение разработки, отладки, модификации и поддержки информационных технологий и систем
ПК-7	способность проводить непосредственное руководство процессами разработки программного обеспечения, организация процессов разработки программного обеспечения, управление программно-техническими, технологически ми и человеческими ресурсами
ПК-8	способность к эффективному управлению работы персоналом, к повышению профессионализма персонала, к организации эффективного взаимодействия
ПК-9	способность выполнять управление проектами в области ИТ любого масштаба в условиях высокой неопределенности вызываемой запросами на изменения и рисками, и с учетом влияния организационного окружения проекта
ПК-12	способность адаптировать типовые проекты информационных систем под конкретные объект, с целью проведения анализа информации
ПК-13	способность проводить разработку и исследование теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях и сферах цифровой экономики

2. ФОРМУЛИРОВКА ЗАДАНИЯ И ЕГО ОБЪЕМ

Контрольная работа включат в себе вопросы по темам, которые нужно изучить самостоятельно и по ним подготовить отчет и презентации. Результаты выполнения контрольной работы предоставляются в электронном виде. Объем контрольной работы

составляет 10-15 печатных листов формата А4.

Варианты задании выбираются из таблицы по последним двум цифрам зачетной книжки.

Послед цифра	Предпоследняя цифра									
0	,11	,12	,13	,14	,15	,16	,17	,18	,19	0,21
1	1,22	2,23	3,24	4,25	5,26	6,27	7,27	8,28	9,29	0,30
2	,30	,29	,28	,27	,26	,25	,24	,23	,22	0,21
3	1,29	2,28	3,27	4,26	5,25	6,24	7,23	8,22	9, 21	0,10
4	9,31	8, 32	7, 33	6,34	5,35	4,18	3,19	2,20	1,21	0,22
5	,35	,34	,33	,32	,31	,30	,29	,28	,27	0,26
6	0,11	9,10	8,9	7,8	6,7	5,6	4,5	3,4	2,3	1, 2
7	0,31	9, 30	8,29	7,28	6,27	5,26	4,25	3,24	2,23	1,22
8	0,29	,31	,33	,35	,25	,23	,21	,19	,17	,15
9	, 20	,26	,24	,23	,22	,20	,18	1,17	,30	, 31

3. ОБЩИЕ ТРЕБОВАНИЯ К НАПИСАНИЮ И ОФОРМЛЕНИЮ РАБОТЫ

Контрольная работа выполняется и сдается в электронном виде на CD/CDRW носителе. На конверте необходимо указать название дисциплины, ФИО студента, факультет, номер группы, шифр зачетной книжки, № варианта задания, и список всех созданных в ходе выполнения задания файлов.

Приведенный в конце методических указаний список литературы может использоваться студентами при выполнении контрольной работы.

4. Варианты заданий для студентов заочной формы обучения

1. Назначение, область применения и основные свойства информационных систем.
2. Методология создания информационных систем.
3. Основные задачи, решение которых должна обеспечивать методология создания информационных систем.
4. Современные методы создания ИС.
5. Объектно-ориентированная технология создания ИС.
6. Технология создания ИС, основанная на знаниях.
7. CASE-технология создания ИС.
8. Основополагающие принципы создания ИС.
9. Принцип современности.
10. Принцип стандартизации (унификации).
11. Принцип системности.
12. Принцип развития (открытости).
13. Принцип эффективности.

14. Этап формирования требований к ИС.
15. Разработка концепции.
16. Разработка и утверждение технического задания на разработку ИС.
17. Разработка эскизного проекта.
18. Разработка технического проекта.
19. Предпроектный этап создания ИС.
20. Проектный этап создания ИС.
21. Внедрение ИС.
22. Анализ функционирования ИС.
23. Механизм обеспечения совместной работы над проектом.
24. Требования к конфигурации рабочих мест разработчиков.
25. Правила подготовки, рассмотрения, согласования и утверждения документации.
26. Требования к стандарту проектирования.
27. Требования к стандарту оформления проектной документации.
28. Требования к стандарту интерфейса пользователя.
29. Требования к технологии создания информационных систем.
30. Основные поддерживаемые процессы.
31. Управление требованиями.
32. Управление конфигурацией и изменениями.
33. Определение потребностей в ТС ИС.
34. Определение требований, предъявляемых к ТС ИС.
35. Оценка вариантов ТС ИС.
36. Адаптация ТС ИС к условиям применения.
37. Основные технологии создания ИС.
38. Оценка и выбор технологии создания информационных систем.
39. Способы оценки и накопление информации о технологиях.
40. Цели и ограничения проекта.
41. Анализ технологий и документации поставщика.

5. План-график выполнения задания

Дата получения задания	Дата предоставления выполненного задания
Установочная сессия.	летняя сессия за две недели до начала сессии.

6. Критерии оценивания работы

Оценка «отлично» выставляется студенту, если он продемонстрировал глубокие, исчерпывающие знания и творческие способности в понимании, изложении и использовании учебно-программного материала; логически последовательные, содержательные, полные, правильные и конкретные ответы на все поставленные вопросы и дополнительные вопросы преподавателя; свободное владение основной и дополнительной литературой, рекомендованной учебной программой.

Оценка «хорошо» выставляется студенту, если он продемонстрировал твердые и достаточно полные знания всего программного материала, правильное понимание сущности и взаимосвязи рассматриваемых процессов и явлений; последовательные, правильные, конкретные ответы на поставленные вопросы при свободном устранении замечаний по отдельным вопросам; достаточное владение литературой, рекомендованной учебной программой.

Оценка «удовлетворительно» выставляется студенту, если он продемонстрировал твердые знания и понимание основного программного материала; правильные, без грубых

ошибок ответы на поставленные вопросы при устранении неточностей и несущественных ошибок в освещении отдельных положений при наводящих вопросах преподавателя; недостаточное владение литературой, рекомендованной учебной программой.

Оценка «неудовлетворительно» выставляется студенту, если он продемонстрировал неправильные ответы на основные вопросы, допущены грубые ошибки в ответах, непонимание сущности излагаемых вопросов; неуверенные и неточные ответы на дополнительные вопросы.

7. Порядок защиты работы

Защита контрольной работы проводится в виде научного дискурса с презентацией выполненных заданий, в соответствии с графиком защиты. После доклада студенту задаются вопросы как преподавателем, так и студентами группы.

В процессе защиты своей работы студент делает доклад продолжительностью 7-10 минут. Доклад должен быть предварительно подготовлен студентом. Лучшее впечатление производит доклад, в форме пересказа, без зачтения текста, которым следует пользоваться только для уточнения цифрового материала. Студент должен свободно ориентироваться в своей работе.

В выступлении необходимо корректно использовать демонстрационные материалы, которые усиливают доказательность выводов и облегчают восприятие доклада студента. Они оформляются в виде презентации в системе Power Point.

8. Учебно-методическое и информационное обеспечение дисциплины

Рекомендуемая литература

Основная литература

5. Избачков Ю.С., Петров В.Н., Васильев А.А., Телина И. С. Информационные системы: Учебник для ВУЗов. 3-е изд. – СПб.: Питер, 2011

6. Информационные системы в экономике / Под ред. Д. Дика – М. 2010.

1Дополнительная литература

1. Благодатских В.А. Стандартизация разработки программных средств: учеб.пособие / В.А. Благодатских, В.А. Волнин, К.Ф. Поскакалов; под ред. Проф. О.С.Разумова, 2010 г. - 288 с.

Методическая литература

4. Методические указания по выполнению лабораторных работ по дисциплине «Технология создания информационных систем»;

5. Методические указания по выполнению контрольной работы по дисциплине «Технология создания информационных систем»;

6. Методические рекомендации для студентов по организации самостоятельной работы по дисциплине «Технология создания информационных систем».