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

ПМ.05 ПРОЕКТИРОВАНИЕ И РАЗРАБОТКА ИНФОРМАЦИОННЫХ СИСТЕМ

ПМ.05.01 ПРОЕКТИРОВАНИЕ И ДИЗАЙН ИНФОРМАЦИОННЫХ СИСТЕМ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ

Специальности СПО

09.02.07 Информационные системы и программирование

Пятигорск 2023 г.

Методические указания к выполнению лабораторных работ по дисциплине ПМ.05.01 Проектирование и дизайн информационных систем составлены в соответствии с требованиями ФГОС СПО. Предназначены для студентов, обучающихся по специальности 09.02.07 Информационные системы и программирование.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Дисциплина «Проектирование и дизайн информационных систем» предусматривает изучение проектирования и создания информационных систем.

При изучении предмета следует соблюдать единство терминологии и обозначения в соответствии с действующими стандартами, Международной системной единицы (СИ).

В результате освоения учебной дисциплины обучающийся должен:

знать:

- основные виды и процедуры обработки информации, модели и методы решения задач обработки информации;

- основные платформы для создания, исполнения и управления информационной системой;

- основные процессы управления проектом разработки;

- основные модели построения информационных систем, их структуру, особенности и области применения;

- методы и средства проектирования, разработки и тестирования информационных систем;

- систему стандартизации, сертификации и систему обеспечения качества продукции.

уметь:

- осуществлять постановку задач по обработке информации;

- проводить анализ предметной области;

- осуществлять выбор модели и средства построения информационной системы и программных средств;

- использовать алгоритмы обработки информации для различных приложений;

- решать прикладные вопросы программирования и языка сценариев для создания программ;

- разрабатывать графический интерфейс приложения;

- создавать и управлять проектом по разработке приложения;

- проектировать и разрабатывать систему по заданным требованиям и спецификациям.

иметь практический опыт в:

- управлении процессом разработки приложений с использованием инструментальных средств;

- обеспечении сбора данных для анализа использования и функционирования информационной системы;

- программировании в соответствии с требованиями технического задания;

- использовании критериев оценки качества и надежности функционирования информационной системы;

- применении методики тестирования разрабатываемых приложений;

-определении состава оборудования и программных средств разработки информационной системы;

- разработке документации по эксплуатации информационной системы;

- проведении оценки качества и экономической эффективности информационной системы в рамках своей компетенции;

- модификации отдельных модулей информационной системы.

В результате освоения учебной дисциплины студент должен овладевать:

Общими компетенциями:

ОК 01. Выбирать способы решения задач профессиональной деятельности, применительно к различным контекстам.

ОК 02. Осуществлять поиск, анализ и интерпретацию информации, необходимой для выполнения задач профессиональной деятельности.

ОК 03. Планировать и реализовывать собственное профессиональное и личностное развитие.

ОК 04. Работать в коллективе и команде, эффективно взаимодействовать с коллегами, руководством, клиентами.

ОК 05. Осуществлять устную и письменную коммуникацию на государственном языке с учетом особенностей социального и культурного контекста.

ОК 09. Использовать информационные технологии в профессиональной деятельности.

ОК 10. Пользоваться профессиональной документацией на государственном и иностранном языке.

Профессиональными компетенциями:

ПК 5.1. Собирать исходные данные для разработки проектной документации на информационную систему.

ПК 5.2. Разрабатывать проектную документацию на разработку информационной системы в соответствии с требованиями заказчика.

ПК 5.3. Разрабатывать подсистемы безопасности информационной системы в соответствии с техническим заданием.

ПК 5.4. Производить разработку модулей информационной системы в соответствии с техническим заданием.

ПК 5.5. Осуществлять тестирование информационной системы на этапе опытной эксплуатации с фиксацией выявленных ошибок кодирования в разрабатываемых модулях информационной системы.

ПК 5.6. Разрабатывать техническую документацию на эксплуатацию информационной системы.

ПК 5.7. Производить оценку информационной системы для выявления возможности ее модернизации.

ОБЩИЕ МЕТОДИЧЕСКИЕ УКАЗАНИЯ
К ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ

Варианты лабораторных работ по дисциплине «Проектирование и дизайн информационных систем» для каждого обучающегося – индивидуальные.

Задачи и ответы на вопросы, выполненные не по своему варианту, не засчитываются.

Лабораторная работа выполняется в отдельной тетради. Условия задачи и формулировки вопросов переписываются полностью. Формулы, расчеты, ответы на вопросы пишутся ручкой, а чертежи, схемы и рисунки выполняются карандашом, на графиках и диаграммах указывается масштаб. Вначале задача решается в общем виде, затем делаются расчёты по условию задания. Решение задач обязательно ведется в Международной системе единиц (СИ).

При выполнении лабораторной работы необходимо следовать методическим указаниям: повторить краткое содержание теории, запомнить основные формулы и законы, проанализировать пример выполнения аналогичного задания, затем преступить непосредственно к решению задачи. К зачету допускаются студенты, получившие положительные оценки по всем лабораторным работам.

Правила выполнения лабораторных работ.

1. Студент должен прийти на лабораторную работу подготовленным к выполнению лабораторной работы.
2. Каждый студент после проведения работы должен представить отчет о проделанной работе с анализом полученных результатов и выводом по работе.
3. Таблицы и рисунки следует выполнять с помощью чертежных инструментов (линейки, циркуля, и т.д.) карандашом с соблюдением ЕСКД.
4. Расчет следует проводить с точностью до двух значащих цифр.
5. Исправления проводить на обратной стороне листа. При мелких исправлениях неправильное слово (буква, число и т.п.) аккуратно зачеркивается и над ним пишут правильное пропущенное слово (букву, число и т.п.).
6. Вспомогательные расчеты можно выполнять на отдельных листах, а при необходимости на листах отчета.
7. Если студент не выполнит лабораторную работу или часть работы, то он выполнит ее во внеурочное время, согласованное с преподавателем.
8. Оценку по лабораторной работе студент получает с учетом срока выполнения работы, если;

- расчеты выполнены правильно и в полном объеме;

1. сделан анализ проделанной работы и вывод по результатам работы;
2. студент может пояснить выполнение любого этапа работы;
3. отчет выполнен в соответствии с требованиями к выполнению работы.

Лабораторная работа №1. Поиск информации для разработки ИС.

Цель: Научиться осуществлять поиск информации по заданной теме.

Задание: В соответствии с индивидуальным вариантом, используя поисковые системы, тематические каталоги и другие средства сети Internet, осуществить поиск необходимых информационных материалов для разработки индивидуального варианта информационной системы (ИС).

1. частности, поиск проектной документации на сходную (похожую) информационную систему, исходных текстов программной документации (полностью/частично отдельных модулей, которые можно использовать в разработке индивидуального варианта ИС, руководств и т.п.).

Найденная информация будет использоваться при выполнении последующих работ.

Отчет должен содержать следующую информацию:

1. организация поиска: средства поиска, атрибуты поиска, использованные ресурсы:
1. просто поисковые машины Internet,
2. специализированные поисковые средства,
3. форумы,
4. конференции Internet,
5. новостные рассылки,
6. иное (указать);
2. найденные первоисточники (указать адреса);
3. краткое описание источников (рецензия): оценка содержания, значимость для своей темы, удобство использования, найденные в источнике материалы и т. д.

Лабораторная работа №2. Предпроектное обследование.

Цель: Научиться проводить предпроектное обследование фирмы организации.

Задание: Разработать отчёт о предпроектном обследовании фирмы организации (по индивидуальному варианту) для внедрения в фирме организации Информационной системы.

Содержание отчета должно соответствовать приложенному к заданию примеру.

Оформление отчета должно соответствовать требованиям стандартов ГОСТ 19.104-78 ЕСПД. Основные надписи» по оформлению листа утверждения и титульного листа, ГОСТ 24.301-80 Система технической документации на АСУ. Общие требования к выполнению текстовых документов» по оформлению остальной части документа.

Лабораторная работа №3. Оформление отчета о предпроектном обследовании.

Цель: Научиться проводить предпроектное обследование фирмы организации.

Задание: Оформить отчёт о предпроектном обследовании фирмы организации (по индивидуальному варианту) для внедрения в фирме организации Информационной системы.

Содержание отчета должно соответствовать приложенному к заданию примеру.

Оформление отчета должно соответствовать требованиям стандартов ГОСТ 19.104-78 ЕСПД. Основные надписи» по оформлению листа утверждения и титульного листа, ГОСТ 24.301-80 Система технической документации на АСУ. Общие требования к выполнению текстовых документов» по оформлению остальной части документа.

Лабораторная работа №4. Сдача отчета о предпроектном обследовании.

Цель: Научиться проводить предпроектное обследование фирмы организации.

Задание: Сдать отчёт о предпроектном обследовании фирмы организации (по индивидуальному варианту) для внедрения в фирме организации Информационной системы.

Порядок сдачи работы: Представить отчёт о предпроектном обследовании фирмы/организации (по индивидуальному варианту) для разработки информационной системы.

Пример отчета о предпроектном обследовании фирмы приведен в Приложении 1.

Лабораторная работа №5. Разработка пояснительной записки к проекту ИС.

Цель: Научиться разрабатывать пояснительную записку к проекту ИС.

Задание: Разработать пояснительную записку к проекту ИС по индивидуальному варианту.

Оформление и содержание пояснительной записки должно соответствовать требованиям стандарта «ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению» и приложенного к заданию примера.

Лабораторная работа №6. Оформление пояснительной записки к проекту ИС.

Цель: Научиться разрабатывать пояснительную записку к проекту ИС.

Задание: Оформить пояснительную записку к проекту ИС по индивидуальному варианту.

Оформление и содержание пояснительной записки должно соответствовать требованиям стандарта «ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению» и приложенного к заданию примера.

Лабораторная работа №7. Сдача пояснительной записки к проекту ИС.

Цель: Научиться разрабатывать пояснительную записку к проекту ИС.

Задание: Сдать пояснительную записку к проекту ИС по индивидуальному варианту.

Оформление и содержание пояснительной записки должно соответствовать требованиям стандарта «ГОСТ 19.404-79. ЕСПД. Пояснительная записка. Требования к содержанию и оформлению» и приложенного к заданию примера.

Порядок сдачи работы: Представить отчёт, содержащий пояснительную записку к проекту ИС фирмы / организации (по индивидуальному варианту) для внедрения в фирме / организации информационной системы.

Пример пояснительной записки найдите в интернете.

Лабораторная работа №8. Разработка технического задания на ИС.

Цель: Научиться разрабатывать техническое задание на ИС.

Задание: Разработать техническое задание на ИС по индивидуальному варианту.

Оформление и содержание технического задания должно соответствовать требованиям стандарта «ГОСТ 19.201-78. ЕСПД. Техническое задание. Требования к содержанию и оформлению» и приложенного к заданию примера.

Лабораторная работа №9. Оформление технического задания на ИС.

Цель: Научиться разрабатывать техническое задание на ИС.

Задание: Оформить техническое задание на ИС по индивидуальному варианту.

Оформление и содержание технического задания должно соответствовать требованиям стандарта «ГОСТ 19.201-78. ЕСПД. Техническое задание. Требования к содержанию и оформлению» и приложенного к заданию примера.

Лабораторная работа №10. Сдача технического задания на ИС.

Цель: Научиться разрабатывать техническое задание на ИС.

Задание: Сдать техническое задание на ИС по индивидуальному варианту.

Оформление и содержание технического задания должно соответствовать требованиям стандарта «ГОСТ 19.201-78. ЕСПД. Техническое задание. Требования к содержанию и оформлению» и приложенного к заданию примера.

Порядок сдачи работы: Представить отчёт, содержащий техническое задание на ИС фирмы организации (по индивидуальному варианту) для внедрения в фирме организации информационной системы.

Пример технического задания найдите в интернете.

Лабораторная работа №11. Информационно-логическая модель информационной системы.

Цель работы: ознакомиться с общей характеристикой процесса проектирования

Краткие теоретические сведения

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО). ЖЦ ПО - это непрерывный процесс, который начинается с момента принятия решения

1. необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 (ISO – International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission – Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО. Более подробно стандарт описан в приложении 1.

Структура ЖЦ ПО по стандарту ISO/IEC 12207 базируется на трех группах процессов:

1. основные процессы ЖЦ ПО (приобретение, поставка, разработка, эксплуатация, сопровождение);
2. вспомогательные процессы, обеспечивающие выполнение основных процессов (документирование, управление конфигурацией, обеспечение качества, верификация, аттестация, оценка, аудит, решение проблем);
3. организационные процессы (управление проектами, создание инфраструктуры проекта, определение, оценка и улучшение самого ЖЦ, обучение).

Разработка включает в себя все работы по созданию ПО и его компонент в соответствии с заданными требованиями, включая оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проверки работоспособности и соответствующего качества программных продуктов, материалов, необходимых для организации обучения персонала и т.д. Разработка ПО включает в себя, как правило, анализ, проектирование и реализацию (программирование).

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения,

модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает выбор методов и инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки, разработку методов и средств испытаний ПО, обучение персонала и т.п. Обеспечение качества проекта связано с проблемами верификации, проверки и тестирования ПО.

Верификация – это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа. Проверка позволяет оценить соответствие параметров разработки с исходными требованиями. Проверка частично совпадает с тестированием, которое связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям. В процессе реализации проекта важное место занимают вопросы идентификации, описания и контроля конфигурации отдельных компонентов и всей системы в целом.

Управление конфигурацией является одним из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО. При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации конфигурационного учета, планирования и управления конфигурациями ПО отражены в проекте стандарта ISO 12207-2.

Каждый процесс характеризуется определенными задачами и методами их решения, исходными данными, полученными на предыдущем этапе, и результатами. Результатами анализа, в частности, являются функциональные модели, информационные модели и соответствующие им диаграммы. ЖЦ ПО носит итерационный характер: результаты очередного этапа часто вызывают изменения в проектных решениях, выработанных на более ранних этапах. Проектная документация представлена в приложении 1-2.

Стандарт ISO/IEC 12207 не предлагает конкретную модель ЖЦ и методы разработки ПО (под моделью ЖЦ понимается структура, определяющая последовательность выполнения и взаимосвязи процессов, действий и задач, выполняемых на протяжении ЖЦ. Модель ЖЦ зависит от специфики ИС и специфики условий, в которых последняя создается и функционирует). Его регламенты являются общими для любых моделей ЖЦ, методологий и технологий разработки. Стандарт ISO/IEC 12207 описывает структуру процессов ЖЦ ПО, но не конкретизирует в деталях, как реализовать или выполнить действия и задачи, включенные в эти процессы.

Порядок выполнения лабораторной работы:

1. Собрать предварительную информацию об исследуемом предприятии.
2. Сформулировать видение выполнения проекта и границы проекта.
3. Составить отчет об обследовании.

Лабораторная работа №12. Изучение структуры информационно-логической модели информационной системы.

Цель работы: изучить структуру информационно-логической модели информационной системы.

Краткие теоретические сведения

1. настоящему времени наибольшее распространение получили следующие две основные модели ЖЦ:

- ♣ каскадная модель (1970-1985 г.г.);
- ♣ спиральная модель (1986-1990 г.г.).

В изначально существовавших однородных ИС каждое приложение представляло собой единое целое. Для разработки такого типа приложений применялся каскадный способ. Его основной характеристикой является разбиение всей разработки на этапы, причем переход с одного этапа на следующий происходит только после того, как будет полностью завершена работа на текущем (рис. 1.1). Каждый этап завершается выпуском полного комплекта документации, достаточной для того, чтобы разработка могла быть продолжена другой командой разработчиков.

Положительные стороны применения каскадного подхода заключаются в следующем: на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности; выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Рис 1.1. Каскадная схема разработки ПО

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем, чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал следующий вид (рис. 1.2):

Рис. 1.2. Реальный процесс разработки ПО по каскадной схеме

Основным недостатком каскадного подхода является существенное запаздывание с получением результатов. Согласование результатов с пользователями производится только в точках, планируемых после завершения каждого этапа работ, требования к ИС "заморожены" в виде технического задания на все время ее создания. Таким образом, пользователи могут внести свои замечания только после того, как работа над системой будет полностью завершена. В случае неточного изложения требований или их изменения в течение длительного периода создания ПО, пользователи получают систему, не удовлетворяющую их потребностям. Модели

(как функциональные, так и информационные) автоматизируемого объекта могут устареть одновременно с их утверждением.

Для преодоления перечисленных проблем была предложена спиральная модель ЖЦ (рис. 1.3), делающая упор на начальные этапы ЖЦ: анализ и проектирование. На этих этапах реализуемость технических решений проверяется путем создания прототипов. Каждый виток спирали соответствует созданию фрагмента или версии ПО, на нем уточняются цели и характеристики проекта, определяется его качество и планируются работы следующего витка спирали. Таким образом, углубляются и последовательно конкретизируются детали проекта, и в результате выбирается обоснованный вариант, который доводится до реализации.

Разработка итерациями отражает объективно существующий спиральный цикл создания системы. Неполное завершение работ на каждом этапе позволяет переходить на следующий этап, не дожидаясь полного завершения работы на текущем. При итеративном способе разработки недостающую работу можно будет выполнить на следующей итерации. Главная же задача – как можно быстрее показать пользователям системы работоспособный продукт, тем самым активизируя процесс уточнения и дополнения требований.

Основная проблема спирального цикла – определение момента перехода на следующий этап. Для ее решения необходимо ввести временные ограничения на каждый из этапов жизненного цикла. Переход осуществляется в соответствии с планом, даже если не вся запланированная работа закончена. План составляется на основе статистических данных, полученных в предыдущих проектах, и личного опыта разработчиков.

Рис 1.3. Спиральная модель ЖЦ

Общие требования к методологии и технологии проектирования ИС

Методологии, технологии и инструментальные средства проектирования (CASE-средства) составляют основу проекта любой ИС. Методология реализуется через конкретные технологии и поддерживающие их стандарты, методики и инструментальные средства, которые обеспечивают выполнение процессов ЖЦ.

Технология проектирования определяется как совокупность трех составляющих:

♣ пошаговой процедуры, определяющей последовательность технологических операций проектирования (рис. 1.4);

1. критериев и правил, используемых для оценки результатов выполнения технологических операций;
2. нотаций (графических и текстовых средств), используемых для описания проектируемой системы.

Рис. 1.4. Представление технологической операции проектирования

Технологические инструкции, составляющие основное содержание технологии, должны состоять из описания последовательности технологических операций, условий, в зависимости от которых выполняется та или иная операция, и описаний самих операций.

Технология проектирования, разработки и сопровождения ИС должна удовлетворять следующим общим требованиям:

1. технология должна поддерживать полный ЖЦ ПО;
2. технология должна обеспечивать гарантированное достижение целей разработки ИС с заданным качеством и в установленное время;
3. технология должна обеспечивать возможность выполнения крупных проектов в виде подсистем (т.е. возможность декомпозиции проекта на составные части, разрабатываемые группами исполнителей ограниченной численности с последующей интеграцией составных частей). Опыт разработки крупных ИС показывает, что для повышения эффективности работ необходимо разбить проект на отдельные слабо связанные по данным и функциям подсистемы. Реализация подсистем должна выполняться отдельными группами специалистов. При этом необходимо обеспечить координацию ведения общего проекта и исключить дублирование результатов работ каждой проектной группы, которое может возникнуть в силу наличия общих данных и функций;
4. технология должна обеспечивать возможность ведения работ по проектированию отдельных подсистем небольшими группами (3-7 человек). Это обусловлено принципами управляемости коллектива и повышения производительности за счет минимизации числа внешних связей;
5. технология должна обеспечивать минимальное время получения работоспособной ИС. Речь идет не о сроках готовности всей ИС, а о сроках реализации отдельных подсистем. Реализация ИС в целом в короткие сроки может потребовать привлечения большого числа разработчиков, при этом эффект может оказаться ниже, чем при реализации в более короткие сроки отдельных подсистем меньшим числом разработчиков. Практика показывает, что даже при наличии полностью завершенного проекта, внедрение идет последовательно по отдельным подсистемам;
1. технология должна предусматривать возможность управления конфигурацией проекта, ведения версий проекта и его составляющих, возможность автоматического выпуска проектной документации и синхронизацию ее версий с версиями проекта;
2. технология должна обеспечивать независимость выполняемых проектных решений от средств реализации ИС (систем управления базами данных (СУБД), операционных систем, языков и систем программирования);
3. технология должна быть поддержана комплексом согласованных CASE-средств, обеспечивающих автоматизацию процессов, выполняемых на всех стадиях ЖЦ.

Порядок выполнения лабораторной работы:

4. Собрать предварительную информацию об исследуемом предприятии.
5. Сформулировать видение выполнения проекта и границы проекта.
6. Составить отчет об обследовании.

Лабораторная работа №13. Рассмотрение пакетной документации по информационно-логической модели информационной системы.

Цель работы: Рассмотреть проектную документацию.

Краткие теоретические сведения

Реальное применение любой технологии проектирования, разработки и сопровождения ИС в конкретной организации и конкретном проекте невозможно без выработки ряда стандартов (правил, соглашений), которые должны соблюдаться всеми участниками проекта. К таким стандартам относятся следующие:

7. стандарт проектирования;
8. стандарт оформления проектной документации;
9. стандарт пользовательского интерфейса.

Стандарт проектирования должен устанавливать:

1. набор необходимых моделей (диаграмм) на каждой стадии проектирования и степень их детализации;

2. правила фиксации проектных решений на диаграммах, в том числе: правила именования объектов (включая соглашения по терминологии), набор атрибутов для всех объектов и правила их заполнения на каждой стадии, правила оформления диаграмм, включая требования к форме и размерам объектов, и т. д.;

3. требования к конфигурации рабочих мест разработчиков, включая настройки операционной системы, настройки CASE-средств, общие настройки проекта и т. д.;

4. механизм обеспечения совместной работы над проектом, в том числе: правила интеграции подсистем проекта, правила поддержания проекта

10. одинаковом для всех разработчиков состоянии (регламент обмена проектной информацией, механизм фиксации общих объектов и т.д.), правила проверки проектных решений на непротиворечивость и т. д.

Стандарт оформления проектной документации должен устанавливать:

1. комплектность, состав и структуру документации на каждой стадии проектирования;
2. требования к ее оформлению (включая требования к содержанию разделов, подразделов, пунктов, таблиц и т.д.),
3. правила подготовки, рассмотрения, согласования и утверждения документации с указанием предельных сроков для каждой стадии;
4. требования к настройке издательской системы, используемой в качестве встроенного средства подготовки документации;
5. требования к настройке CASE-средств для обеспечения подготовки документации в соответствии с установленными требованиями.

Стандарт интерфейса пользователя должен устанавливать:

1. правила оформления экранов (шрифты и цветовая палитра), состав и расположение окон и элементов управления;
2. правила использования клавиатуры и мыши;
3. правила оформления текстов помощи;
4. перечень стандартных сообщений;
5. правила обработки реакции пользователя.

Порядок выполнения лабораторной работы:

6. Получить следующие данные:

1. Краткая информация о компании (профиль клиента).
2. Цели проекта.
3. Подразделения и пользователи системы.

7. На основе предварительной информации сформировать и согласовать с заказчиком общее представление о проекте.

8. Оформить результаты в виде отдельного документа – отчета об обследовании.

Лабораторная работа №14. Организация проектирования информационных систем.

Цель: ознакомиться с этапами разработки функциональной модели системы, анализом исходных данных для проектирования.

Краткие теоретические сведения

Функциональная методика IDEF0

Методологию IDEF0 можно считать следующим этапом развития хорошо известного графического языка описания функциональных систем SADT (Structured Analysis and Design Teqnique). Исторически IDEF0 как стандарт был разработан в 1981 году в рамках обширной программы автоматизации промышленных предприятий, которая носила обозначение ICAM (Integrated Computer Aided Manufacturing). Семейство стандартов IDEF унаследовало свое обозначение от названия этой программы (IDEF=Icam DEFinition), и последняя его редакция была выпущена в декабре 1993 года Национальным Институтом по Стандартам и Технологиям США (NIST).

Целью методики является построение функциональной схемы исследуемой системы, описывающей все необходимые процессы с точностью, достаточной для однозначного моделирования деятельности системы.

1. основе методологии лежат четыре основных понятия: функциональный блок, интерфейсная дуга, декомпозиция, глоссарий.

Функциональный блок (Activity Box) представляет собой некоторую конкретную *функцию* в рамках рассматриваемой системы. По требованиям стандарта название каждого функционального блока должно быть сформулировано в глагольном наклонении (например, "производить услуги"). На диаграмме функциональный блок изображается прямоугольником (рис. 2.1). Каждая из четырех сторон функционального блока имеет свое определенное значение (роль), при этом:

1. верхняя сторона имеет значение "Управление" (Control);
2. левая сторона имеет значение "Вход" (Input);
3. правая сторона имеет значение "Выход" (Output);
4. нижняя сторона имеет значение "Механизм" (Mechanism).

Рис. 2.1. Функциональный блок

Интерфейсная дуга (Arrow) отображает элемент системы, который обрабатывается функциональным блоком или оказывает иное влияние на *функцию*, представленную данным функциональным блоком. Интерфейсные дуги часто называют потоками или стрелками.

1. помощью интерфейсных дуг отображают различные объекты, в той или иной степени определяющие процессы, происходящие в системе. Такими объектами могут быть элементы реального мира (детали, вагоны, сотрудники и т.д.) или потоки данных и информации (документы, данные, инструкции и т.д.).

В зависимости от того, к какой из сторон функционального блока подходит данная интерфейсная дуга, она носит название "входящей", "исходящей" или "управляющей".

Необходимо отметить, что любой функциональный блок по требованиям стандарта должен иметь, по крайней мере, одну управляющую интерфейсную дугу и одну исходящую. Это и понятно – каждый процесс должен происходить по каким-то правилам (отображаемым управляющей дугой) и должен выдавать некоторый результат (выходящая дуга), иначе его рассмотрение не имеет никакого смысла.

Обязательное наличие управляющих интерфейсных дуг является одним из главных отличий стандарта IDEF0 от других методологий классов DFD (Data Flow Diagram) и WFD (Work Flow Diagram).

Декомпозиция (Decomposition) является основным понятием стандарта IDEF0. Принцип декомпозиции применяется при разбиении сложного процесса на составляющие его *функции*. При этом уровень детализации процесса определяется непосредственно разработчиком модели.

Декомпозиция позволяет постепенно и структурировано представлять модель системы в виде иерархической структуры отдельных диаграмм, что делает ее менее перегруженной и легко усваиваемой.

Последним из понятий IDEF0 является глоссарий (Glossary). Для каждого из элементов IDEF0 — диаграмм, функциональных блоков, интерфейсных дуг — существующий стандарт подразумевает создание и поддержание набора соответствующих определений, ключевых слов, повествовательных изложений и т.д., которые характеризуют объект, отображенный данным элементом. Этот набор называется глоссарием и является описанием сущности данного элемента. Глоссарий гармонично дополняет наглядный графический язык, снабжая диаграммы необходимой дополнительной информацией.

Модель IDEF0 всегда начинается с представления системы как единого целого – одного функционального блока с интерфейсными дугами, простирающимися за пределы рассматриваемой области. Такая диаграмма с одним функциональным блоком называется контекстной диаграммой.

1. пояснительном тексте к контекстной диаграмме должна быть указана цель (Purpose) построения диаграммы в виде краткого описания и зафиксирована точка зрения (Viewpoint).

Определение и формализация цели разработки IDEF0-модели является крайне важным моментом. Фактически цель определяет соответствующие области в исследуемой системе, на которых необходимо фокусироваться в первую очередь.

Точка зрения определяет основное направление развития модели и уровень необходимой детализации. Четкое фиксирование точки зрения позволяет разгрузить модель, отказавшись от детализации и исследования отдельных элементов, не являющихся необходимыми, исходя из выбранной точки зрения на систему. Правильный выбор точки зрения существенно сокращает временные затраты на построение конечной модели.

Выделение *подпроцессов*. В процессе декомпозиции функциональный блок, который в контекстной диаграмме отображает систему как единое целое, подвергается детализации на другой диаграмме. Получившаяся диаграмма второго уровня содержит функциональные блоки, отображающие главные подфункции функционального блока контекстной диаграммы, и называется дочерней (Child Diagram) по отношению к нему (каждый из функциональных блоков, принадлежащих дочерней диаграмме, соответственно называется дочерним блоком – Child Box). В свою очередь, функциональный блок — предок называется родительским блоком по отношению к дочерней диаграмме (Parent Box), а диаграмма, к которой он принадлежит – родительской диаграммой (Parent Diagram). Каждая из подфункций дочерней диаграммы может быть далее детализирована путем аналогичной декомпозиции соответствующего ей функционального блока. В каждом случае декомпозиции функционального блока все интерфейсные дуги, входящие в данный блок или исходящие из него, фиксируются на дочерней диаграмме. Этим достигается структурная целостность IDEF0– модели.

Иногда отдельные интерфейсные дуги высшего уровня не имеет смысла продолжать рассматривать на диаграммах нижнего уровня, или наоборот — отдельные дуги нижнего отражать на диаграммах более высоких уровней – это будет только перегружать диаграммы и делать их сложными для восприятия. Для решения подобных задач в стандарте IDEF0 предусмотрено понятие туннелирования. Обозначение "туннеля" (Arrow Tunnel) в виде двух круглых скобок вокруг начала интерфейсной дуги обозначает, что эта дуга не была унаследована от функционального родительского блока и появилась (из "туннеля") только на этой диаграмме. В свою очередь, такое же обозначение вокруг конца (стрелки) интерфейсной дуги в непосредственной близи от блока–приемника означает тот факт, что в дочерней по отношению к этому блоку диаграмме эта дуга отображаться и рассматриваться не будет. Чаще всего бывает, что отдельные объекты и соответствующие им интерфейсные дуги не рассматриваются на некоторых промежуточных уровнях иерархии, – в таком случае они сначала "погружаются в туннель", а затем при необходимости "возвращаются из туннеля".

Обычно IDEF0-модели несут в себе сложную и концентрированную информацию, и для того, чтобы ограничить их перегруженность и сделать удобочитаемыми, в стандарте приняты соответствующие ограничения сложности.

Рекомендуется представлять на диаграмме от трех до шести функциональных блоков, при этом количество подходящих к одному функциональному блоку (выходящих из одного функционального блока) интерфейсных дуг предполагается не более четырех.

Стандарт IDEF0 содержит набор процедур, позволяющих разрабатывать и согласовывать модель большой группой людей, принадлежащих к разным областям деятельности моделируемой системы. Обычно процесс разработки является итеративным и состоит из следующих условных этапов:

Создание модели группой специалистов, относящихся к различным сферам деятельности предприятия. Эта группа в терминах IDEF0 называется авторами (Authors). Построение первоначальной модели является динамическим процессом, в течение которого авторы опрашивают компетентных лиц о структуре различных процессов, создавая модели деятельности подразделений. При этом их интересуют ответы на следующие вопросы:

1. Что поступает в подразделение "на входе"?
2. Какие *функции* и в какой последовательности выполняются в рамках подразделения?
3. Кто является ответственным за выполнение каждой из *функций*?
4. Чем руководствуется исполнитель при выполнении каждой из *функций*?
5. Что является результатом работы подразделения (на выходе)?
6. На основе имеющихся положений, документов и результатов опросов создается черновик (Model Draft) модели.

Распространение черновика для рассмотрения, согласований и комментариев.

На этой стадии происходит обсуждение черновика модели с широким кругом компетентных лиц. При этом каждая из диаграмм черновой модели письменно критикуется и комментируется, а затем передается автору. Автор, в свою очередь, также письменно соглашается с критикой или отвергает ее с изложением логики принятия решения и вновь возвращает откорректированный черновик для дальнейшего рассмотрения. Этот цикл продолжается до тех пор, пока авторы и читатели не придут к единому мнению.

Официальное утверждение модели. Утверждение согласованной модели происходит руководителем рабочей группы в том случае, если у авторов модели и читателей отсутствуют разногласия по поводу ее адекватности. Окончательная модель представляет собой согласованное представление о предприятии (системе) с заданной точки зрения и для заданной цели.

Наглядность графического языка IDEF0 делает модель вполне читаемой

1. для лиц, которые не принимали участия в проекте ее создания, а также эффективной для проведения показов и презентаций. В дальнейшем на базе построенной модели могут быть организованы новые проекты, нацеленные на производство изменений в модели.

Порядок выполнения лабораторной работы:

1. Составить DFD диаграмму предметной области обследуемого предприятия.

Лабораторная работа №15. Функциональная методика потоков данных.

Цель: ознакомиться с этапами разработки функциональной модели системы, анализом исходных данных для проектирования. Рассмотреть защиту информации при реализации информационных процессов (ввод, вывод, передача, обработка, накопление, хранение); организационное обеспечение информационной безопасности; защита информации от несанкционированного доступа.

Краткие теоретические сведения

Функциональная методика потоков данных

Целью методики является построение модели рассматриваемой системы в виде диаграммы потоков данных (Data Flow Diagram — DFD), обеспечивающей правильное описание выходов (отклика системы в виде данных) при заданном воздействии на вход системы (подаче сигналов через внешние интерфейсы). Диаграммы потоков данных являются основным средством моделирования функциональных требований к проектируемой системе.

При создании диаграммы потоков данных используются четыре основных понятия: потоки данных, процессы (работы) преобразования входных потоков данных в выходные, внешние сущности, накопители данных (хранилища).

Потоки данных являются абстракциями, использующимися для моделирования передачи информации (или физических компонент) из одной части системы в другую. Потоки на диаграммах изображаются именованными стрелками, ориентация которых указывает направление движения информации.

Назначение процесса (работы) состоит в продуцировании выходных потоков из входных в соответствии с действием, задаваемым именем процесса. Имя процесса должно содержать глагол в неопределенной форме с последующим дополнением (например, "получить документы по отгрузке продукции"). Каждый процесс имеет уникальный номер для ссылок на него внутри диаграммы, который может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Хранилище (накопитель) данных позволяет на указанных участках определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет "срезы" потоков данных во времени. Информация, которую оно содержит, может использоваться в любое время после ее получения, при этом данные могут выбираться в любом порядке. Имя хранилища должно определять его содержимое и быть существительным.

Внешняя сущность представляет собой материальный объект вне контекста системы, являющейся источником или приемником системных данных. Ее имя должно содержать существительное, например, "склад товаров". Предполагается, что объекты, представленные как внешние сущности, не должны участвовать ни в какой обработке.

Кроме основных элементов, в состав DFD входят словари данных и миниспецификации.

Словари данных являются каталогами всех элементов данных, присутствующих в DFD, включая групповые и индивидуальные потоки данных, хранилища и процессы, а также все их атрибуты.

Миниспецификации обработки — описывают DFD- процессы нижнего уровня. Фактически миниспецификации представляют собой алгоритмы описания задач, выполняемых процессами: множество всех миниспецификаций является полной спецификацией системы.

Процесс построения DFD начинается с создания так называемой основной диаграммы типа "звезда", на которой представлен моделируемый процесс и все внешние сущности, с которыми он взаимодействует. В случае сложного основного процесса он сразу представляется в виде декомпозиции на ряд взаимодействующих процессов. Критериями сложности в данном случае являются: наличие большого числа внешних сущностей, многофункциональность системы, ее распределенный характер. Внешние сущности выделяются по отношению к основному процессу. Для их определения необходимо выделить поставщиков и потребителей основного процесса, т.е. все объекты, которые взаимодействуют с основным процессом. На этом этапе описание взаимодействия заключается в выборе глагола, дающего представление о том, как внешняя сущность использует основной процесс или используется им. Например, основной процесс – "учет обращений граждан", внешняя сущность – "граждане", описание взаимодействия – "подает заявления и получает ответы". Этот этап является принципиально важным, поскольку именно он определяет границы моделируемой системы.

Для всех внешних сущностей строится таблица событий, описывающая их взаимодействие с основным потоком. Таблица событий включает в себя наименование внешней сущности, событие, его тип (типичный для систем) или исключительный, реализующийся при определенных условиях) и реакцию системы.

На следующем шаге происходит декомпозиция основного процесса на набор взаимосвязанных процессов, обменивающихся потоками данных. Сами потоки не конкретизируются, определяется лишь характер взаимодействия. Декомпозиция завершается, когда процесс становится простым, т.е.:

1. процесс имеет два-три входных и выходных потока;
2. процесс может быть описан в виде преобразования входных данных в выходные;
1. процесс может быть описан в виде последовательного алгоритма. Для простых процессов строится миниспецификация – формальное описание алгоритма преобразования

входных данных в выходные. ***Миниспецификация*** удовлетворяет следующим требованиям: для каждого процесса строится одна спецификация; спецификация однозначно определяет входные и выходные потоки для данного процесса; спецификация не определяет способ преобразования входных потоков в выходные; спецификация ссылается на имеющиеся элементы, не вводя новые; спецификация по возможности использует стандартные подходы и *операции*.

После декомпозиции основного процесса для каждого *подпроцесса* строится аналогичная таблица внутренних событий.

Следующим шагом после определения полной таблицы событий выделяются потоки данных, которыми обмениваются процессы и внешние сущности. Простейший способ их выделения заключается в анализе таблиц событий. События преобразуются в потоки данных от инициатора события к запрашиваемому процессу, а реакции – в обратный поток событий. После построения входных и выходных потоков аналогичным образом строятся внутренние потоки. Для их выделения для каждого из внутренних процессов выделяются поставщики и потребители информации. Если поставщик или потребитель информации представляет процесс сохранения или запроса информации, то вводится хранилище данных, для которого данный процесс является интерфейсом.

После построения потоков данных диаграмма должна быть проверена на полноту и непротиворечивость. Полнота диаграммы обеспечивается, если в системе нет "повисших" процессов, не используемых в процессе преобразования входных потоков в выходные. Непротиворечивость системы обеспечивается выполнением наборов формальных правил о возможных типах процессов: на диаграмме не может быть потока, связывающего две внешние сущности – это взаимодействие удаляется из рассмотрения; ни одна сущность не может непосредственно получать или отдавать информацию в хранилище данных – хранилище данных является пассивным элементом, управляемым с помощью интерфейсного процесса; два хранилища данных не могут непосредственно обмениваться информацией – эти хранилища должны быть объединены.

К преимуществам методики DFD относятся:

1. возможность однозначно определить внешние сущности, анализируя потоки информации внутри и вне системы;
2. возможность проектирования сверху вниз, что облегчает построение модели "как должно быть";
3. наличие спецификаций процессов нижнего уровня, что позволяет преодолеть логическую незавершенность функциональной модели и построить полную функциональную спецификацию разрабатываемой системы.

К недостаткам модели отнесем: необходимость искусственного ввода управляющих процессов, поскольку управляющие воздействия (потоки) и управляющие процессы с точки зрения DFD ничем не отличаются от обычных; отсутствие понятия времени, т.е. отсутствие анализа временных промежутков при преобразовании данных (все ограничения по времени должны быть введены в спецификациях процессов).

Порядок выполнения лабораторной работы:

1. Составить ERD диаграмму предметной области обследуемого предприятия.
2. Провести классификацию пользователей по уровню доступа к данным.

Лабораторная работа №16. Объектно-ориентированная модель предметной области.

Цель: ознакомиться с объектно-ориентированным представлением программных систем.

Краткие теоретические сведения

Для объяснения сути объектной ориентации информационных систем воспользуемся аналогией с объектами реального мира. Окружающий мир состоит из объектов (object), пребывающих в состоянии (state), которое определяется текущими значениями атрибутов объекта.

Например, кружка на столе находится в состоянии filled (наполнена), поскольку она приспособлена для хранения жидкостей и в ней все еще есть кофе. Когда в ней нет больше кофе, состояние кружки можно определить как empty (пуста). Если она упадет на пол и разобьется, она перейдет в состояние broken (разбита).

Кофейная чашка, конечно, пассивна — она не обладает собственным поведением (behavior). Однако этого нельзя сказать о собаке или эвкалиптовом дереве за окном. Собака лает, дерево растет и т.д. Итак, некоторые объекты реального мира обладают поведением.

Все объекты реального мира обладают также идентичностью (identity) – постоянным свойством, с помощью которого мы отличаем один объект от другого. Если на моем столе стоят две чашки из одного набора, можно сказать, что они одинаковые, но не идентичные. Чашки одинаковые, потому что значения их свойств совпадают (они одинакового размера и формы, черного цвета и пустые). Однако, на объектно-ориентированном языке они не идентичны, поскольку их две, и есть выбор, которую из них использовать.

Реальные объекты, обладающие тремя свойствами (состояние, поведение, идентичность), образуют системы с естественным поведением. Естественные системы, безусловно, являются самыми сложными системами из всех известных. Никакая компьютерная система не может сравниться по сложности с животным или заводом.

Несмотря на сложность, естественные системы способны работать: они демонстрируют интересное поведение, могут приспосабливаться к внешним и внутренним изменениям, могут эволюционировать со временем и т.д. Вывод очевиден. Наверное, мы должны конструировать искусственные системы с помощью моделирования структуры и поведения естественных систем.

Искусственные системы являются моделью реальности. Кофейная чашка на экране компьютера всего лишь модель реальной "сущности" так же, как собака или эвкалиптовое дерево на экране. Кофейная чашка может быть, таким образом, смоделирована с помощью поведенческих свойств. Она может, к примеру, упасть на пол, если ее уронить. Действие "падения" можно смоделировать как поведенческую операцию (operation) чашки. Еще одним логическим "действием" чашки может быть операция "разбиться" при ударе об пол. Большинство, если не все, объекты в компьютерной системе "оживают" – они обладают поведением.

Объект-экземпляр

Объект – это экземпляр (instance) некоей "сущности". Он может быть одним из множества экземпляров одной и той же "сущности". Чашка – экземпляр множества всевозможных чашек.

Общее описание "сущности" называется классом (class). Поэтому объект является экземпляром класса. Класс также может нуждаться в конкретизации – он может быть объектом. Поэтому необходимо различать объект-экземпляр (instance object) и объект-класс (class object).

Для краткости объект-экземпляр часто называют объектом или экземпляром. Название "экземпляр объекта" сбивает с толку. Объектно-ориентированная система состоит из взаимодействующих объектов. В объектно-ориентированной системе нет ничего, кроме объектов, будь-то объект экземпляра (объект-экземпляр) или объект класса (объект-класс).

Порядок выполнения лабораторной работы:

1. Выделить действующие лица обследуемого предприятия.
2. Составить глоссарий проекта.
3. Выявить варианты использования.
4. Построить диаграмму вариантов использования.

Лабораторная работа №17. Язык моделирования UML.

Цель: знакомство с унифицированным языком моделирования UML.

Краткие теоретические сведения

Объектная нотация

1. языке UML объект обозначается прямоугольником с двумя отделениями. Верхнее отделение содержит имя объекта и имя класса, которому принадлежит объект. Синтаксис этой конструкции выглядит так:

objectname: classname.

Нижнее отделение содержит список имен атрибутов и значений. С помощью этого синтаксиса можно также показать типы атрибутов:

attributename: type = value.

На рис. 3.1 показан объект Course (Дисциплина) с именем cl. Объект обладает двумя атрибутами. Типы атрибутов не показаны – они заданы в определении класса.

cl: Course
course_number= COMP227
course_name= Requirements Analysis and System Design

Рис. 3.1. Объект-экземпляр

Важно иметь в виду, что объектная нотация не предусматривает в обозначении объекта особого "отделения" для перечня операций, которые может выполнять объект-экземпляр. Это связано с тем, что операции, выполняемые всеми объектами-экземплярами, идентичны, и поэтому хранить их в каждом объекте-экземпляре было бы накладно. Операции можно хранить в объекте-классе или же можно связать их с объектами-экземплярами другими средствами (реализованными в ПО базовой объектно-ориентированной системы).

Кооперация объектов

Количество объектов определенного класса может быть очень большим. Показать большое количество объектов на диаграмме нереально, а иногда просто невозможно. Объекты изображаются только для того, чтобы привести пример системы в определенный момент времени или проиллюстрировать, каким образом они кооперируются (collaborate) с течением времени при выполнении определенных задач. Например, чтобы упорядочить товары, может потребоваться установить кооперацию между объектом Stock (Склад) и объектом Purchase (Покупка).

Системные задачи выполняются объектами, которые активизируют операции (поведение) друг друга. Мы говорим, что они обмениваются сообщениями (message). Сообщения запускают операции на объектах, которые могут приводить к изменению состояний объектов и вызывать другие операции.

Рис. 3.2 иллюстрирует поток сообщений между четырьмя объектами. Скобки после имени сообщения указывают на то, что сообщение может принимать параметры (аналогично вызову функции в традиционном программировании). Объект Order (Заказ) предписывает объекту Shipment (Поставка) доставить заказ. Для этого объект Shipment инструктирует объект Stock о необходимости вычесть соответствующее количество товаров. Затем объект Stock анализирует новый уровень запаса и, если он оказывается ниже определенного значения, предписывает объекту Purchase сделать повторный заказ дополнительного объема товаров.

Хотя приведенное объяснение кооперации объектов выглядит как последовательность видов деятельности, а сообщения даже пронумерованы, в общем случае на порядок, в котором активируются объекты, не

накладывается строгих ограничений. Например, сообщения analyzeStockLevels (проанализировать уровни запаса) и reorderProducts (повторить заказ товара) могут выполняться в любой последовательности, возможно, независимо от сообщений shipOrder (доставить заказ) и subtractProduct (уменьшить количество товара) . Поэтому в дальнейшем при рассмотрении кооперации объектов на уровне реализации мы откажемся от нумерации сообщений.

Рис. 3.2. Кооперация объектов

Идентификация объектов

Каждому объекту при создании присваивается идентификатор объекта (object identifier – OID). Идентификатор объекта (OID) представляет собой дескриптор (handle) объекта – уникальный номер, который остается с объектом на протяжении всего времени его существования. Если объекту X необходимо отправить сообщение объекту Y, объект X каким-либо образом должен узнать OID объекта Y.

На практике для установления связи по OID между объектами существует два подхода, каждому из которых соответствует определенный тип связи.

1. Постоянная связь по OID.
2. Временная связь по OID.

Различие между этими видами связи определяется продолжительностью существования объекта. Время жизни некоторых объектов не превышает времени выполнения программы – они создаются программой и уничтожаются во время выполнения программы или по ее завершении. Это так называемые временные объекты (transient object). Другие объекты "переживают" выполнение программы – после завершения программы они запоминаются в долговременной дисковой памяти и доступны при следующем выполнении программы. Это так называемые постоянные объекты (persistent object).

Класс

Класс (class) – это дескриптор множества объектов, обладающих одинаковым набором атрибутов и операций. Он служит в качестве шаблона (template) для создания объектов. Каждый объект, созданный по шаблону, содержит значения атрибута, соответствующие типу атрибута, определенному в классе, и может вызвать операции, определенные в классе.

Графически класс представляется в виде прямоугольника с тремя отделениями, разделенными горизонтальными линиями, как показано на рис. 3.3. Верхнее отделение хранит имя класса. Среднее отделение содержит объявления всех атрибутов класса. Нижнее отделение содержит определения операций.

Атрибуты

Атрибут представляет собой пару тип-значение. Класс определяет типы атрибутов. Объекты содержат значения атрибутов. Рис. 3.4 иллюстрирует два класса с определениями имен и типов принадлежащих им атрибутов.

Тип атрибута может быть встроенным элементарным типом (primitive type) или другим классом. Элементарный тип – это непосредственно распознаваемый и поддерживаемый базовой объектно-ориентированной средой тип данных. Все типы атрибутов, показанных на рис. 3.4, обозначают элементарные типы.

Class name
Attributes
Operations ()

Рис.3.3. "Отделения" в представлении класса

Order_value: Curren

Course
Course_number: String

Course_name: String

Order

Order_number: Integer
Order_date: Date

Тип атрибута, обозначающий класс

Тип атрибута также может обозначать класс. Применительно к конкретному объекту некоторого класса подобный атрибут содержит значение идентификатора объекта (OID), указывающее на объект другого класса. В UML -моделях анализа атрибуты, типы которых обозначают классы, не приводятся в среднем «отделении» представления класса (в отличие от примитивных типов). Вместо этого они представляются с помощью ассоциации между классами. На рис. 3.5. показана подобная ассоциация между двумя классами.

Два имени на ассоциативной линии (the_shipment ([конкретная] постановка) и the_order ([конкретный] заказ)) представляют так называемые ролевые имена. Ролевое имя (rolename) определяет значение для одного из концов ассоциации и используется для перемещения к объекту другого класса в ассоциации.

1. реализованной системе ролевое имя (на противоположном конце ассоциации) становится атрибутом класса, тип которого является классом, на который указывает ролевое имя. На рис. 3.6 показано, как выглядят два класса с рис. 3.5 после того, как они были реализованы.

Рис. 3.5. Ролевые имена, обозначающие классы (модель анализа)

3.6. Атрибуты, обозначающие классы (модель реализации)

Видимость атрибутов

Объекты взаимодействуют с помощью отправки друг другу сообщений. Сообщение вызывает операцию класса. Операция обслуживает запрос вызывающего объекта с помощью доступа к значениям атрибутов в своем собственном объекте. Чтобы подобный сценарий стал возможен, операции должны быть видимы внешними объектами (сообщения должны «увидеть» операции). Говорят, что подобные операции обладают открытой видимостью.

Говорят, что операции инкапсулируют (encapsulate) атрибуты. Однако, инкапсуляция применима к классам. Один объект не может скрыть (инкапсулировать) ничего, кроме другого объекта того же класса.

Видимость обычно обозначается с помощью символов плюс и минус:

1. для открытой видимости; - для закрытой видимости.

В некоторых CASE-средствах эти символы заменены на графические

пиктограммы. На рис. 3.7продемонстрировано два графических представления для обозначения видимости атрибутов.

Рис. 3.7. Закрытые атрибуты и открытые операции

Операции

Объект содержит данные (атрибуты) и алгоритмы (операции) для работы с этими данными. Операция объявляется в классе. Процедура, реализующая операцию, называется методом (method).

Операция (или метод, если быть точным) вызывается с помощью отправленного ей сообщения. Имя сообщения и имя операции совпадают. Операция может содержать список параметров, которым в вызове сообщения могут быть присвоены определенные значения, и может возвращать значение вызывающему объекту.

Имя операции вместе со списком типов формальных аргументов называется сигнатурой (signature) операции. Сигнатура в пределах класса должна быть уникальной. Это значит, что класс может обладать множеством операций с одним и тем же именем, при условии, что списки типов параметров этих операций отличаются.

Объектно-ориентированная программа выполняется, реагируя на случайные события, которые инициирует пользователь. Источником событий выступает клавиатура, щелчки мышью, пункты меню, кнопки действий и любые другие входные устройства. Инициированное пользователем событие преобразуется в сообщение, отправляемое объекту. Для выполнения задания многим объектам требуется действовать совместно. Объекты "сотрудничают" с помощью вызова операций других объектов

На рис. 3.8 показаны операции классов, необходимые для поддержки совместных действий объектов, приведенных на рис. 3.2. Каждое сообщение на рис. 3.2 запрашивает операцию класса, обозначенного как адресат сообщения. В данном простом примере у класса Order операции отсутствуют. Объект класса Order инициирует совместную работу объектов. Объект Order (Заказ) предписывает объекту Shipment (Поставка) осуществить его "поставку". В результате поставки (т.е. выполнения заказа за счет товаров со склада) может потребоваться пополнение запаса новыми товарами.

Рис. 3.8. Операции служат основой совместных действий объектов

Принципы видимости операций не отличаются от принципов видимости атрибутов. Видимость операции определяет, является ли операция видимой объектам классов, отличных от класса, который определяет операцию. Если она видима, то ее видимость является открытой. В противном случае она будет закрытой. Пиктограммы перед именами операций на рис. 3.8 обозначают открытую видимость.

Как правило, большинство операций объектно-ориентированной системы обладают открытой видимостью. Чтобы объект мог предоставить сервис внешнему миру, операция "обслуживания" должна быть видима.

Однако большинство объектов имеют также несколько внутренних операций "местного" значения. Видимость этих операций должна быть закрыта. Они доступны только объектам класса, в котором они определены.

Следует различать видимость операции и область действия (scope) операции. Операцию можно вызвать на объекте-экземпляре или же на объекте-классе. В первом случае говорят, что операция обладает областью действия экземпляра. Во втором случае – областью действия класса. Например, операция, предназначенная для отыскания возраста работника, обладает областью действия экземпляра, а операция вычисления среднего возраста всех работников обладает областью действия класса.

Порядок выполнения лабораторной работы:

1. Произвести детализацию диаграммы вариантов использования.

2. Составить диаграмму классов.

Лабораторная работа № 18. Инструментальные средства проектирования информационных систем.

Цель: ознакомиться с классификацией инструментальных средств проектирования информационных систем,

Краткие теоретические сведения

Классификация CASE-средств

Таблица 4.1.
Классификация CASE-средств

Название	Фирма	Функции	Данные	События
BPWin	Logic Works	+	-	-
CASE Аналитик	Эйтекс	+	+	+
CASE/4/0	MicroTOOL	+	+	+
DatabaseDesigner	Oracle	-	+	-
Design/IDEF	Meta Software	+	+	-
Designer/2000	Oracle	+	+	-
EasyCASE	Evergreen CASE Tools	+	+	+
ERWin	Logic Works	-	+	-
I-CASE Yourdan	CAYENNE	+	+	+
Prokit*WORK	BENCHMDIS	+	+	-
S-Desidgner	Sybase/Powersoft	+	+	-
SILVERRun	CSA	+	+	+
Visible Analyst Workbench	Visible Systems	+	+	-

Все CASE-средства делятся на типы, категории и уровни. Классификация по типам отражает функциональную ориентацию CASE-средств в технологическом процессе.

1. **Анализ и проектирование**. Средства данной группы используются для создания спецификаций системы и ее проектирования; они поддерживают широко известные методологии проектирования. К таким средствам относятся: CASE, Аналитик (Эйтэкс), The Developer (ASYST Technologies), POSE (Computer Systems Advisers), ProKil*Workbench (McDonnell Douglas), Excelerator (Index Technology), Design-Aid (Naslec), Design Machine (Optima), MicroStep (Meta Systems), vsDesigner (Visucil Sofhvctrr), Anuijsl'Designer (Yourdon), Design/lDEF (Meta Soft\vare), BPWin (Logic Works). SELECT (Select Sofhvare Tools), System Architect (Popkin Software & Systems), Wesfmounl I-C.iSE Youi-don (Westmount Technology B. V. & CADRE Technologies), CASE/4/0 (microTOOL GmbH). Их целью является определение системных требований и свойств, которыми система должна обладать, а также создание проекта системы, удовлетворяющей этим требованиям и обладающей соответствующими свойствами. На выходе продуцируются спецификации компонент системы и интерфейсов, связывающих эти компоненты, а также "калька" архитектуры системы и детальная "калька" проекта, включающая алгоритмы и определения структур данных.

2. **Проектирование баз данных и файлов**. Средства данной группы обеспечивают логическое моделирование данных, автоматическое преобразование моделей данных в Третью Нормальную Форму, автоматическую генерацию схем БД и описаний форматов файлов на уровне программного кода: ERWin (Logic Works), Chen Toolkit (Chen & Asssociales), S-Designor (SDP), Designer2000 (Oracle), Silverrun (Computer Systems Advisers).

3. **Программирование**. Средства этой группы поддерживают этапы программирования и тестирования, а также автоматическую кодогенерацию из спецификаций, получая полностью документированную выполняемую программу: COBOL 2/Workhench (Mikro Focus), DECASE (DEC), NETRON/CAP (Netron), APS (Sage Softwtire). Помимо диаграммеров различного назначения и средств поддержки работы с репозиторием. в эту группу средств включены и традиционные генераторы кодов, анализаторы кодов (как в статике, так и в динамике), генераторы наборов тестов, анализаторы покрытия тестами, отладчики.

4. **Сопровождение и реинжиниринг**. К таким средствам относятся документаторы, анализаторы программ, средства реструктурирования и реинженирингa: Adpac CASE Tools (Adpac). Scan/COBOL, u Superstructure (Computer Data Systems), Inspector/Recoder (Language Technology). Их целью является корректировка, изменение, анализ, преобразование и реинжениринг существующей системы. Средства позволяют осуществлять поддержку всей системной документации, включая коды, спецификации, наборы тестов; контролировать покрытие тестами для оценки полноты тестируемости; управлять функционированием системы и т.п. Особый интерес представляют средства обеспечения мобильности (в CASE они получили название средств миграции) и реинжиниринга. К средствам миграции относятся трансляторы, конверторы, макрогенераторы и др., позволяющие обеспечить перенос существующей системы в новое операционное или аппаратурное окружение.

Средства реинжиниринга включают:

1. статические анализаторы для продуцирования схем системы ПО из

5. кодов, оценки влияния модификаций (например, "эффекта ряби" - внесение изменений с целью исправления ошибок порождает новые ошибки);

1. динамические анализаторы (обычно, компиляторы и интерпретаторы с встроенными отладочными возможностями);

2. документаторы, позволяющие автоматически получать обновленную документацию при изменении кода;

3. редакторы кодов, автоматически изменяющие при редактировании и все предшествующие коду структуры (например, спецификации);

4. средства доступа к спецификациям, их модификации и генерации нового (модифицированного) кода;

5. средства реверсного инжиниринга, транслирующие коды в спецификации.

5) Окружение. Средства поддержки платформ для интеграции, создания и придания товарного вида CASE-средствам: Multi/Cam (AGS Management Systems), Design/OA (Meta Software).

6) Управление проектом. Средства, поддерживающие планирование, контроль. руководство, взаимодействие, т.е. функции, необходимые в процессе разработки и сопровождения проектов: Project Workbench (Applied Business Technology). Классификация по категориям определяет уровень интегрированности по выполняемым функциям и включает вспомогательные программы (tools), пакеты разработчика (toolkit) и инструментальные средства (workbench). Категория tools обозначает вспомогательный пакет, решающий небольшую автономную задачу, принадлежащую проблеме более широкого масштаба. Категория toolkit представляет совокупность интегрированных программных средств, обеспечивающих помощь для одного из классов программных задач; использует репозиторий для всей технической и управляющей информации о проекте, концентрируясь при этом на поддержке, как правило, одной фазы или одного этапа разработки ПО.

Категория workbench представляет собой интеграцию программных средств, которые поддерживают системный анализ, проектирование и разработку ПО; используют репозиторий, содержащий всю техническую и управляющую информацию о проекте; обеспечивают автоматическую передачу системной информации между разработчиками и этапами разработки; организуют поддержку практически полного ЖЦ (от анализтребований и проектирования ПО до получения документированной выполняемой программы). Workbench, по сравнению с toolkit, обладает более высокой степенью интеграции выполняемых функций, большей самостоятельностью и автономностью использования, а также наличием тесной связи с системными и техническими средствами аппаратно-вычислительной среды, на которой workbench функционирует. По существу, workbench может рассматриваться как автоматизированная рабочая станция, используемая как инструментарий для автоматизации всех или отдельных совокупностей работ по созданию ПО.

Классификация по уровням связана с областью действия CASE в пределах жизненного цикла ПО. Однако четкие критерии определения границ между уровнями не установлены, поэтому данная классификация имеет, вообще говоря, качественный характер.

Верхние (Upper) CASE часто называют средствами компьютерного планирования. Они призваны повышать эффективность деятельности руководителей фирмы и проекта путем сокращения затрат на определение политики фирмы и на создание общего плана проекта. Этот план включает цели и стратегии их достижения, основные действия в свете целей и задач фирмы, установление стандартов на различные виды взаимосвязей и т.д. Использование верхних CASE позволяет построить модель предметной области, отражающую всю существующую специфику. Она направлена на понимание общего и частного механизмов функционирования, имеющихся возможностей, ресурсов, целей проекта в соответствии с назначением фирмы. Эти средства позволяют проводить анализ различных сценариев (в том числе наилучших и наихудших), накапливая информацию для принятия оптимальных решений. Средние (Middle) CASE считаются средствами поддержки этапов анализа требований и проектирования спецификаций и структуры ПО. Их использование существенно сокращает цикл разработки проекта; при этом важную роль играет возможность накопления и хранения знаний, обычно имеющихся только в голове разработчика-аналитика, что позволит использовать накопленные решения при создании других проектов. Основная выгода от использования среднего CASE состоит в значительном облегчении проектирования систем, проектирование превращается в итеративный процесс, включающий следующие действия:

1. пользователь обсуждает с аналитиком требования к проектируемой системе;
2. аналитик документирует эти требования, используя диаграммы и словари входных данных;
3. пользователь проверяет эти диаграммы и словари, при необходимости модифицируя их;
4. аналитик отвечает на эти модификации, изменяя соответствующие спецификации.

Кроме того, средние CASE обеспечивают возможности быстрого документирования требований и быстрого прототипирования.

Нижние (Lower) CASE являются средствами разработки ПО (при этом может использоваться до 30% спецификаций, созданных средствами среднего CASE). Они содержат системные словари и графические средства, исключающие необходимость разработки физических спецификаций. Имеются системные спецификации, которые непосредственно переводятся в программные коды разрабатываемой системы (при этом автоматически генерируется до 80-90% кодов). На эти средства возложены также функции тестирования, управления конфигурацией, формирования документации. Главными преимуществами нижних CASE являются: значительное уменьшение времени на разработку, облегчение модификаций, поддержка возможностей прототипирования (совместно со средними CASE).

Порядок выполнения лабораторной работы:

1. Выполнить сравнительный анализ рассмотренных ранее методов проектирования применительно к обследуемому предприятию.

Лабораторная работа № 19.
Примеры инструментальных средств проектирования информационных систем.

Цель: ознакомиться с инструментальными средствами проектирования информационных систем.

Краткие теоретические сведения

Примеры различных CASE-средств

1. Silverrun

CASE-средство Silverrun американской фирмы Computer Systems Advisers, Inc. (CSA) используется для анализа и проектирования ИС бизнес-класса и ориентировано в большей степени на спиральную модель ЖЦ. Оно применимо для поддержки любой методологии, основанной на раздельном построении функциональной и информационной моделей (диаграмм потоков данных и диаграмм "сущность-связь").

Настройка на конкретную методологию обеспечивается выбором требуемой графической нотации моделей и набора правил проверки проектных спецификаций. В системе имеются готовые настройки для наиболее распространенных методологий: DATARUN (основная методология, поддерживаемая Silverrun), Gane/Sarson, Yourdon/DeMarco, Merise, Ward/Mellor, Information Engineering. Для каждого понятия, введенного в проекте имеется возможность добавления собственных описателей. Архитектура Silverrun позволяет наращивать среду разработки по мере необходимости.

Структура и функции

Silverrun имеет модульную структуру и состоит из четырех модулей, каждый из которых является самостоятельным продуктом и может приобретаться и использоваться без связи с остальными модулями.

Модуль построения моделей бизнес-процессов в форме диаграмм потоков данных (BPM - Business Process Modeler) позволяет моделировать функционирование обследуемой организации или создаваемой ИС. В модуле BPM обеспечена возможность работы с моделями большой сложности:

автоматическая перенумерация, работа с деревом процессов (включая визуальное перетаскивание ветвей), отсоединение и присоединение частей модели для коллективной разработки. Диаграммы могут изображаться в нескольких предопределенных нотациях, включая Yourdon/DeMarco и Gane/Sarson. Имеется также возможность создавать собственные нотации, в том числе добавлять в число изображаемых на схеме дескрипторов определенные пользователем поля.

Модуль концептуального моделирования данных (ERX – Entity-Relationship eXpert) обеспечивает построение моделей данных "сущность-связь", не привязанных к конкретной реализации. Этот модуль имеет встроенную экспертную систему, позволяющую создать корректную нормализованную модель данных посредством ответов на содержательные вопросы о взаимосвязи данных. Возможно автоматическое построение модели данных из описаний структур данных. Анализ функциональных зависимостей атрибутов дает

возможность проверить соответствие модели требованиям третьей нормальной формы и обеспечить их выполнение. Проверенная модель передается в модуль RDM.

Модуль реляционного моделирования (RDM – Relational Data Modeler) позволяет создавать детализированные модели "сущность -связь", предназначенные для реализации в реляционной базе данных. В этом модуле документируются все конструкции, связанные с построением базы данных: индексы, триггеры, хранимые процедуры и т.д. Гибкая изменяемая нотация и расширяемость репозитория позволяют работать по любой методологии. Возможность создавать подсхемы соответствует подходу ANSI SPARC к представлению схемы базы данных. На языке подсхем моделируются как узлы распределенной обработки, так и пользовательские представления. Этот модуль обеспечивает проектирование и полное документирование реляционных баз данных.

Менеджер репозитория рабочей группы (WRM – Workgroup Repository Manager) применяется как словарь данных для хранения общей для всех моделей информации, а также обеспечивает интеграцию модулей Silverrun в единую среду проектирования.

Платой за высокую гибкость и разнообразие изобразительных средств построения моделей является такой недостаток Silverrun, как отсутствие жесткого взаимного контроля между компонентами различных моделей (например, возможности автоматического распространения изменений между DFD различных уровней декомпозиции). Следует, однако, отметить, что этот недостаток может иметь существенное значение только в случае использования каскадной модели ЖЦ ПО.

Взаимодействие с другими средствами

Для автоматической генерации схем баз данных у Silverrun существуют мосты к наиболее распространенным СУБД: Oracle, Informix, DB2, Ingres, Progress, SQL Server, SQLBase, Sybase. Для передачи данных в средства разработки приложений имеются мосты к языкам 4GL: JAM, PowerBuilder, SQL Windows, Uniface, NewEra, Delphi. Все мосты позволяют загрузить в Silverrun RDM информацию из каталогов соответствующих СУБД или языков 4GL. Это позволяет документировать, перепроектировать или переносить на новые платформы уже находящиеся в эксплуатации базы данных и прикладные системы. При использовании моста Silverrun расширяет свой внутренний репозиторий специфичными для целевой системы атрибутами. После определения значений этих атрибутов генератор приложений переносит их во внутренний каталог среды разработки или использует при генерации кода на языке SQL. Таким образом, можно полностью определить ядро базы данных с использованием всех возможностей конкретной СУБД: триггеров, хранимых процедур, ограничений ссылочной целостности. При создании приложения на языке 4GL данные, перенесенные из репозитория Silverrun, используются либо для автоматической генерации интерфейсных объектов, либо для быстрого их создания вручную.

Для обмена данными с другими средствами автоматизации проектирования, создания специализированных процедур анализа и проверки проектных спецификаций, составления специализированных отчетов в соответствии с различными стандартами в системе Silverrun имеется три способа выдачи проектной информации во внешние файлы:

1. Система отчетов. Можно, определив содержимое отчета по репозиторию, выдать отчет в текстовый файл. Этот файл можно затем загрузить в текстовый редактор или включить в другой отчет;
2. Система экспорта/импорта. Для более полного контроля над структурой файлов в системе экспорта/импорта имеется возможность определять не только содержимое экспортного файла, но и разделители записей, полей в записях, маркеры начала и конца текстовых полей. Файлы с указанной структурой можно не только формировать, но и загружать в репозиторий. Это дает возможность обмениваться данными с различными системами: другими CASE-средствами, СУБД, текстовыми редакторами и электронными таблицами;
3. Хранение репозитория во внешних файлах через ODBC-драйверы. Для доступа к данным репозитория из наиболее распространенных систем управления базами данных обеспечена возможность хранить всю проектную информацию непосредственно в формате этих СУБД.

Групповая работа

Групповая работа поддерживается в системе Silverrun двумя способами:

1. В стандартной однопользовательской версии имеется механизм контролируемого разделения и слияния моделей. Разделив модель на части, можно раздать их нескольким разработчикам. После детальной доработки модели объединяются в единые спецификации;

2. Сетевая версия Silverrun позволяет осуществлять одновременную групповую работу с моделями, хранящимися в сетевом репозитории на базе СУБД Oracle, Sybase или Informix. При этом несколько разработчиков могут работать с одной и той же моделью, так как блокировка объектов происходит на уровне отдельных элементов модели.

Среда функционирования

Имеются реализации Silverrun трех платформ - MS Windows, Macintosh

1. OS/2 Presentation Manager - с возможностью обмена проектными данными между ними.

Для функционирования в среде Windows необходимо иметь компьютер с процессором модели не ниже i486 и оперативную память объемом не менее 8 Мб (рекомендуется 16 Мб). На диске полная инсталляция Silverrun занимает 20 Мб.

1. Rational Rose

Rational Rose – CASE-средство фирмы Rational Software Corporation (США) – предназначено для автоматизации этапов анализа и проектирования ПО, а также для генерации кодов на различных языках и выпуска проектной документации. Rational Rose использует синтез-методологию объектно-ориентированного анализа и проектирования, основанную на подходах трех ведущих специалистов в данной области: Буча, Рамбо и Джекобсона. Разработанная ими универсальная нотация для моделирования объектов (UML - Unified Modeling Language) претендует на роль стандарта в области объектно-ориентированного анализа и проектирования. Конкретный вариант Rational Rose определяется языком, на котором генерируются коды программ (C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro) . Основной вариант – Rational Rose/C++ – позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также генерировать программные коды на C++. Кроме того, Rational Rose содержит средства реинжиниринга программ, обеспечивающие повторное использование программных компонент в новых проектах.

Структура и функции

1. основе работы Rational Rose лежит построение различного рода диаграмм и спецификаций, определяющих логическую и физическую структуры модели, ее статические и динамические аспекты. В их число входят диаграммы классов, состояний, сценариев, модулей, процессов.

1. составе Rational Rose можно выделить 6 основных структурных компонент: репозиторий, графический интерфейс пользователя, средства просмотра проекта (browser), средства контроля проекта, средства сбора статистики и генератор документов. К ним добавляются генератор кодов (индивидуальный для каждого языка) и анализатор для C++, обеспечивающий реинжиниринг – восстановление модели проекта по исходным текстам программ.

Репозиторий представляет собой объектно-ориентированную базу данных. Средства просмотра обеспечивают "навигацию" по проекту, в том числе, перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому и т. д. Средства контроля и сбора статистики дают возможность находить и устранять ошибки по мере развития проекта, а не после завершения его описания. Генератор отчетов формирует тексты выходных документов на основе содержащейся в репозитории информации.

Средства автоматической генерации кодов программ на языке C++, используя информацию, содержащуюся в логической и физической моделях проекта, формируют файлы заголовков и файлы описаний классов и объектов. Создаваемый таким образом скелет программы может быть уточнен путем прямого программирования на языке C++. Анализатор кодов C++ реализован в виде отдельного программного модуля. Его назначение состоит в

том, чтобы создавать модули проектов в форме Rational Rose на основе информации, содержащейся в определяемых пользователем исходных текстах на C++. В процессе работы анализатор осуществляет контроль правильности исходных текстов и диагностику ошибок. Модель, полученная в результате его работы, может целиком или фрагментарно использоваться в различных проектах. Анализатор обладает широкими возможностями настройки по входу и выходу. Например, можно определить типы исходных файлов, базовый компилятор, задать, какая информация должна быть включена в формируемую модель и какие элементы выходной модели следует выводить на экран. Таким образом, Rational Rose/C++ обеспечивает возможность повторного использования программных компонент.

2. результате разработки проекта с помощью CASE-средства Rational Rose формируются следующие документы:

1. диаграммы классов;
2. диаграммы состояний;
3. диаграммы сценариев;
4. диаграммы модулей;
5. диаграммы процессов;
6. спецификации классов, объектов, атрибутов и операций
7. заготовки текстов программ;
8. модель разрабатываемой программной системы.

Последний из перечисленных документов является текстовым файлом, содержащим всю необходимую информацию о проекте (в том числе необходимую для получения всех диаграмм и спецификаций).

Тексты программ являются заготовками для последующей работы программистов. Они формируются в рабочем каталоге в виде файлов типов .h (заголовки, содержащие описания классов) и .cpp (заготовки программ для методов). Система включает в программные файлы собственные комментарии, которые начинаются с последовательности символов //##. Состав информации, включаемой в программные файлы, определяется либо по умолчанию, либо по усмотрению пользователя. В дальнейшем эти исходные тексты развиваются программистами в полноценные программы.

Взаимодействие с другими средствами и организация групповой работы

Rational Rose интегрируется со средством PVCS для организации групповой работы и управления проектом и со средством SoDA – для документирования проектов. Интеграция Rational Rose и SoDA обеспечивается средствами SoDA.

Для организации групповой работы в Rational Rose возможно разбиение модели на управляемые подмодели. Каждая из них независимо сохраняется на диске или загружается в модель. В качестве подмодели может выступать категория классов или подсистема.

Для управляемой подмодели предусмотрены операции:

1. загрузка подмодели в память;
2. выгрузка подмодели из памяти;
3. сохранение подмодели на диске в виде отдельного файла;
4. установка защиты от модификации;
5. замена подмодели в памяти на новую.

Наиболее эффективно групповая работа организуется при интеграции Rational Rose со специальными средствами управления конфигурацией и контроля версий (PVCS). В этом случае защита от модификации устанавливается на все управляемые подмодели, кроме тех, которые выделены конкретному разработчику. В этом случае признак защиты от записи устанавливается для файлов, которые содержат подмодели, поэтому при считывании "чужих" подмоделей защита их от модификации сохраняется и случайные воздействия окажутся невозможными.

Среда функционирования

Rational Rose функционирует на различных платформах: IBM PC (в среде Windows), Sun SPARC stations (UNIX, Solaris, SunOS), Hewlett-Packard (HP UX), IBM RS/6000 (AIX).

Для работы системы необходимо выполнение следующих требований:

1. Платформа Windows - процессор 80386SX или выше (рекомендуется 80486), память8Мб (рекомендуется 12Мб), пространство на диске 8Мб + 1-3Мб для одной модели.

2. Платформа UNIX – память 32+(16*число пользователей)Мб, пространство на диске 30Мб + 20 при инсталляции + 1-3Мб для одной модели.

Совместимость по версиям обеспечивается на уровне моделей.

3. Vantage Team Builder (Westmount I-CASE)

Vantage Team Builder представляет собой интегрированный программный продукт, ориентированный на реализацию каскадной модели ЖЦ ПО и поддержку полного ЖЦ ПО.

Структура и функции

Vantage Team Builder обеспечивает выполнение следующих функций:

1. проектирование диаграмм потоков данных, "сущность-связь", структур данных, структурных схем программ и последовательностей экранных форм;
2. проектирование диаграмм архитектуры системы – SAD (проектирование состава и связи вычислительных средств, распределения задач системы между вычислительными средствами, моделирование отношений типа "клиент-сервер", анализ использования менеджеров транзакций и особенностей функционирования систем в реальном времени);
3. генерация кода программ на языке 4GL целевой СУБД с полным обеспечением программной среды и генерация SQL-кода для создания таблиц БД, индексов, ограничений целостности и хранимых процедур;
4. программирование на языке C со встроенным SQL;
5. управление версиями и конфигурацией проекта;
6. многопользовательский доступ к репозиторию проекта;
7. генерация проектной документации по стандартным и индивидуальным шаблонам;
8. экспорт и импорт данных проекта в формате CDIF (CASE Data Interchange Format).

Vantage Team Builder поставляется в различных конфигурациях в зависимости от используемых СУБД (ORACLE, Informix, Sybase или Ingres) или средств разработки приложений (Uniface). Конфигурация Vantage Team Builder for Uniface отличается от остальных некоторой степенью ориентации на спиральную модель ЖЦ ПО за счет возможностей быстрого прототипирования, предоставляемых Uniface. Для описания проекта ИС используется достаточно большой набор диаграмм, конкретные варианты которого для наиболее распространенных конфигураций приведены ниже в таблице 4.2

Таблица 4.2.

Варианты диаграмм для различных конфигураций

Тип диаграммы	Обозначение	Vantage Team Builder for ORACLE	Vantage Team Builder for Informix	Vantage Team Builder for Uniface
Сущность-связь	ERD	+	+	+
Потоков данных	DFD	+	+	+
Структур данных	DSD	+	+	+
Архитектуры системы	SAD	+	+	+
Потоков управления	CSD	+	+	+
Типов данных	DTD	+	+	+
Структуры меню	MSD	+		
Последовательности блоков	BSD	+		
Последовательности форм	FSD		+	+

Содержимого форм	FCD		+	+
Переходов состояний	STD	+	+	+
Структурных схем	SCD	+	+	+

При построении всех типов диаграмм обеспечивается контроль соответствия моделей синтаксису используемых методов, а также контроль соответствия одноименных элементов и их типов для различных типов диаграмм.

При построении DFD обеспечивается контроль соответствия диаграмм различных уровней декомпозиции. Контроль за правильностью верхнего уровня DFD осуществляется с помощью матрицы списков событий (ELM). Для контроля за декомпозицией составных потоков данных используется несколько вариантов их описания: в виде диаграмм структур данных (DSD) или в нотации БНФ (форма Бэкуса-Наура).

Для построения SAD используется расширенная нотация DFD, дающая возможность вводить понятия процессоров, задач и периферийных устройств, что обеспечивает наглядность проектных решений.

При построении модели данных в виде ERD выполняется ее нормализация и вводится определение физических имен элементов данных и таблиц, которые будут использоваться в процессе генерации физической схемы данных конкретной СУБД. Обеспечивается возможность определения альтернативных ключей сущностей и полей, составляющих дополнительные точки входа в таблицу (поля индексов), и мощности отношений между сущностями.

Наличие универсальной системы генерации кода, основанной на специфицированных средствах доступа к репозиторию проекта, позволяет поддерживать высокий уровень исполнения проектной дисциплины разработчиками: жесткий порядок формирования моделей; жесткая структура и содержимое документации; автоматическая генерация исходных кодов программ и т.д. – все это обеспечивает повышение качества и надежности разрабатываемых ИС.

Для подготовки проектной документации могут использоваться издательские системы FrameMaker, Interleaf или Word Perfect. Структура и состав проектной документации могут быть настроены в соответствии с заданными стандартами. Настройка выполняется без изменения проектных решений.

При разработке достаточно крупной ИС вся система в целом соответствует одному проекту как категории Vantage Team Builder. Проект может быть декомпозирован на ряд систем, каждая из которых соответствует некоторой относительно автономной подсистеме ИС и разрабатывается независимо от других. В дальнейшем системы проекта могут быть интегрированы.

Процесс проектирования ИС с использованием Vantage Team Builder реализуется в виде 4-х последовательных фаз (стадий) – анализа, архитектуры, проектирования и реализации, при этом законченные результаты каждой стадии полностью или частично переносятся (импортируются) в следующую фазу. Все диаграммы, кроме ERD, преобразуются в другой тип или изменяют вид в соответствии с особенностями текущей фазы. Так, DFD преобразуются в фазе архитектуры в SAD, DSD – в DTD. После завершения импорта логическая связь с предыдущей фазой разрывается, т.е. в диаграммы могут вноситься все необходимые изменения.

Взаимодействие с другими средствами

Конфигурация Vantage Team Builder for Uniface обеспечивает совместное использование двух систем в рамках единой технологической среды проектирования, при этом схемы БД (SQL-модели) переносятся в репозиторий Uniface, и, наоборот, прикладные модели, сформированные средствами Uniface, могут быть перенесены в репозиторий Vantage Team Builder. Возможные рассогласования между репозиториями двух систем устраняются с помощью специальной утилиты. Разработка экранных форм в среде Uniface выполняется на базе диаграмм последовательностей форм (FSD) после импорта SQL- модели. Технология разработки ИС на базе данной конфигурации показана на рисунке 4.1.

Структура репозитория (хранящегося непосредственно в целевой СУБД) и интерфейсы Vantage Team Builder являются открытыми, что в принципе позволяет интеграцию с любыми другими средствами.

Среда функционирования

Vantage Team Builder функционирует на всех основных UNIX-платформах (Solaris, SCO UNIX, AIX, HP-UX) и VMS.

Vantage Team Builder можно использовать в конфигурации "клиент-сервер", при этом база проектных данных может располагаться на сервере, а рабочие места разработчиков могут быть клиентами.

Рис. 4.1. Взаимодействие Vantage Team Builder и Uniface

Вспомогательные средства поддержки жизненного цикла ПО

Средства конфигурационного управления

Цель конфигурационного управления (КУ) – обеспечить управляемость

1. контролируемость процессов разработки и сопровождения ПО. Для этого необходима точная и достоверная информация о состоянии ПО и его компонент в каждый момент времени, а также о всех предполагаемых и выполненных изменениях.

Для решения задач КУ применяются методы и средства обеспечивающие идентификацию состояния компонент, учет номенклатуры всех компонент и модификаций системы в целом, контроль за вносимыми изменениями в компоненты, структуру системы и ее функции, а также координированное управление развитием функций и улучшением характеристик системы.

Наиболее распространенным средством КУ является PVCS фирмы Intersolv (США), включающее ряд самостоятельных продуктов: PVCS Version Manager, PVCS Tracker, PVCS Configuration Builder и PVCS Notify.

PVCS Version Manager предназначен для управления всеми компонентами проекта и ведения планомерной многоверсионной и многоплатформенной разработки силами команды разработчиков в условиях одной или нескольких локальных сетей. Понятие "проект" трактуется как совокупность файлов. В процессе работы над проектом промежуточное состояние файлов периодически сохраняется в архиве проекта, ведутся записи о времени сохранения, соответствии друг другу нескольких вариантов разных файлов проекта. Кроме этого, фиксируются имена разработчиков, ответственных за тот или иной файл, состав файлов промежуточных версий проекта и др. Это позволяет вернуться при необходимости к

какому-либо из предыдущих состояний файла (например, при обнаружении ошибки, которую в данный момент трудно исправить).

PVCS Version Manager предназначен для использования в рабочих группах. Система блокировок, реализованная в PVCS Version Manager позволяет предотвратить одновременное внесение изменений в один и тот же файл. В то же время, PVCS Version Manager позволяет разработчикам работать с собственными версиями общего файла с полуавтоматическим разрешением конфликтов между ними.

Доступ к архивам PVCS Version Manager возможен не только через сам Version Manager, но и из более чем 50 инструментальных средств, в том числе MS Visual C и MS Visual Basic, Uniface, PowerBuilder, SQL Windows, JAM, Delphi, Paradox и др.

Результатом работы PVCS Version Manager является созданный средствами файловой системы репозиторий, хранящий в компактной форме все рабочие версии программного продукта вместе с необходимыми комментариями и метками.

PVCS Version Manager функционирует в среде MS Windows, Windows 95, Windows NT, OS/2, SunOS, Solaris, HP-UX, AIX и SCO UNIX и может исполняться на любом персональном компьютере с процессором 80386 или выше, рабочих станциях Sun, HP и IBM (RS-6000).

Другим средством конфигурационного управления является PVCS Tracker – специализированная надстройка над офисной электронной почтой, предназначенная для обработки сообщений об ошибках в продукте, доставке их исполнителям и контроля за исполнением. Интеграция с PVCS Version Manager дает возможность связывать с сообщениями те или иные компоненты проекта. Отчетные возможности PVCS Tracker включают множество разновидностей графиков и диаграмм, отражающих состояние проекта и процесса его отладки, срезы по различным компонентам проекта, разработчикам и тестировщикам. С их помощью можно наглядно показать текущее состояние работы над проектом и ее временные тенденции.

Персонал, работающий с PVCS Tracker делится на пять групп в зависимости от их обязанностей: пользователи, разработчики, группа тестирования и контроля качества, группа технической поддержки и сопровождения, управленческий персонал. Этим пяти группам персонала соответствуют пять предопределенных групп PVCS Tracker:

1. пользователи (Submitters) - имеют ограниченные права на внесение замечаний и сообщений об ошибках в базу данных PVCS Tracker;
2. разработчики (Development Engineers) - имеют право производить основные операции с требованиями и замечаниями в базе данных PVCS Tracker. Если разработчики делятся на подгруппы, то для каждой подгруппы могут быть заданы отдельные списки прав доступа;
3. тестировщики (Quality Engineers) - имеют право производить основные операции с требованиями и замечаниями;
4. сопровождение (Support Engineers) - имеют право вносить любые замечания, требования и рекомендации в базу данных, но не имеют прав по распределению работ и изменению их приоритетности и сроков исполнения;
5. руководители (Managers) - имеют право распределять работы между исполнителями и принимать решения об их надлежащем исполнении.

Руководителям разных групп могут заданы различные права доступа к базе данных PVCS Tracker.

В дополнение к этим пяти предопределенным группам, существует группа администратора базы данных и 11 дополнительных групп, которые могут быть настроены в соответствии со специфическими должностными обязанностями сотрудников, использующих PVCS Tracker.

Требование или замечание поступающее в PVCS Tracker проходит четыре этапа обработки:

1. регистрация - внесение замечания в базу данных;
1. распределение – назначение ответственного исполнителя и сроков исполнения;

2. исполнение – устранение замечания, которое в свою очередь может вызвать дополнительные замечания или требования на дополнительные работы;

3. приемка – приемка работ и снятие их с контроля или направление на доработку.

Требования и замечания, поступающие в базу данных PVCS Tracker оформляются в виде специальной формы, которая может содержать до 18 полей выбора стандартных значений и до 12 произвольных текстовых строк. При разработке формы следует определить оптимальный набор информации, характерный для всех записей в базе данных.

Для получения содержательной информации о ходе разработки PVCS Tracker позволяет получать три типа статистических отчетов: частотные, тренды и диаграммы распределения.

Частотные отчеты содержат информацию о частоте поступающих замечаний за один час тестирования программного продукта. Однако универсального частотного отчета не существует, т.к. на оценку качества влияют тип методов тестирования, серьезность выявленных ошибок и значение дефектных модулей для функционирования всей системы. Малое число фатальных ошибок, приводящих к полной остановке разработки, хуже большого числа замечаний к внешнему виду интерфейса пользователя. Следовательно, частотные отчеты должны быть настроены на выявление какого-либо конкретного аспекта качества для того, чтобы их можно было использовать для прогнозирования окончания работ над проектом.

Тренды содержат информацию об изменениях того или иного показателя во времени и характеризуют стабильность и непрерывность процесса разработки. Они позволяют ответить на вопросы:

1. успевает ли группа разработчиков справляться с поступающими замечаниями;
2. улучшается ли качество программного продукта и какова динамика этого процесса;
3. как повлияло то или иное решение (увеличение числа разработчиков, введение скользящего графика, внедрение нового метода тестирования) на работу группы и т.п.

Диаграммы распределения – наиболее разнообразные и полезные для осуществления оперативного руководства формы отчетов. Они позволяют ответить на вопросы: какой метод тестирования более эффективен, какие модули вызывают наибольшее число нареканий, кто из разработчиков лучше справляется с конкретным типом заданий, нет ли перекоса в распределении работ между исполнителями, нет ли модулей, тестированию которых было уделено недостаточно внимания и т.д.

PVCS Tracker предназначен для использования в рабочих группах, объединенных в общую сеть. В этом случае центральная база или проект PVCS Tracker находится на общедоступном сервере сети, доступ к которому реализуется посредством ODBC-драйверов, входящих в состав PVCS Tracker. Главной особенностью PVCS Tracker по сравнению с обычным приложением СУБД является его способность автоматически уведомлять пользователя о поступлении интересующей его или относящейся к его компетенции информации и гибкая система распределения полномочий внутри рабочей группы. При необходимости PVCS Tracker может использовать для уведомления удаленных членов группы электронную почту.

PVCS Tracker поддерживает групповую работу в локальных сетях и взаимодействует с СУБД dBase, ORACLE, SQL Server и SYBASE посредством ODBC.

PVCS Tracker может быть интегрирован с любой системой электронной почты, поддерживающей стандарты VIM, MAPI или SMTP.

PVCS Version Manager и PVCS Tracker окружены вспомогательными компонентами: PVCS Configuration Builder и PVCS Notify.

PVCS Configuration Builder предназначен для сборки окончательного продукта из компонент проекта. PVCS Configuration Builder позволяет описывать процесс сборки как на стандартном языке MAKE, так и на собственном внутреннем языке, имеющем существенно большие возможности. PVCS Configuration Builder позволяет осуществлять сборку программного продукта на основании файлов, хранящихся в репозитории PVCS Version Manager.

Обычная процедура сборки программного продукта с помощью PVCS Configuration Builder состоит из трех шагов:

1. строится файл зависимостей между исходными модулями;
2. в полученный файл вносятся изменения с целью его настройки и оптимизации;
3. осуществляется сборка программного продукта из исходных модулей.

Результатом работы PVCS Configuration Builder является специальный файл, описывающий оптимальный алгоритм сборки программного продукта, построенный на основе анализа дерева зависимостей между исходными модулями.

PVCS Notify обеспечивает автоматическую рассылку сообщений об ошибках из базы данных пакета PVCS Tracker по рабочим станциям назначения. При этом используется офисная система электронной почты cc:Mail или Microsoft Mail. PVCS Notify расширяет возможности PVCS Tracker и используется только совместно с ним.

PVCS Notify настраивается из среды PVCS Tracker. Настройка включает в себя определение интервала времени, через который PVCS Notify проверяет содержимое базы данных, определение критериев отбора записей для рассылки уведомлений, определение списков адресов для рассылки. После настройки PVCS Notify начинает работу в автономном режиме, автоматически рассылая уведомления об изменениях в базе данных PVCS Tracker.

PVCS Notify предназначен для использования в больших рабочих группах, часть членов которых хотя и доступна только через средства электронной почты, однако должна иметь оперативную информацию о требованиях на изменение программного продукта, замечаниях, ошибках, ходе и результатах его тестирования.

Результатом работы PVCS Notify являются оформленные в соответствии с одним из стандартов почтовые сообщения, готовые для рассылки посредством системы электронной почты.

Порядок выполнения лабораторной работы:

1. Обосновать выбор того или иного средства проектирования.

Лабораторная работа №20. Управление проектами ИС.

Цель: получить представление об анализе и оценке производительности информационных систем.

Краткие теоретические сведения

Задачи календарного планирования

Центральное место в управлении сроками проекта занимают задачи календарного планирования – процесса составления и корректировки расписания, в котором работы, выполняемые различными организациями, увязываются во времени между собой и с возможностями их обеспечения различными видами материально-технических и трудовых ресурсов. При увязке должно быть обеспечено соблюдение заданных ограничений, оптимальное (по принятому критерию) распределение ресурсов. Результатом процесса является создание календарных планов.

Календарные планы - это расписания и графики работ, выполняемых различными участниками, которые увязывают эти работы между собой по времени и возможностям обеспечения различными ресурсами. Типы календарных планов выбираются в зависимости от целей планирования и особенностей проекта.

В простейшем случае параметры календарного плана составляют даты начала и окончания каждой работы, их продолжительность и необходимые ресурсы. При анализе календарных планов определяют также резерв времени (величина возможного отклонения продолжительности для каждой работы, которая не повлияет на завершение проекта в срок). В большинстве сложных календарных планов существует до 6 вариантов моментов начала, окончания, продолжительности работ и резервов времени. Это - ранние, поздние, базовые, плановые и фактические даты, реальный и свободный резерв времени.

Методы расчета сетевых моделей позволяют вычислять только ранние и поздние даты. Базовые и текущие плановые даты необходимо выбирать с учетом других факторов. Существует три варианта выбора:

♣ календарный план по ранним началам (жестко слева): используется для стимулирования исполнителей проекта;

♣ календарный план по поздним окончаниям (жестко справа): используется для представления выполнения проекта в лучшем свете для потребителя;

♣ календарный план между ними: делается или для сглаживания потребляемых ресурсов или для показа заказчику наиболее вероятного исхода.

Ключевым понятием при составлении календарного плана является продолжительность работы. Продолжительность – это время выполнения работы. Обычно в детерминированных планах продолжительность работы считается неизменной. В действительности она зависит от внешних факторов и является случайной величиной, задается законом распределения (или плотностью распределения). Часто продолжительность меняется из-за изменения количества трудовых ресурсов на этой работе. Перед началом составления плана для каждой работы известна ее оценочная продолжительность. После начала работы, но до ее окончания можно вычислить оставшуюся продолжительность. Она может быть равна плановой продолжительности минус время, прошедшее с момента начала работы, или можно переоценить оставшуюся продолжительность на основании знаний, полученных при выполнении работы на данный момент времени. Как только работа будет закончена, можно зафиксировать фактическую продолжительность. Фактическую продолжительность полезно знать, так как, сравнивая ее с плановой, можно вычислить отклонения от плана, что используется для контроля процесса выполнения работ и вычисления тенденции.

Сетевой анализ и календарное планирование проектов

При анализе работы менеджеров проекта целесообразно различают три стороны дела: составление плана, составление графика, управление.

Составление графиков по проекту - дело сложное, поэтому в помощь менеджерам проектов было разработано много вспомогательных средств. Два из них – сетевой график и диаграмма Гантта. Они обычно реализованы в разнообразных пакетах программного обеспечения для управления проектами, такие пакеты широко представлены на рынке. Благодаря этим средствам резко ускоряется обработка данных и можно быстро вносить поправки в графики проектов. Кроме того, имеется возможность проанализировать различные варианты отклонений он номинала. Как и при использовании любой другой машинной программы, результат работы такого пакета будет не более полноценным, чем исходные данные. Поэтому важно, чтобы каждый пользователь таких программ понимал заложенные в них принципы.

Сетевой анализ – это метод планирования работ проектного характера, т.е. работ, операции в которых, как правило, не повторяются. Методы сетевого анализа позволяют осуществить анализ проекта, который включает в себя большое число взаимосвязанных операций. Мы можем определить вероятную продолжительность выполнения работ, их стоимость, возможные размеры экономии времени или денежных средств, а также то, выполнение каких операций нельзя отсрочить, не задержав при этом срок выполнения проекта в целом. Немаловажной является и проблема обеспечения ресурсами. Методы сетевого анализа могут быть использованы при составлении календарного плана выполнения операций, удовлетворяющего существующим ограничениям на обеспечение ресурсами.

Анализ любого проекта осуществляется в три этапа:

1. Расчленение проекта на ряд отдельных работ (или операций), из которых затем составляется логическая схема.

2. Оценка продолжительности выполнения каждой операции; составление календарного плана выполнения проекта и выделение работ, которые определяют завершение выполнения проекта в целом.

3. Оценка потребностей каждой операции в ресурсах; пересмотр плана выполнения операций с учетом обеспечения ресурсами либо перераспределение денежных или других ресурсов, которое улучшает план.

Сетевая диаграмма (сеть, граф сети, PERT диаграмма) – графическое отображение работ проекта и их взаимосвязей. В планировании и управлении проектами под термином сеть понимается полный комплекс работ и вех проекта с установленными между ними зависимостями. Сетевые диаграммы отображают сетевую модель в графическом виде как множество вершин, соответствующих работам, связанных линиями, представляющими взаимосвязи между работами. Этот граф, называемый сетью типа вершина – работа или диаграммой предшествования, является наиболее распространенным представлением сети на сегодняшний день. Сетевая диаграмма не является блок-схемой в том смысле, в котором это средство используется для моделирования деловых процессов. Принципиальным отличием от блок- схемы является то, что сетевая диаграмма моделирует только логические зависимости между элементарными работами. Она не отображает входы, процессы и выходы, и не допускает повторяющихся циклов или петель.

Методы сетевого планирования – методы, основная цель которых заключается в том, чтобы сократить до минимума продолжительность проекта, основываются на разработанных практически одновременно и независимо методе критического пути (МКП) и методе оценки и пересмотра планов PERT.

Первым шагом в анализе любого проекта является составление списка входящих в него операций. Детали такого списка зависят от специфики конкретного проекта. Тем не менее, во всех случаях необходимо выделить непосредственно предшествующую операцию или операции. Непосредственно предшествующими называются операции, выполнение которых должно быть закончено прежде, чем может начаться данная операция. Например, при постройке дома крыша не может быть построена до того момента, пока не закончится возведение стен. После того как составлен список, логическая последовательность выполнения операций может быть проиллюстрирована с помощью графа.

После того как проведена идентификация операций (работ), можно оценить их продолжительность. На основе продолжительности выполнения каждой операции и руководствуясь логической схемой, можно найти время выполнения проекта в целом. На данном этапе предполагается, что продолжительность выполнения каждой операции является фиксированной величиной, не испытывающей влияния неопределенности. В каждом графе существует несколько возможных путей. Общее время, необходимое для того, чтобы пройти какой -либо путь, есть сумма выполнения всех операций (работ), принадлежащих данному пути. Продолжительность выполнения всего проекта занимает наибольшее время. Более длительные операции называются критическими.

Критический путь – максимальный по продолжительности полный путь в сети (в сетевой модели). Работы, лежащие на этом пути, называются критическими работами. Длительность критического пути определяет наименьшую общую продолжительность работ по проекту в целом. Как правило, критические работы составляют небольшую часть всех работ сети, но именно они определяют продолжительность выполнения комплекса в целом. Длительность выполнения всего проекта в целом может быть сокращена за счет сокращения длительности задач, лежащих на критическом пути. Соответственно, любая задержка выполнения задач критического пути повлечет увеличение длительности проекта.

Существуют также работы с очень маленькими резервами времени. Они являются субкритическими и на них нужно обращать столько же внимания, сколько и на критические работы.

1. каждом графе найдется, по крайней мере, один критический путь. Для того чтобы найти общую продолжительность выполнения проекта, нужно определить продолжительность критического пути. В большинстве графов идентифицировать все идущие сквозь граф пути, чтобы выявить среди них тот, который занимает наибольшее время, достаточно трудно. Для определения критического пути используется метод критического пути.

Метод критического пути (МКП) – является основным математическим средством для вычисления ранних и поздних начал и окончаний работ и резервов времени.

Существует два возможных метода, позволяющих отследить движение времени в графе:

2. Определение для каждой операции наиболее ранних сроков начала и окончания ее выполнения.

3. Определение для каждого события наиболее раннего срока его наступления.

Второй метод применим только в случае стрелочных графов.

Любые замедления на критическом пути приведут к задержке срока выполнения всего проекта. Между тем для некритических путей можно допустить некоторые задержки при выполнении составляющих их операций или пересмотреть график их выполнения. Запас времени, который существует в схеме проекта, называется резервом времени.

Различают несколько видов резервов времени, возникающих под влиянием различных воздействий, которые оказывает запас времени на схему выполнения проекта.

Общим резервом называется количество времени, на которое можно увеличить продолжительность операции в результате продления срока ее выполнения или пересмотра плана, не влияющего на продолжительность выполнения проекта в целом.

Свободным резервом времени называется количество времени, на которое можно увеличить продолжительность операции в результате продления срока ее выполнения или пересмотра плана, не оказывающего воздействия на наиболее ранний срок выполнения любой последующей операции.

Иногда используют третий вид, так называемый независимый резерв времени. Он не оказывает никакого влияния на предшествующие или последующие операции.

Для любой операции (работы)

Общий резерв времени = LET окончания - EET начала - Продолжительность,

а также

Свободный резерв времени = EET окончания - EET начала – Продолжительность

и

Независимый резерв времени = EET окончания - LET начала - Продолжительность.

Иногда бывает полезно изобразить на графе имеющийся в наличии резерв времени, особенно если план выполнения операций необходимо пересмотреть. В этом случае одним из возможных методов является график (диаграмма) Гантта.

Сетевой граф отражает логическую последовательность выполнения операций, входящих в проект. До сих пор при анализе мы не принимали во внимание и не рассматривали конкретно какие-либо ограничения на обеспечение ресурсами. Исходный календарный план выполнения операций составлялся при условии, что все необходимые ресурсы имеются в достаточном количестве. Однако такая ситуация имеет место далеко не всегда, а если это так, то использование ресурсов в соответствии с потребностями, указанными в календарном плане, может оказаться неэкономичным.

Ограничения на ресурсы. При планировании необходимо учитывать неизбежную ограниченность ресурсов по нескольким факторам.

Оборудование. У некоторых машин могут быть не такие технологические возможности для конкретных операций, как у других. Могут ставиться дополнительные требования по их обслуживанию (технической эксплуатации). Может потребоваться какое-то время на переоснащение и переналадку каких-нибудь станков для конкретных операций.

Люди. Люди могут отсутствовать на работе не только в связи с плановыми отпусками и государственными праздниками, но и, например, в связи с прохождением курсов повышения квалификации. Очень бы хотелось иметь универсалов, способных полностью заменять друг друга, но такие – большая редкость.

Материалы. Могут быть большие задержки в поставках некоторых материалов; возможны и другие проблемы.

Метод составления календарного плана с учетом обеспечения ресурсами зависит от конкретных целей лиц, осуществляющих контроль за ходом выполнения проекта. Например, вопросом первостепенной важности может оказаться завершение проекта к определенному

сроку безотносительно к затратам ресурсов - такие планы ограничены во времени. И, наоборот, в условиях ограниченности в денежных средствах на выполнение проекта отводится определенное количество ресурсов, тогда как срок выполнения не принимается в расчет – такие планы ограничены по ресурсам. В данном контексте к ресурсам можно отнести рабочую силу, оборудование, сырье, денежные средства, производственные площади и т.д.

Перед тем как приступить к выполнению проекта, управляющий должен четко сформулировать критерий, в соответствии с которым будет осуществляться распределение ресурсов. В качестве такого критерия можно выбрать:

1. Максимальное использование ресурсов. Оценить использование ресурсов можно через соответствующий коэффициент: Коэффициент использования = Общее количество используемых ресурсов / Общее количество наличных ресурсов.
2. Минимизацию максимальных потребностей в ресурсах.
3. Минимизацию максимальных изменений потребностей в ресурсах. Кроме названных, существует и множество других критериев.

Существует также и множество возможных методов решения проблемы распределения ресурсов, таких, как например, эвристические методы, методы линейного и других видов математического программирования, реализуемых обычно в разнообразных программных продуктах. Для практического использования и понимания принципов работы стандартных программ по управлению проектами рассмотрим один из простейших алгоритмов, в котором используются графики ресурсов и «метод проб и ошибок».

Графики ресурсов. Если общая потребность в некотором ресурсе определяется на основе постоянных интервалов, например, за один день или за одну неделю, то можно построить график ресурса.

Ресурсы, требуемые для осуществления каждой работы, складываются по всем работам, выполняемым одновременно, в предположении, что каждая работа начинается в наиболее ранний срок ее выполнения. Как следует из приведенного графика, иногда потребности в рабочей силе превышают ее наличие, но в то же время общее число требуемых человеко-часов не превосходит их наличного количества.

Если потребность в ресурсе превысила его лимит, необходимо либо вложить в проект дополнительное количество ресурса, либо пересмотреть календарный план выполнения операций. Иногда в таких ситуациях необходимо задержать срок выполнения проекта. Несмотря на то, что некоторые операции проекта не имеют явной логической последовательной взаимосвязи, одновременное их выполнение часто оказывается невозможным вследствие ограничений на ресурсы. Это ограничение можно отразить на графике ресурса, если провести линию, соответствующую наличному количеству данного ресурса. Такой прием позволит не планировать выполнение определенных операций на один и тот же день.

Порядок выполнения лабораторной работы:

1. Составить сетевой график разработки проекта.
2. Составить календарный план.

Лабораторная работа №21. Примеры управления проектами ИС.

Цель: рассмотреть примеры управления проектами.

Краткие теоретические сведения

Управление стоимостью проекта

Главной задачей управления стоимостью (затратами) проекта является стоимостной анализ. Потребность в стоимостном анализе проекта возникает на всех его стадиях. Для этого привлекается система технико-экономических показателей и соответствующие методические правила. Стоимостной анализ проекта предполагает сопоставление слагаемых затрат и результатов. Практическая проблема состоит в том, что отнести к затратам и результатам и как вычислить.

Методы стоимостного анализа проектов делят на статические и динамические. В первом случае показатели расходов и доходов за разные интервалы времени приводят к суммарному показателю, во втором – анализируют денежные временные потоки.

Стоимостной анализ проекта занимает ведущую роль среди остальных видов проектного анализа: технического, коммерческого, экологического, организационного, социального, экономического, так как служит базой для принятия управленческих решений на всех фазах жизненного цикла проекта.

1. процессам управления стоимостью относятся: 1. Планирование ресурсов.
2. Оценка затрат.
3. Составление бюджета.
4. Контроль затрат.

1. Планирование ресурсов.

Данный процесс включает определение перечня ресурсов (людей, материалов, оборудования), требуемых для выполнения работ проекта, а также их количества (выраженного в физических единицах для расходных материалов и чел/часах для людских ресурсов).

Входные материалы для планирования ресурсов

1. Структура работ.
2. Архивная и статистическая информация.
3. Информация о том, какие ресурсы и в каком количестве требовались в прошлом для схожих проектов.

1. Свод содержания проекта. Пул ресурсов – это совокупность людских и прочих ресурсов, имеющихся в распоряжении компании, которые можно использовать для обеспечения работ данного проекта.
2. Административные процедуры. На этапе планирования ресурсов должны приниматься во внимания такие бизнес-процедуры компании, как наем персонала или материально-техническое снабжение.
3. Инструменты и методы, используемые при планировании ресурсов
4. Заключение экспертов. В качестве экспертов могут привлекаться как служащие компании, так и внешние консультанты.
5. Идентификация альтернатив. Многие ресурсы в той или иной степени являются взаимозаменяемыми. Поэтому стоит предусмотреть запасные варианты при планировании критически важных для проекта ресурсов.

Выходные материалы процесса планирования ресурсов

1. Потребности в ресурсах. Выходом процесса планирования ресурсов является информация о том, сколько единиц ресурсов и каких видов требуется для выполнения работ, соответствующих каждому из узлов WBS. В дальнейшем эти ресурсы будут предоставлены проекту в результате найма персонала или в результате приобретения требуемых продуктов и услуг на соответствующем рынке.
2. Данный процесс включает в себя приблизительную оценку затрат на ресурсы, которые потребуются для выполнения проекта, и распределение этих затрат во времени. Следует отличать оценку затрат от определения контрактной цены. Оценка затрат – это оценка издержек, которых потребует от компании-исполнителя создание продукта или предоставление услуги заказчику. Определение цены контракта – это бизнес-решение о том, какой счет выставить заказчику. Здесь принимается во внимание не только уровень собственных издержек, но и коньюктура рынка, уникальность продукта или услуги и т.д.

Входные материалы для процесса оценки затрат

1. WBS (Структура работ). Должна быть проведена оценка затрат для всех работ, определенных на предыдущих шагах, т.е. для всех узлов WBS.
2. Потребности в ресурсах.
3. Тарифы и цены за единицу каждого ресурса. Стоимость одного чел./часа конкретного специалиста, стоимость одной тонны цемента и т.д.

4. Оценки продолжительностей работ. Если к работе приписан некоторый ресурс на все время ее выполнения, то чем больше будет продолжительность данной работы, тем больше будет нагрузка на ресурс, а значит и связанные с этим затраты.

5. Архивная и статистическая информация. Существуют специализированные базы данных по затратам на проекты разных типов, оформленные в виде коммерческих продуктов, а также разнообразная нормативная документация.

6. План счетов. Затраты по проекту отражаются на счете в плане счетов, принятом в организации.

Инструменты и методы для процесса оценки затрат

Оценки по проектам-аналогам. Аналоговые оценки, иначе называемые оценками сверху вниз, предполагают, что в качестве базиса используются фактические затраты по уже выполненному схожему проекту. Такой метод часто используется для оценки общих затрат по проекту в условиях недостатка подробной информации (например, на ранних стадиях проекта).

Параметрическое моделирование. Этот метод предполагает построение математической модели, отражающей некоторые характеристики проекта. Такие модели могут быть как сравнительно простыми, так и чрезвычайно сложными. Точность метода также варьируется в широких пределах. Наилучшие результаты могут быть получены, если модель была построена на основе достоверных архивных данных, параметры проекта поддаются количественной оценке и модель является масштабируемой (т.е. одинаково хорошо работает как для больших, так и для малых проектов).

Оценки снизу вверх. Эта техника предполагает оценку затрат сначала для промежуточных продуктов самого нижнего уровня, а затем вычисление итоговых затрат для продуктов более высоких уровней.

Программные средства. Задача оценки затрат по вовлеченным ресурсам традиционно подлежит автоматизации, и соответствующий блок реализован в той или иной степени во всех программных продуктах по управлению проектами.

Выходные материалы для процесса оценки затрат

1. Оценки затрат. Рассчитываются по каждой работе, а потом суммируются в соответствии с узлами WBS.

2. Вспомогательные материалы. Сюда входит описание принятых допущений, оценки точности выполненных расчетов (например, 10.000$ плюс/минус 10%) и т.д. План управления затратами. План, определяющий порядок действий на случай обнаружения в ходе выполнения проекта отклонений по затратам и регламентирующий внесение изменений в бюджет проекта.

Составление бюджета проекта

Под бюджетом проекта мы будем понимать общую сумму затрат по проекту и распределение этих затрат во времени на протяжении всего жизненного цикла проекта с разбивкой по блокам работ (узлам WBS).

Входные материалы для составления бюджета проекта

1. Оценки затрат.

2. График проекта. График проекта определяет сроки начала/завершения каждой из работ проекта, а значит и отчетный период (неделю, месяц, квартал), на который следует отнести затраты по выполнению этой работы. Как правило, делается допущение, что затраты распределяются равномерно на весь период выполнения работы. Однако следует учитывать как вариант 100% предоплаты, так и вариант оплаты после завершения работ.

Управление рисками проекта

Причиной возникновения рисков являются неопределенности, существующие в каждом проекте. Риски могут быть “известные”- те, которые определены, оценены, для которых возможно планирование. Риски “неизвестные” – те, которые не идентифицированы и не могут быть спрогнозированы. Хотя специфические риски и условия их возникновения не определены, менеджеры проекта знают, исходя из прошлого опыта, что большую часть рисков можно предвидеть.

Реализуя проекты, имеющие высокую степень неопределенности в таких элементах, как цели и технологии их достижения многие компании уделяют внимание разработке и применению корпоративных методов управления рисками. Данные методы учитывают как специфику проектов, так и корпоративных методов управления.

Американский Институт управления проектами (PMI), разрабатывающий и публикующий стандарты в области управления проектами, значительно переработал разделы, регламентирующие процедуры управления рисками. В новой версии PMBOK описаны шесть процедур управления рисками.

Управление рисками – это процессы, связанные с идентификацией, анализом рисков и принятием решений, которые включают максимизацию положительных и минимизацию отрицательных последствий наступления рисковых событий. Процесс управления рисками проекта обычно включает выполнение следующих процедур:

1. Планирование управления рисками – выбор подходов и планирование деятельности по управлению рисками проекта.
2. Идентификация рисков – определение рисков, способных повлиять на проект, и документирование их характеристик.
3. Качественная оценка рисков – качественный анализ рисков и условий их возникновения с целью определения их влияния на успех проекта.
4. Количественная оценка – количественный анализ вероятности возникновения и влияния последствий рисков на проект.
5. Планирование реагирования на риски – определение процедур и методов по ослаблению отрицательных последствий рисковых событий и использованию возможных преимуществ.
6. Мониторинг и контроль рисков – мониторинг рисков, определение остающихся рисков, выполнение плана управления рисками проекта и оценка эффективности действий по минимизации рисков.

Все эти процедуры взаимодействуют друг с другом, а также с другими процедурами. Каждая процедура выполняется, по крайней мере, один раз в каждом проекте. Несмотря на то, что процедуры, представленные здесь, рассматриваются как дискретные элементы с четко определенными характеристиками, на практике они могут частично совпадать и взаимодействовать.

Планирование управления рисками – процесс принятия решений по применению и планированию управления рисками для конкретного проекта. Этот процесс может включать в себя решения по организации, кадровому обеспечению процедур управления рисками проекта, выбор предпочтительной методологии, источников данных для идентификации риска, временной интервал для анализа ситуации. Важно спланировать управление рисками, адекватное как уровню и типу риска, так и важности проекта для организации.

Идентификация рисков определяет, какие риски способны повлиять на проект, и документирует характеристики этих рисков. Идентификация рисков не будет эффективной, если она не будет проводиться регулярно на протяжении реализации проекта.

Идентификация рисков должна привлекать как можно больше участников: менеджеров проекта, заказчиков, пользователей, независимых специалистов.

Идентификация рисков – итерационный процесс. Вначале идентификация рисков может быть выполнена частью менеджеров проекта или группой аналитиков рисков. Далее идентификацией может заниматься основная группа менеджеров проекта. Для формирования объективной оценки в завершающей стадии процесса могут участвовать независимые специалисты. Возможное реагирование может быть определено в течение процесса идентификации рисков.

Качественная оценка рисков – процесс представления качественного анализа идентификации рисков и определения рисков, требующих быстрого реагирования. Такая оценка рисков определяет степень важности риска и выбирает способ реагирования. Доступность сопровождающей информации помогает легче расставить приоритеты для разных категорий рисков. Качественная оценка рисков - это оценка условий возникновения рисков и определение их воздействия на проект стандартными методами и средствами.

Использование этих средств помогает частично избежать неопределенности, которые часто встречаются в проекте. В течение жизненного цикла проекта должна происходить постоянная переоценка рисков.

Количественная оценка рисков определяет вероятность возникновения рисков и влияние последствий рисков на проект, что помогает группе управления проектами верно принимать решения и избегать неопределенностей. Количественная оценка рисков позволяет определять:

Вероятность достижения конечной цели проекта

1. Степень воздействия риска на проект и объемы непредвиденных затрат и материалов, которые могут понадобиться.
2. Риски, требующие скорейшего реагирования и большего внимания, а также влияние их последствий на проект.
3. Фактические затраты, предполагаемые сроки окончания. Количественная оценка рисков часто сопровождает качественную

оценку и также требует процесс идентификации рисков. Количественная и качественная оценка рисков могут использоваться по отдельности или вместе, в зависимости от располагаемого времени и бюджета, необходимости в количественной или качественной оценке рисков.

Планирование реагирования на риски – это разработка методов и технологий снижения отрицательного воздействия рисков на проект. Берет на себя ответственность за эффективность защиты проекта от воздействия на него рисков. Планирование включает в себя идентификацию и распределение каждого риска по категориям. Эффективность разработки реагирования прямо определит, будут ли последствия воздействия риска на проект положительными или отрицательными.

Стратегия планирования реагирования должна соответствовать типам рисков, рентабельности ресурсов и временным параметрам. Вопросы, обсуждаемые во время встреч, должны быть адекватны задачам на каждой стадии проекта, и согласованы со всеми членами группы по управлению проектом. Обычно требуются несколько вариантов стратегий реагирования на риски.

Мониторинг и контроль следят за идентификацией рисков, определяют остаточные риски, обеспечивают выполнение плана рисков и оценивают его эффективность с учетом понижения риска. Показатели рисков, связанные с осуществлением условий выполнения плана, фиксируются. Мониторинг и контроль сопровождает процесс внедрения проекта в жизнь.

Качественный контроль выполнения проекта предоставляет информацию, помогающую принимать эффективные решения для предотвращения возникновения рисков. Для предоставления полной информации о выполнении проекта необходимо взаимодействие между всеми менеджерами проекта.

Целью мониторинга и контроля является выяснить, было ли:

1. Система реагирования на риски внедрена в соответствии с планом.
2. Реагирование достаточно эффективно или необходимы изменения.
3. Риски изменились по сравнению с предыдущим значением.
4. Наступление влияния рисков.
5. Необходимые меры приняты.

Воздействие рисков оказалось запланированным или явилось случайным результатом.

Контроль может повлечь за собой выбор альтернативных стратегий, принятие корректив, перепланировку проекта для достижения базового плана. Между менеджерами проекта и группой риска должно быть постоянное взаимодействие, должны фиксироваться все изменения и явления. Отчеты по выполнению проекта должны формироваться регулярно.

Управления проектами в Microsoft Project 2003

Управление проектами – сложный и объемный процесс. Сейчас на рынке программного обеспечения предлагается множество пакетов для решения задач управления проектами.

Управление проектами – сложный, емкий процесс, включающий в себя планирование, распределение, управление временем, ресурсами, финансами и качеством. Время также можно отнести к ресурсам, но это уникальный ресурс. Вы можете найти деньги, нанять людей не составит большого труда, но вы нигде и ни за какие деньги не сможете купить дополнительное время. При управлении проектом нужно четко осознавать приоритеты данного проекта и вовремя принимать решения. Вы можете принять в качестве критерия управления проектом бюджет проекта, сроки выполнения или объемы используемых ресурсов. Если в процессе выполнения плана его будет необходимо корректировать, исходя из критерия управления проектом, вы увеличиваете срок выполнения проекта, или размер бюджета, или снижаете качество выполнения работ.

Разработка тщательно продуманного плана имеет решающее значение для его успешного выполнения. Такой план позволяет сэкономить время и деньги. Для более удобного управления проектами стали создаваться различные системы управления проектами.

Система управления проектами представляет собой специализированные программные средства, для которых характерно: наличие основных элементов – работ, ресурсов, назначений ресурсов работам; формирование модели реализации проекта таким образом, что все работы в проекте отражают технологическую последовательность их выполнения с учетом иерархической структуры работ проекта; использование метода критического пути – основа методов сетевого планирования и управления; использование линейной диаграммы (обычно линейные диаграммы называют диаграммой Гантта) и др.

Это позволяет использовать системы управления проектами для достижения таких целей, как прогноз технико - экономических показателей проектов, выявление проблем и анализ способов их разрешения, обоснование управляющих решений и документирование прогнозов и результатов работ при помощи экранных форм и отчетов. К таким системам управления проектами относится MS Project.

Программное обеспечение MS Project автоматизирует многие из этих процедур. С MS Project можно:

1. Произвести предварительное планирование элементов вашего проекта, что позволяет более точно оценить время и ресурсы, необходимые для выполнения проекта.
2. Контроль выполнения. Контролируя выполнение, вы можете точно оценить перспективу выполнения проекта.
3. Выявление конфликтов. Выявляя конфликты ресурсов на ранних стадиях, вы можете применить метод сценариев и разрешить существующие в проекте конфликты.
4. Внесение изменений. Вы можете вносить изменения в продолжительность ресурсов, задачи и затраты, и автоматически произойдет настройка и оценка всего проекта в целом.
5. Выполнение профессиональных отчетов. Вы можете создавать отчеты о статусе вашего проекта, помогать членам команды верно расположить приоритеты задач и помочь управлению принять квалифицированное решение.

С улучшенным взаимодействием внутри рабочей группы с помощью корпоративной сети или с помощью электронной почты, MS Project также делает связь и сотрудничество среди членов команды намного более легкой и более производительной. Вам не придется перемещаться из кабинета в кабинет и доводить информацию по проекту до каждого участника проекта, будь то изменения или дополнения по проекту.

MS Project – это один из наиболее популярных инструментов, которым пользуются миллионы пользователей. Он позволяет в значительной мере сократить затраты времени на планирование и контроль работы коллектива. С его помощью можно выполнить расчеты временных затрат, определить стоимость разработки, выявить критический путь в сетевом графике, определить загрузку сотрудников, получить множество отчетов, которые связаны с выполняемым проектом. Вы достаточно быстро сможете "проиграть" несколько вариантов

выполнения работ и оценить их эффективность и стоимость. Правда, при этом вы можете получить и такие неприятные результаты: в проекте заложены нереальные сроки, или его бюджет слишком мал. Встроенные в Microsoft Project средства оптимизации позволят вам найти пути решения проблемы. Вы сможете отмечать процесс выполнения проекта и корректировать сроки в соответствии с жизненными реалиями. Кстати, по окончании проекта можно сравнить план и реальность с тем, чтобы учесть это при своей дальнейшей работе.

Возможности MS Project. Во-первых, легкий доступ к информации и коллективная работа в сети Интернет. Исходя из этого, вы можете сохранить любой файл Project (или любой документ Office) в формате HTML так, чтобы вы могли предоставить доступ к этому файлу любому, кто имеет Интернет браузер. Кроме того, вы можете использовать Microsoft Project 2000 для редактирования HTML-страниц, созданных в Project, без потери форматирования или функциональных возможностей. Во-вторых, чтобы уменьшить расходы на установку и развертывание, Microsoft создал Windows Installer, который позволяет легко настроить установку в вашей организации. Windows Installer позволяет определить особенности инсталляции, так называемая "установка по требованию". Команда появляется в меню, но, когда пользователь выбирает команду, появляется диалог, сообщающий, что данная возможность в настоящее время не установлена и связана с выбором особенностей установки. Как правило, этот тип установки исключает шаблоны.

Порядок выполнения лабораторной работы:

1. Рассчитать стоимость разработки.

Лабораторная работа № 22. Разработка описания и анализ информационной системы.

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

Теоретические сведения

Общие сведения о проектировании информационных систем

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

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

Руководители проектов призваны спланировать все этапы разработки программного продукта. Они также должны контролировать ход выполнения работ и соблюдения всех требуемых стандартов. Постоянный контроль за ходом выполнения работ необходим для того, чтобы процесс разработки не выходил за временные и бюджетные ограничения. Хорошее управление не гарантирует успешного завершения проекта, но плохое управление обязательно приведет к его провалу. Это может выразиться в задержке сроков сдачи готового ПО, в превышении сметной стоимости проекта и в несоответствии готового ПО спецификации требований.

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

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.

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

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

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

Процесс управления разработкой программного обеспечения

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

1. Написание предложений по созданию ПО.
2. Планирование и составление графика работ по созданию ПО.
3. Оценивание стоимости проекта.
4. Подбор персонала.
5. Контроль за ходом выполнения работ.
6. 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 опыта.

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

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

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

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

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

Руководители проектов обычно обязаны сами подбирать исполнителей для своих проектов. В идеальном случае профессиональный уровень исполнителей должен соответствовать той работе, которую они будут выполнять в ходе реализации проекта. Однако во многих случаях руководители должны полагаться на команду разработчиков, которая далека от идеальной. Такая ситуация может быть вызвана следующими причинами:

Бюджет проекта не позволяет привлечь высококвалифицированный персонал. В таком случае за меньшую плату привлекаются менее квалифицированные специалисты.

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.

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.

Выделены следующие роли в группе по разработке ПО:

1. Руководитель – общее руководство проектом, написание документации, общение с заказчиком ПО
2. Системный аналитик – разработка требований (составление технического задания, проекта программного обеспечения)

3. Тестер – составление плана тестирования и аттестации готового ПО (продукта), составление сценария тестирования, базовый пример, проведение мероприятий по плану тестирования

4. Разработчик – моделирование компонент программного обеспечения, кодирование

5. Планирование проекта разработки программного обеспечения

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

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

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

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

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

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

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

Организация выполнения проекта. Описание способа подбора команды разработчиков и распределение обязанностей между членами команды.

Анализ рисков. Описание возможных проектных рисков, вероятности их проявления и стратегий, направленных на их уменьшение.

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.

Разбиение работ на этапы. Процесс реализации проекта разбивается на отдельные процессы, определяются этапы выполнения проекта, приводится описание результатов ("выходов") каждого этапа и контрольные отметки.

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.

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.

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

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

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

Критическим является вопрос, будет ли система соответствовать целям организации. Если система не соответствует этим целям, она не представляет никакой ценности для организации. В то же время многие организации разрабатывают системы, не соответствующие их целям, либо не совсем ясно понимая эти цели, либо под влиянием политических или общественных факторов.

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

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?

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:
4. Что произойдет с организацией, если система не будет введена в эксплуатацию?
5. Какие текущие проблемы существуют в организации и как новая система поможет их решить?
6. Каким образом система будет способствовать целям бизнеса?
7. Требует ли разработка системы технологии, которая до этого не использовалась в организации?
8. Результатом анализа должно явиться заключение о возможности реализации проекта.
9. Распределить роли в группе (руководитель проекта - разработчик, системный аналитик-разработчик, тестер-разработчик).

Заполнить разделы плана:

1. Введение
2. Организация выполнения проекта
3. Анализ рисков
4. Разделы должны содержать рекомендации относительно разработки системы, базовые предложения по объёму требуемого бюджета, числу разработчиков, времени и требуемому программному обеспечению.
5. Составить отчет о проделанной работе.

Содержание отчета

В отчете следует указать:

Цель работы

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

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

Анализ осуществимости (согласно требованиям к результатам выполнения лабораторного практикума п.2), указать возможные проблемы и пути их решения.

Роли участников группы разработки ПО.

Программно-аппаратные средства, используемые при выполнении работы.

Заключение (выводы)

Список используемой литературы

Лабораторная работа № 23. Разработка требований к информационной системе.

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

Теоретические сведения

Общие сведения о требованиях к информационным системам

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

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

Разработка требований

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

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

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

Рис. 1. Процесс разработки требований

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

Формирование и анализ требований

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

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

Рис. 2. Процесс формирования и анализа требований

Процесс формирования и анализа требований проходит через ряд этапов.

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

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

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

Опорные точки зрения

Подход с использованием различных опорных точек зрения к разработке требований признает различные (опорные) точки зрения на проблему и использует их в качестве основы построения и организации как процесса формирования требований, так и непосредственно самих требований.

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

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.

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.

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.:

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

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

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

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:

1. список всех товаров;
2. список товаров, имеющихся в наличии;
3. список товаров, количество которых необходимо пополнить;
4. 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)		Печать чека	Проверка статистики
Получение чека			Назначение цены		Доступ к каталогу	Переопределение 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.

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.

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.

Проверка на полноту. Спецификация требований должна содержать требования, которые определяют все системные функции и ограничения, налагаемые на систему.

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.

Обзор требований. Требования системно анализируются рецензентами.

Прототипирование. На этом этапе прототип системы демонстрируется конечным пользователям и заказчику. Они могут экспериментировать с этим прототипом, чтобы убедиться, что он отвечает их потребностям.

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.

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 программного обеспечения (пример см. Приложение Б). ТЗ должно содержать основные разделы, описанные в ГОСТ 34.602-89 (см. Приложение А).
6. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

Содержание отчета

В отчете следует указать:

1. Цель работы

2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основная часть (описание самой работы).
5. Заключение (выводы)
6. Список используемой литературы

Контрольные вопросы

1. Предложите, кто бы мог участвовать в формировании требований для университетской системы регистрации студентов. Объясните, почему почти неизбежно, что требования, сформулированные разными лицами, будут противоречивы.

2. Разрабатывается система ПО для автоматизации библиотечного каталога. Эта система будет содержать информацию относительно всех книг в библиотеке и будет полезна библиотечному персоналу, абонентам и читателям. Система должна иметь средства просмотра каталога, средства создания запросов и средства, позволяющие пользователям резервировать книги, находящиеся в данный момент на руках. Определите основные опорные точки зрения, которые необходимо учесть в спецификации системы, и покажите их взаимоотношения, используя диаграмму иерархии точек зрения.

3. Для трех точек зрения, определенных в системе библиотечного каталога, укажите сервисы и соответствующие данные, которые обеспечиваются этими точками зрения, и события, которые управляют этими сервисами.

4. Кто должен проводить обзор требований? Нарисуйте модель процесса обзора требований.

7. Ваша компания использует стандартный метод анализа требований. В процессе работы вы обнаружили, что этот метод не учитывает социальные факторы, важные для системы, которую вы анализируете. Ваш руководитель дал вам ясно понять, какому методу анализа нужно следовать. Обсудите, что вы должны делать в такой ситуации.

Лабораторная работа №24. Методология функционального моделирования.

Цель: Изучить методологии функционального моделирования IDEF0 и IDEF3.

Методические указания

Занятие направлено 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.

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:

1. принцип функциональной декомпозиции;
2. принцип ограничения сложности;
3. 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 декомпозици

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.

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 № 7 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:
3. с помощью методологии IDEF0 построить контекстную диаграмму;
4. с помощью методологии IDEF0 построить диаграмму 1-го уровня (A0) – модель окружения;
5. с помощью методологии IDEF3 декомпозировать функциональные блоки модели окружения на 1-2 уровня вглубь до потоков, связи с внешними системами и
6. на каждой диаграмме 2-го уровня должно быть не менее 4-х функциональных блоков;
7. на каждой диаграмме 3-го уровня и далее не менее 2-х функциональных блоков.
8. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

Содержание отчета

В отчете следует указать:

1. Цель работы
2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основную часть (описание самой работы), Заключение (выводы)
5. Список используемой литературы

Контрольные вопросы.

1. Перечислите основные объекты IDEF0, их описание и назначение.
2. Назовите базовые принципы моделирования в IDEF0.
3. В каких случаях целесообразно применять построение модели “как есть”, а в каких “как будет”?
4. Перечислите основные объекты IDEF3, их описание и назначение.
5. В чём смысл использования перекрёстков в IDEF3?
6. В чём отличия IDEF0 и IDEF3? Когда целесообразней использовать IDEF0, а когда IDEF3?

Лабораторная работа №25. Методология объектно-ориентированного моделирования.

Цель: Ознакомление с основными элементами определения программных систем с помощью языка UML.

Методические указания

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.

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:

1. является языком визуального моделирования, который обеспечивает разработку репрезентативных моделей для организации взаимодействия заказчика и разработчика ИС, различных групп разработчиков ИС;
2. содержит механизмы расширения и специализации базовых концепций языка.
3. 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.
4. 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:
5. строить модели на основе средств ядра, без использования механизмов расширения для большинства типовых приложений;
6. 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):
4. диаграммы взаимодействия (interaction diagrams):
5. диаграммы последовательности (sequence diagrams) и
6. кооперативные диаграммы (collaboration diagrams) – для моделирования процесса обмена сообщениями между объектами;
7. диаграммы состояний (statechart diagrams) – для моделирования поведения объектов системы при переходе из одного состояния в другое;
8. диаграммы деятельностей (activity diagrams) – для моделирования поведения системы в рамках различных вариантов использования, или моделирования деятельностей;
9. диаграммы реализации (implementation diagrams):
10. диаграммы компонентов (component diagrams) – для моделирования иерархии компонентов (подсистем) системы;
11. диаграммы развертывания (deployment diagrams) – для моделирования физической архитектуры системы.

Диаграммы вариантов использования

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

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

Рис.2. Вариант использования

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

Рис.3. Действующее лицо (актер)

Действующие лица делятся на три основных типа:

1. пользователи;
2. системы;
3. другие системы, взаимодействующие с данной;
4. 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

Лабораторная работа №26. Диаграммы взаимодействия, кооперации и классов в UML.

Цель: Ознакомление с основными элементами представления программных систем с помощью языка UML.

Диаграммы взаимодействия (interaction diagrams)

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

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

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

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

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

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

Диаграмма последовательности (sequence diagrams)

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

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

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

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

Рис. 7. Пример диаграммы последовательности

Диаграмма кооперации (collaboration diagram)

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

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

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

Рис. 8. Пример диаграммы кооперации

Диаграммы классов

Общие сведения

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

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

На диаграмме классов изображаются следующие элементы:

1. Пакет (package) - набор элементов модели, логически связанных между собой;
2. Класс (class) - описание общих свойств группы сходных объектов;
3. 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:

1. Boundary (граница);
2. Entity (сущность);
3. 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:

1. Операции реализации;
2. Операции управления;
3. Операции доступа;
4. 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:

Изучите диаграммы последовательности и кооперативные диаграммы. Большая часть сообщений на этих диаграммах является операциями реализации. Рефлексивные сообщения будут вспомогательными операциями.

Рассмотрите управляющие операции. Может потребоваться добавить конструкторы и деструкторы.

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.

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

Лабораторная работа №27. Проектирование и моделирование UML.

Цель: Ознакомление с основными элементами проектирования и моделирования программных систем с помощью языка UML.

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:
4. диаграмм классов, реализующих вариант использования;
5. диаграмм взаимодействия (диаграмм последовательности и кооперативных диаграмм), отражающих взаимодействие объектов в процессе реализации варианта использования.
6. Разделить классы по пакетам использую один из механизм разбиения.
7. Постройте диаграмму состояний для конкретных объектов информационной системы.
8. Построить отчёт, включающий все полученные уровни модели, описание функциональных блоков, потоков данных, хранилищ и внешних объектов.

Содержание отчета

В отчете следует указать:

1. Цель работы
2. Введение
3. Программно-аппаратные средства, используемые при выполнении работы.
4. Основную часть (описание самой работы)
5. Заключение (выводы)
6. Список используемой литературы

Контрольные вопросы

1. Предложите, кто бы мог участвовать в формировании требований для коледжской системы регистрации студентов. Объясните, почему почти неизбежно, что требования, сформулированные разными лицами, будут противоречивы.

2. Разрабатывается система ПО для автоматизации библиотечного каталога. Эта система будет содержать информацию относительно всех книг в библиотеке и будет полезна библиотечному персоналу, абонентам и читателям. Система должна иметь средства просмотра каталога, средства создания запросов и средства, позволяющие пользователям резервировать книги, находящиеся в данный момент на руках. Определите основные опорные точки зрения, которые необходимо учесть в спецификации системы, и покажите их взаимоотношения, используя диаграмму иерархии точек зрения.

3. Для трех точек зрения, определенных в системе библиотечного каталога, укажите сервисы и соответствующие данные, которые обеспечиваются этими точками зрения, и события, которые управляют этими сервисами.

4. Кто должен проводить обзор требований? Нарисуйте модель процесса обзора требований.

1. Ваша компания использует стандартный метод анализа требований. В процессе работы вы обнаружили, что этот метод не учитывает социальные факторы, важные для системы, которую вы анализируете. Ваш руководитель дал вам ясно понять, какому методу анализа нужно следовать. Обсудите, что вы должны делать в такой ситуации.

ПРИЛОЖЕНИЯ

Приложение 1. ***Пример отчета о предпроектном обследовании***

Отчёт
о предпроектном обследовании ЗАО «ХХХ»
для внедрения корпоративной информационной системы (КИС)
1С: «Институт типовых решений производства» (ИТРП)
2008 г.

СОДЕРЖАНИЕ

1. ЦЕЛЬ ПРОВЕДЕНИЯ ОБСЛЕДОВАНИЯ

1. процессе предпроектного обследования ЗАО «ХХХ» изучаются основные направления производственно-хозяйственной деятельности, организационная структура ЗАО «ХХХ». Определяются функции подразделений, существующие информационные взаимосвязи между подразделениями, внутренний и внешний документооборот.

На основе анализа указанной информации определяются требуемые учетные подсистемы (например, складской учет ГП), охватывающие несколько под-разделений, каждое из которых заинтересовано в оперативности и актуальности данных. Разрабатываются рекомендации по усовершенствованию документооборота, исключению дублирования информации. При проектировании комплексного решения ИТРП закладываются решения, реализующие принципы однократного ввода.

Изучается текущий уровень автоматизации: определяется перечень разработанных подсистем, состав автоматизированных рабочих мест и круг решаемых задач с целью определения функциональной полноты системы и автоматизации учетных функций. Разрабатываются предложения по требуемому составу вы-бранных подсистем КИС (Корпоративная Информационная Система), уточнению перечня задач, подлежащих автоматизации, и расширению состава автоматизированных рабочих мест с целью получения полной оперативной информации по бухгалтерскому, оперативному и управленческому учету производственно-хозяйственной деятельности ЗАО «ХХХ», обеспечивающей принятие верного управленческого решения в режиме реального времени.

Определяются используемые программное, информационное обеспечения и обследуется состояние существующего компьютерного парка с целью разработки предложений по внедрению новых информационных технологий, предложений по модернизации или расширению компьютерного парка.

Исследуется используемая учетная политика с целью определения специфики бухгалтерского учета для ЗАО «ХХХ».

Осуществляется обследование существующих бизнес-процессов и бизнес процедур. Производится сравнительный анализ технологий управления предприятием, существующего документооборота с технологиями и документооборотом в рамках системы 1С: ИТРП.

Формируется организационно-функциональная схема автоматизации и разрабатываются требования к проектируемой КИС 1С: ИТРП. На основании установленных учетных подсистем и готовности их для автоматизации формируется поэтапный календарный план внедрения КИС 1С: ИТРП.

1. ОРГАНИЗАЦИОННАЯ СТРУКТУРА ЗАО «ХХХ»

Назначение и информационные связи подразделений. Структурная схема ЗАО «ХХХ» представлена в Приложении 1. *Генеральный директор* руководит производственно-хозяйственной и финансово-экономической деятельностью ЗАО «ХХХ», неся всю полноту ответственности за последствия принимаемых решений, финансово-хозяйственные результаты деятельности. Организует работу и взаимодействие всех структурных подразделений, направляет их деятельность на развитие и совершенствование производства, повышение эффективности работы, рост объемов сбыта продукции и на увеличение прибыли, качества и конкурентоспособности производимой продукции.

Исполнительный директор определяет техническую политику и направления технического развития ЗАО «ХХХ», пути реконструкции и технического перевооружения действующего производства. В соответствии с утвержденными планами руководит разработкой мероприятий по реконструкции и модернизации предприятия. Организует работу по увеличению ассортимента и улучшению качества продукции, внедрение в производство средств комплексной механизации и автоматизации технологических процессов.

Коммерческий директор осуществляет руководство финансово-хозяйственной деятельностью предприятия в области материально-технического обеспечения, заготовки и хранения материалов, сбыта продукции на рынке. Принимает меры по своевременному заключению хозяйственных и финансовых договоров с поставщиками материалов и покупателями, реализации готовой продукции, расширению прямых и длительных хозяйственных связей, обеспечивает выполнение договорных обязательств по поставкам продукции, осуществляет внешнеэкономическую деятельность.

Технический директор обеспечивает техническую подготовку производства, эксплуатацию, ремонт и модернизацию оборудования. Обеспечивает выполнение работ по теплоснабжению, вентиляции, обслуживанию компрессорных установок, строительства.

Начальник производства осуществляет руководство производственно-хозяйственной деятельностью ЗАО «ХХХ». Руководит работой по оперативному регулированию хода производства, обеспечению ритмичного выпуска продукции требуемого ассортимента и качества. Проводит работу по выявлению и освоению технических новшеств, передового опыта, способствующих улучшению технологии, организации производства и росту производительности труда.

Финансовый директор привносит в стратегическую и политическую деятельность финансовые аспекты; ведет работы в области диагностики финансового состояния, экономических тенденций, по трактовке законодательных и финансовых аспектов госрегулирования. Отвечает за управление денежными средствами предприятия, обеспечение своевременного выполнения компанией обязательств.

Обеспечивает работу по анализу планово-экономической деятельности производства и подготовке бухгалтерской, налоговой, управленческой отчетности. Осуществляет контроль за информационно-техническим отделом. Координирует работу с банками и лизинг-компаниями.

Производство состоит из трех цехов: Цех цельномолочной продукции, Цех Сыродельного производства, Цех сушки, Цех мороженого.

Цех мороженого является удаленным цехом (расстояние ~2 км), но является неотъемлемой частью основного производства.

Приемкой молока занимается отдел заготовок сырья, руководящий главным приемным пунктом, расположенный на территории комбината, и 8 молокоприемников, расположенных в Псковской области.

Лаборатория осуществляет контроль качества принимаемого молока и выпущенной готовой продукции.

Сбыт готовой продукции состоит из Отдела сбыта комбината и структурных подразделений: Торговый Дом «XXX-1», Торговый Дом «XXX-2», Торговый Дом «XXX-3».

1. ОПИСАНИЕ ФУНКЦИЙ ПОДРАЗДЕЛЕНИЙ И СУЩЕСТВУЮЩЕГО ДОКУМЕНТООБОРОТА

1. Бухгалтерия (Приложение 2)

2. Планово-экономический отдел (Приложение 3)

3. Производство (Приложение 4)

4. Отдел сбыта (Приложение 5)

5. Отдел снабжения (Приложение 6)

6. Отдел логистики (ОПГП-ЦПМ, ОПГП – Цех мороженого) (Приложение 7).

1. ПРИНЯТАЯ УЧЕТНАЯ ПОЛИТИКА

Ниже раскрыты ключевые положения учетной политики, определяющей ведение бухгалтерского учета в ЗАО «XXX».

Основными задачами бухгалтерского учета являются:

1. формирование полной и достоверной информации о хозяйственных процессах и результатах деятельности предприятия;

2. обеспечение контроля наличия и движения имущества, использования материальных, трудовых и финансовых ресурсов;

3. своевременное предупреждение негативных явлений в хозяйственно-финансовой деятельности.

Согласно учетной политике предприятия, бухгалтерский учет имущества, обязательств и хозяйственных операций ведется на основе натуральных измерителей в денежном

выражении путем сплошного, непрерывного, документального и взаимосвязанного взаимодействия.

Должен обеспечиваться контроль и отражение на счетах всех хозяйственных операций, представление оперативной и результативной информации в установленные сроки.

Факт свершения хозяйственной операции фиксируется первичными документами, которые и являются основанием для записи в регистрах бухгалтерского учета. Бухгалтерский учет ведется по журнально-ордерной (ж/о) форме с последующим заполнением Главной книги.

1. ОПИСАНИЕ ТЕКУЩЕГО УРОВНЯ АВТОМАТИЗАЦИИ

Начало работ по автоматизации учетных функций в ЗАО «ХХХ» относится к 1993 году. На данный момент функционирует автоматизированная система, построенная как совокупность автоматизированных рабочих мест (АРМ), каждое из которых имеет свою четко выраженную функциональную специализацию и обеспечивает решение задач по отдельным участкам учета. При этом автоматизированное рабочее место отдельного участка учета реализовано на одном или нескольких компьютерах, не связанных физически между собой, но работающих на одной информационной базе, которая поддерживается на всех компьютерах данного рабочего места. Обмен информацией между компьютерами автоматизированного рабочего места осуществляется через дискету. Обмен информацией между отдельными АРМами не производится из-за отсутствия единой информационной базы и единой системы классификации и кодирования информации, что значительно снижает оперативность получения обобщающей информации.

Основная литература:

1. Антонов, В.Ф. Методы и средства проектирования информационных систем : учебное пособие / В.Ф. Антонов, А.А. Москвитин ; Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Северо-Кавказский федеральный университет», Министерство образования и науки Российской Федерации. - Ставрополь : СКФУ, 2016. - 342 с. : ил. - Библиогр. в кн. ; То же [Электронный ресурс]. - URL: http://biblioclub.ru/index.php?page=book&id=458663

2. Антимиров, В. М. Проектирование аппаратуры систем автоматического управления. В 2 ч. Ч. 1 : учебное пособие для СПО / В. М. Антимиров. — 2-е изд. — Саратов, Екатеринбург : Профобразование, Уральский федеральный университет, 2019. — 92 c. — ISBN 978-5-4488-0401-4, 978-5-7996-2834-5. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87852.html. — Режим доступа: для авторизир. Пользователей

3. Курушин, В. Д. Графический дизайн и реклама / В. Д. Курушин. — 2-е изд. — Саратов : Профобразование, 2019. — 271 c. — ISBN 978-5-4488-0094-8. — Текст : электронный // Электронно-библиотечная система IPR BOOKS : [сайт]. — URL: http://www.iprbookshop.ru/87990.html. — Режим доступа: для авторизир. пользователей

Дополнительная литература:

Федорова Г.И. Разработка, внедрение и адаптация программного обеспечения отраслевой направленности. Учебное пособие. Изд.: КУРС, Инфра-М. Среднее профессиональное образование. 2016 г. 336 стр.