ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

УПРАВЛЕНИЕ ДИСТАНЦИОННОГО ОБУЧЕНИЯ И ПОВЫШЕНИЯ КВАЛИФИКАЦИИ

Кафедра «Программное обеспечение вычислительной техники и автоматизированных систем»

Учебно-методическое пособие

по дисциплине

«Введение в программную инженерию»

Автор
Рыбалко И.П.

Ростов-на-Дону, 2018

Аннотация

Учебно-методическое пособие предназначено для студентов очной формы обучения направления 09.03.04 «Программная инженерия».

Авторы

К.т.н.
Доцент каф. ПОВТиАС
Рыбалко И.П.

Оглавление

1. Лабораторная работа №1: Анализ информации пользователя 4

1.1. Задание 4
1.2. Содержание отчета.. 4
1.3. Контрольные вопросы......................................12

2. Лабораторная работа №2: Документирование программных средств 12

2.1. Краткая теория 12
2.2. Отечественная нормативная база 17
2.3. Задание к лабораторной работе 28
2.4. Контрольные вопросы......................................29

3. Лабораторная работа №3: Технологические этапы разработки программного обеспечения............................29

3.1. Технологические этапы создания ПС 29
3.2. Финансирование проекта..................................40
3.3. Задание к лабораторной работе 48
3.4. Контрольные вопросы......................................48

4. Лабораторная работа №4: Структурно-модульное проектирование HIPO-СХЕМЫ..49

4.1. Структурно-модульное проектирование............49
4.2. HIPO-схемы..56
4.3. Задание к лабораторной работе 60
4.4. Контрольные вопросы......................................61

5. Лабораторная работа №5: Этапы сопровождения программного обеспечения..61

5.1. Теория 61
5.2. Задание к лабораторной работе. 146
5.3. Контрольные вопросы....................................146

1. ЛАБОРАТОРНАЯ РАБОТА №1: АНАЛИЗ ИНФОРМАЦИИ ПОЛЬЗОВАТЕЛЯ

1.1. Задание

1. Провести анализ предметной области программного средства.
2. Выделить не менее 3-х групп пользователей. Определить их характеристики, которые необходимо учесть при проектировании интерфейса.
3. Определить характеристики среды и особенности взаимодействия пользователей с системой в этой среде.
4. Провести анализ характеристик технических средств и характеристик программных средств, с которыми должна взаимодействовать система. Разработать схему взаимодействия подсистем и пользователей.
5. Определить требования пользователей к системе (основные цели и задачи). Определить требования к выполнению задач (каких показателей должны достичь пользователи в количественных и качественных характеристиках).

1.2. Содержание отчета

1. Описание предметной области.

Необходимо привести подробное описание имеющихся процессов ***<u>до внедрения разрабатываемой системы.</u>***

Описание предметной области **<u>может</u>** содержать:

- данные о деятельности предприятия, например:
  - вид деятельности,
  - организационная структура предприятия,
  - описание производственных процессов и пр.
- описание взаимодействия участников процесса:
  - схема информационных потоков,
  - форматы передаваемых данных,
  - способы обмена данными и пр.
- описание используемых на текущий момент программно-аппаратных средств;
- и прочее.

2. Анализ характеристик пользователей, их целей и задач.

В рамках предметной области выделить возможные группы пользователей (3-4 группы, отличающихся по своим характеристикам и/или задачам).

Для каждого пользователя привести следующее:

– группа пользователей/пользователь (*название);*
– характеристики профиля пользователя*;*
– описание рабочей среды;
– цели и задачи использования системы, показатели качества.

Группировать описания необходимо **по пользователям!!!**, а не по характеристикам (т.е, вначале описать ВСЁ для одной группы пользователей, потом для второй и т.д.).

Характеристики профиля пользователя могут быть описаны в виде таблицы или в текстовом виде:

Характеристика	Значение/диапазон значений	На что будет влиять в интерфейсе
Группа характеристик1		
1.		
2.		
…		
Группа характеристик 2		
1.		
2.		
…		

Рекомендуемые группы характеристик:

- *Психофизиологические характеристики*

Влияют на:

– возможность использования определенных каналов восприятия;
– масштабирование элементов интерфейса и пр.

- *Социально-демографические сведения*

Влияют на:

– глубину и уровень детализации программы;
– набор используемых метафор;

- язык представления информации;
- стиль взаимодействия с программой;
- лексику элементов интерфейса программы.

- *Опыт работы с компьютерами и прикладными программами (уровень компьютерной грамотности, специальные навыки)*.

Влияет на:

- содержание и время обучения;
- объем и структуру справочной системы;
- организацию протокола взаимодействия (привычность).

Примеры описания характеристик (**в работе указывать только НЕОБХОДИМЫЕ характеристики**):

Характеристика	Значение/диапазон значений	На что будет влиять в интерфейсе
Психофизиологические характеристики		
Возраст/проблемы со зрением	Программой могут пользоваться пожилые люди/ дальнозоркость	Возможность масштабирования шрифтов *или* Использование крупных шрифтов. Размер используемых в экранных формах шрифтов аналогичен 14-16 Times New Roman
Возраст	Дети 3-6 лет	Не умеют читать. В интерфейсе стоит использовать графические метафоры, а не текстовые обозначения

Инвалидность	По зрению	Использование голосовых подсказок, голосового ввода команд и пр.
Пол	Женский	Использование цветовой палитры в теплых тонах (например, при реализации женского форума)
...		
Социально-демографические сведения		
Язык	Русский	Язык интерфейса системы русский
Язык	Русский, украинский, английский	Мультиязычность интерфейса. Возможность переключения с одного языка на другой.
Сфера деятельности	Бухгалтерия	Специальные метафоры и термины, используемые при обозначении элементов интерфейса (названия пунктов меню, подписи, подсказки и пр.)
Социальная среда	Готы	Используемая в интерфейсе символика, система метафор, цветовая палитра.

Религиозные взгляды	Мусульманство	Отсутствие в интерфейсе элементов, противоречащих религиозным взглядам.
…		

При описании опыта работы с компьютерами и прикладными программами можно опираться на нижеприведенные характеристики (обратите внимание, что конкретные группы пользователей в рассматриваемой Вами предметной области могут обладать другими наборами навыков!!!):

Начинающий пользователь:¶

Уровень «начинающего пользователя» подразумевает владение следующими навыками:

1. Включить, выключить, перезагрузить компьютер.
2. Пользоваться клавиатурой и компьютерной мышкой.
3. Работать с документами MS Office или Open Office:
   - создать новый и/или открыть существующий файл;
   - копировать и вставлять текст;
   - открыть и сохранить файл под новым именем в новом месте в другом формате;
   - копировать, переносить и удалять файлы;
   - переименовать файл;
4. Создать новую папку.
5. Пользоваться электронной почтой.¶

Опытный пользователь:

Уровень «опытного пользователя» подразумевает владение навыками начинающего, плюс следующие навыки (*дополнения по пунктам навыков в соответствии с их номерами*):

1. Опыт работы в какой-либо операционной системе
   - OS Microsoft XP;
   - OS Microsoft Vista;
   - OS MAC 8; 9; X.
2. Организовать структуру папок и/или файлов;
3. Найти файл на компьютере;
4. Скачивать на свой компьютер программы и файлы из Интернета;
5. Пользоваться электронной почтой, включая Приложения;

2. Работа с текстом
   - Microsoft Office Word

Создание документа, установка параметров страниц, форматирование текста, изменение цвета и шрифта, вставка картинки из файла, вставка диаграммы, вставка таблицы, сохранение и открытие файла документа.

3. Создание электронных таблиц, графиков, диаграмм

- Microsoft Office Excel

Создание и сохранение документа, форматирование текста, цвета и шрифта, знание функций ячеек таблицы.

- Adobe Acrobat Reader

Открытие документа, пролистывание страниц, масштабирование документа.

6. Работа в сети Internet

- Microsoft Internet Explorer Browser

Загрузка интернет страниц в окно браузера, свободное перемещение с использованием гиперссылок и панели навигации;

Уметь добавить страницу на панель «Избранное» и создать закладку;

Пользоваться поисковыми системами.

7. Работа с почтовым клиентом

- Microsoft Internet Explorer Browser, или
- Microsoft Outlook, или
- - The Bat или др.

Создание ящика электронной почты, отправка и прочтение сообщений, загрузка файлов в сообщение.

8. Упаковка и распаковка данных

- - 7-Zip, или
- - WinRar, или
- - WinZip

Создание, сохранение и распаковка архива.

Продвинутый пользователь:

Уровень «продвинутого пользователя» подразумевает владение навыками опытного пользователя, плюс следующие навыки:

1. Устанавливать удалять программы;
2. Осуществить настройки компьютера, необходимые для работы с требуемыми программами.
3. Создание электронных таблиц, графиков, диаграмм

- Microsoft Office Excel
- Работа с мастером функций и диаграмм, просмотр электронных публикаций
- Adobe Acrobat Reader

Перевод файла в формат PDF.

Создание презентаций

- Microsoft Office Powerpoint

Создание слайда на основе шаблонов, вставка картинки из файла, работа с текстом, сохранение и открытие файла презентации, настройка анимации.

6. Работа в сети Internet

- Microsoft Internet Explorer Browser (либо любой другой)

Настройка свойств обозревателя браузера.

8. Упаковка и распаковка данных

- - 7-Zip, или
- - WinRar, или
- - WinZip

Создание самораспаковывающегося архива, установка пароля на архив.

Описание рабочей среды может включать нижеприведенные характеристики (описать в виде таблицы или в текстовом виде по аналогии с характеристиками профиля пользователя).

Физическая сторона рабочей среды:

- освещение
- шум
- рабочее пространство
- температура
- наличие компьютеров, телефонов
- количество персонала и т.п.

Место работы пользователя и степень его мобильности:

- офис/квартира/…
- стационарно/с передвижениями и т.д.

Вопросы эргономики и условий труда:

- задействуется ли зрение/слух
- работа ведется сидя/стоя
- режимы работы, длительность смен и т.д.

Особые запросы:

- уровень подготовки
- физическое состояние
- интерес к познавательному процессу
- особенности речи и возможные недостатки

Интернационализация и другие культурологические условия

Программно-аппаратные средства, используемые на текущий момент и те, которые планируется задействовать в системе:

- характеристики компьютеров и/или других

устройств (конфигурация устройств)

- характеристики технических каналов связи
- операционная система
- сопутствующее ПО и пр.

Цели и задачи использования системы, показатели качества привести в виде таблицы или в текстовом виде. Количество целей 2-4. каждая цель достигается решением одной или нескольких задач. Каждой задаче соответствуют свои показатели качества.

Типичный программный продукт должен:

- сокращать работу с бумагами;
- уменьшать ошибки пользователей;
- автоматизировать существующие ручные процессы;
- повышать скорость совершения транзакций и пр.

Сценарий выполнения задачи должен содержать описание действий, выполняемых пользователем и ответных реакций системы. При описании сценария следует избегать привязки к конкретным интерфейсным решениям (например, названий конкретных элементов интерфейса – кнопка, раскрывающийся список и пр.).

Табличное представление целей и задач пользователей:

Цель	Задачи	Показатели качества	Сценарий
Цель1	1		
	2		
	…		
Цель2	1		
	2		
	…		
…			

Текстовое представление целей и задач пользователей:

Цель1: *формулировка*

Задача 1.1:

Формулировка*: описание задачи.*

Показатели качества: *описание.*

Сценарий: *описание сценария.*

Задача 1.2:

Формулировка*: описание задачи.*

Показатели качества: *описание.*
Сценарий: *описание сценария.*
Цель2: *формулировка*
Задача 2.1: …

1.3. Контрольные вопросы

1. Функции когнитолога.
2. Принципы организации опроса эксперта и системы приобретения знаний.
3. Простое и структурированное интервью.
4. Протокольный анализ.
5. Метод декомпозиции цели.
6. Методы сопоставления.
7. Методы заочной консультации.
8. Принципы построения автоматизированных систем инженерии знаний.

2. ЛАБОРАТОРНАЯ РАБОТА №2: ДОКУМЕНТИРОВАНИЕ ПРОГРАММНЫХ СРЕДСТВ

2.1. Краткая теория

При разработке ПС создается и используется большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками ПС, как средство управления разработкой ПС и как средство передачи пользователям информации, необходимой для применения и сопровождения ПС. На создание этой документации приходится большая доля стоимости ПС.

Эту документацию можно разбить на две группы:

- Документы управления разработкой ПС.
- Документы, входящие в состав ПС.

Документы управления разработкой ПС (*software process documentation*) управляют и протоколируют процессы разработки и сопровождения ПС, обеспечивая связи внутри коллектива разработчиков ПС и между коллективом разработчиков и *менеджерами ПС* (*software managers*) - лицами, управляющими разработкой ПС. Эти документы могут быть следующих типов:

- *Планы, оценки, расписания.* Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения ПС.

- *Отчеты об использовании ресурсов в процессе разработки.* Создаются менеджерами.
- *Стандарты.* Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПС. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка ПС.
- *Рабочие документы.* Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПС.
- *Заметки и переписка.* Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

Документы, входящие в состав ПС (*software product documentation*), описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС). Здесь следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПС (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами) - во всяком случае, они должны быть проверены (протестированы) на соответствие программам ПС. Эти документы образуют два комплекта с разным назначением:

- Пользовательская документация ПС (П-документация).
- Документация по сопровождению ПС (С-документация).

Пользовательская документация программных средств

Пользовательская документация ПС (*user documentation*) объясняет пользователям, как они должны действовать, чтобы применить разрабатываемое ПС. Она необходима, если ПС предполагает какое-либо взаимодействие с пользователями. К такой документации относятся документы, которыми должен руководствоваться пользователь при *инсталляции* ПС (при установке ПС с соответствующей настройкой на среду применения ПС), при применении ПС для решения своих задач и при управлении ПС (например, когда разрабатываемое ПС будет взаимодействовать с другими системами). Эти документы частично затрагивают вопро-

сы сопровождения ПС, но не касаются вопросов, связанных с модификацией программ.

В связи с этим следует различать две категории пользователей ПС: ординарных пользователей ПС и администраторов ПС. *Ординарный пользователь ПС* (e*nd-user*) использует ПС для решения своих задач (в своей предметной области). Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью ПС. Он может и не знать многих деталей работы компьютера или принципов программирования. *Администратор ПС* (*system administrator*) управляет использованием ПС ординарными пользователями и осуществляет сопровождение ПС, не связанное с модификацией программ. Например, он может регулировать права доступа к ПС между ординарными пользователями, поддерживать связь с поставщиками ПС или выполнять определенные действия, чтобы поддерживать ПС в рабочем состоянии, если оно включено как часть в другую систему.

Состав пользовательской документации зависит от аудиторий пользователей, на которые ориентировано разрабатываемое ПС, и от режима использования документов. Под *аудиторией* здесь понимается контингент пользователей ПС, у которого есть необходимость в определенной пользовательской документации ПС. Удачный пользовательский документ существенно зависит от точного определения аудитории, для которой он предназначен. Пользовательская документация должна содержать информацию, необходимую для каждой аудитории. Под *режимом использования* документа понимается способ, определяющий, каким образом используется этот документ. Обычно пользователю достаточно больших программных систем требуются либо документы для изучения ПС (использование в виде *инструкции*), либо для уточнения некоторой информации (использование в виде *справочника*).

В соответствии с работами можно считать типичным следующий состав пользовательской документации для достаточно больших ПС:

- *Общее функциональное описание ПС*. Дает краткую характеристику функциональных возможностей ПС. Предназначено для пользователей, которые должны решить, насколько необходимо им данное ПС.
- *Руководство по инсталяции ПС*. Предназначено для администраторов ПС. Оно должно детально предписывать, как устанавливать системы в конкретной среде, в частности, должно содержать описание компьютерно-считываемого

носителя, на котором поставляется ПС, файлы, представляющие ПС, и требования к минимальной конфигурации аппаратуры.

- *Инструкция по применению ПС.* Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для ее изучения.
- *Справочник по применению ПС*. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для избирательного поиска отдельных деталей.
- *Руководство по управлению ПС*. Предназначено для администраторов ПС. Оно должно описывать сообщения, генерируемые, когда ПС взаимодействует с другими системами, и как должен реагировать администратор на эти сообщения. Кроме того, если ПС использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Как уже говорилось ранее (см. лекцию 4), разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПС. Она должна быть достаточно проста и удобна для пользователя (в противном случае это ПС, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПС, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов , в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

Пользовательская документация программных средств

Документация по сопровождению ПС (*system documentation*) описывает ПС с точки зрения ее разработки. Эта документация необходима, если ПС предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ. Как уже отмечалось, сопровождение - это продолжающаяся разработка. Поэтому в случае необходимости модернизации ПС к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же

документацией, которая определяла деятельность команды первоначальных (основных) разработчиков ПС, - с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Чтобы понять строение и процесс разработки модернизируемого ПС, команда разработчиков-сопроводителей должна изучить эту документацию, а затем внести в нее необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное ПС.

Документация по сопровождению ПС можно разбить на две группы:

1. документация, определяющая строение программ и структур данных ПС и технологию их разработки;
2. документацию, помогающую вносить изменения в ПС.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПС. Она включает следующие документы:

- Внешнее описание ПС (Requirements document).
- Описание архитектуры ПС (description of the system architecture), включая внешнюю спецификацию каждой ее программы (подсистемы).
- Для каждой программы ПС - описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
- Для каждого модуля - его спецификация и описание его строения (design description).
- Тексты модулей на выбранном языке программирования (program source code listings).
- Документы установления достоверности ПС (validation documents), описывающие, как устанавливалась достоверность каждой программы ПС и как информация об установлении достоверности связывалась с требованиями к ПС.

Документы установления достоверности ПС включают, прежде всего, документацию по тестированию (схема тестирования и описание комплекта тестов), но могут включать и результаты других видов проверки ПС, например, доказательства свойств программ. Для обеспечения приемлемого качества этой документации полезно следовать общепринятым рекомендациям и стандартам .

Документация второй группы содержит

- *Руководство по сопровождению ПС* (system maintenance guide), которое описывает особенности реализации ПС (в частности, трудности, которые пришлось преодолевать) и как учтены возможности развития ПС в его строении (конструкции). В нем также фиксируются, какие части ПС являются аппаратно- и программно-зависимыми.

Общая проблема сопровождения ПС - обеспечить, чтобы все его представления шли в ногу (оставались согласованными), когда ПС изменяется. Чтобы этому помочь, связи и зависимости между документами и их частями должны быть отражены в руководстве по сопровождению, и зафиксированы в базе данных управления конфигурацией.

2.2. Отечественная нормативная база

Основу отечественной нормативной базы в области документирования ПС составляет комплекс стандартов Единой системы программной документации (ЕСПД). Основная и большая часть комплекса ЕСПД была разработана в 70-е и 80-е годы 20 века. Сейчас этот комплекс представляет собой систему межгосударственных стандартов стран СНГ (ГОСТ), действующих на территории Российской Федерации на основе межгосударственного соглашения по стандартизации.

Единая система программной документации — это комплекс государственных стандартов, устанавливающих взаимоувязанные правила разработки, оформления и обращения программ и программной документации.

Стандарты ЕСПД в основном охватывают ту часть документации, которая создается в процессе разработки ПС, и связаны, по большей части, с документированием функциональных характеристик ПС. Следует отметить, что стандарты ЕСПД (ГОСТ 19) носят рекомендательный характер. Впрочем, это относится и ко всем другим стандартам в области ПС (ГОСТ 34, международному стандарту ISO/IEC и др.). Дело в том, что в соответствии с Законом РФ «О стандартизации» эти стандарты становятся обязательными на контрактной основе, т.е. при ссылке на них в договоре на разработку (поставку) ПС.

В состав ЕСПД входят:

- основополагающие и организационно-методические стандарты;
- стандарты, определяющие формы и содержание программных документов, приме- няемых при обработке данных;

• стандарты, обеспечивающие автоматизацию разработки программных документов.

Говоря о состоянии ЕСПД в целом, можно констатировать, что большая часть стандартов ЕСПД морально устарела. К числу основных недостатков ЕСПД можно отнести:

• ориентацию на единственную «каскадную» модель жизненного цикла ПС;

• отсутствие четких рекомендаций по документированию характеристик качества ПС;

• отсутствие системной увязки с другими действующими отечественными системами стандартов по ЖЦ и документированию
продукции в целом, например ЕСКД;

• нечетко выраженный подход к документированию ПС как товарной продукции;

• отсутствие рекомендаций по самодокументированию ПС, на пример, в виде экранных меню и средств оперативной помощи пользователю (хелпов);

отсутствие рекомендаций по составу, содержанию и оформлению перспективных документов на ПС, согласованных с рекомендациями международных и региональных стандартов.

ГОСТ 19.404-79 ЕСПД. Пояснительная записка. Требования к содержанию и оформлению

Согласно данному стандарту пояснительная записка должна включать следующие разделы:

1. Введение.
2. Назначение и область применения.
3. Технические характеристики.
4. Ожидаемые технико-экономические показатели.
5. Источники, использованные при разработке.

Введение должно содержать наименование программы и/или обозначение темы разработки, а также документы, на основе которых ведется разработка.

При описании ***назначения*** *и области* ***применения*** указывают назначение программы, краткую характеристику области применения программы.

В разделе *Технические* ***характеристики*** содержатся:

• постановка задачи на разработку программы, описание применяемых математических методов и различных ограничений, связанных с выбранным математическим аппаратом;

• описание алгоритма и/или функционирования программы

с обоснованием выбора схемы алгоритма решения задачи, возможного взаимодействия программы с другими программами;

- описание и обоснование выбора метода организации входных и выходных данных;
- описание и обоснование выбора состава технических и программных средств на основе проведенных расчетов и анализов, распределение носителей данных, которые использует программа.

В качестве *ожидаемых технико-экономических показателей* указывают показатели, обосновывающие преимущество выбранного варианта технического решения, а также при необходимости ожидаемые оперативные показатели.

При описании источников, использованных при разработке, необходимо привести перечень научно-технических публикаций, нормативно-технических документов и других научно-технических материалов, на которые есть ссылки в основном тексте.

ГОСТ 19.503-79 ЕСПД. Руководство системного программиста. Требования к содержанию и оформлению

Руководство системного программиста должно содержать следующие разделы:

1. Общие сведения о программе.
2. Структура программы.
3. Настройка программы.
4. Проверка программы.
5. Дополнительные возможности.
6. Сообщения системному программисту.

При необходимости допускается опускать раздел, описывающий дополнительные возможности.

При описании *общих сведений о программе* необходимо указать назначение и функции программы и сведения о технических и программных средствах, обеспечивающих выполнение данной программы.

В разделе *Структура программы* приводятся сведения о структуре программы, ее составных частях и связях с другими программами.

Раздел *Настройка программы* должен содержать описание действий по настройке программы на условия конкретного применения.

При описании *проверки программы* необходимо привести и описать способы проверки, позволяющие дать общее заключение о работоспособности программы (контрольные примеры, методы

прогона, результаты).

Раздел *Дополнительные возможности* должен содержать описание дополнительных разделов функциональных возможностей программы и способов их выбора.

В разделе *Сообщения системному программисту* необходимо указать тексты сообщений, выдаваемых в ходе выполнения программы, описание содержания и действий, которые необходимо предпринять по этим сообщениям.

ГОСТ 19.402-78 ЕСПД. Описание программыДанный стандарт определяет состав и требования к содержанию программного документа «Описание программы». Описание программы включает:

1. Общие сведения.
2. Функциональное назначение.
3. Описание логической структуры.
4. Используемые технические средства.
5. Вызов и загрузка.
6. Входные данные.
7. Выходные данные.

В разделе ***Общие сведения*** указывают:

- обозначение и наименование программы;
- программное обеспечение, необходимое для функционирования программы;
- языки программирования, на которых написана программа.

Раздел ***Функциональное назначение*** должен отражать классы решаемых задач и/или назначение программы, сведения о функциональных ограничениях на применение.

При описании *логической структуры* должны быть отражены:

- алгоритм программы;
- используемые методы;
- структура программы с описанием функций составных частей и связей между ними;
- связи программы с другими программами.

В разделе *Используемые* ***технические средства*** указывают типы ЭВМ и устройств, которые используются при работе программы.

При описании раздела ***Вызов и*** *загрузка* указывают способ вызова программы с соответствующего носителя данных и входные точки в программу.

Раздел ***Входные данные*** отражает:

• характер, организацию и предварительную подготовку входных данных;

• формат, описание и способ кодирования входных данных.

Раздел ***Выходные данные*** отражает:

• характер и организацию выходных данных;

• формат, описание и способ кодирования выходных данных.

ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам

Данный стандарт устанавливает общие требования к оформлению программных документов для вычислительных машин, комплексов и систем независимо от их назначения и области применения и предусмотренных стандартами Единой системы программной документации (ЕСПД) для любого способа выполнения документов на различных носителях данных.

Программный документ может быть представлен на различных типах носителей данных и состоит из следующих условных частей:

• титульной;

• информационной;

• основной.

Правила оформления документа и его частей на каждом носителе данных устанавливаются стандартами ЕСПД на правила оформления документов на соответствующих носителях данных.

Титульная часть оформляется согласно ГОСТ 19.104—78.

Информационная часть должна состоять из аннотации и содержания. В аннотации приводят сведения о назначении документа и краткое изложение основной части. Содержание включает перечень записей о структурных элементах основной части документа.

Состав и структура основной части программного документа устанавливаются стандартами ЕСПД на соответствующие документы.

ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению

Техническое задание (ТЗ) содержит совокупность требований к ПС и может использоваться как критерий проверки и приемки разработанной программы. Поэтому достаточно полно составленное (с учетом возможности внесения дополнительных разделов) и принятое заказчиком и разработчиком ТЗ является одним из основополагающих документов проекта ПС.

Техническое задание должно содержать следующие разделы:

- введение;
- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки;
- в техническое задание допускается включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов

В данном разделе должны быть указаны предварительный состав программной документации и при необходимости специальные требования к ней. ГОСТ подразделяет программы на следующие виды:

Компонент — программа, рассматриваемая как единое целое, выполняющая законченную функцию и применяемая самостоятельно или в составе комплекса.

Комплекс — программа, состоящая из двух или более компонентов, выполняющих взаимосвязанные функции, и применяемая самостоятельно или в составе другого комплекса.

Документация, разработанная на программу, может использоваться для реализации и передачи программы на носителях данных, а также для изготовления программного изделия.

К числу программных данный ГОСТ относит документы, содержащие сведения, необходимые для разработки, изготовления, сопровождения и эксплуатации программ. Рассмотрим виды программных документов и их содержание:

Спецификация — содержит состав программы и документацию на нее.

Ведомость держателей подлинников — содержит перечень предприятий, на которых хранят подлинники программных документов.

Текст программы — представляет запись программы с необходимыми комментариями.

Описание программы — содержит сведения о логической

структуре и функционировании программы.

Программа и методика испытаний — содержит требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля.

Техническое задание — описывает назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний.

Пояснительная записка — содержит схему алгоритма, общее описание алгоритма и (или) функционирования программы, а также обоснование принятых технических и технико-экономических решений.

Эксплуатационные документы — содержат сведения для обеспечения функционирования и эксплуатации программы.

В зависимости от способа выполнения и характера применения программные документы подразделяются на подлинник, дубликат и копию (ГОСТ 2.102-68), предназначенные для разработки, сопровождения и эксплуатации программы.

Допускается объединять отдельные виды эксплуатационных документов (за исключением ведомости эксплуатационных документов и формуляра). Необходимость объединения этих документов указывается в техническом задании. Объединенному документу присваивают наименование и обозначение одного из объединяемых документов. В объединенных документах должны быть приведены сведения, которые необходимо включать в каждый объединяемый документ

Документация, создаваемая и используемая в процессе разработки программных средств

При разработке ПС создается и используется большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками ПС, как средство управления разработкой ПС и как средство передачи пользователям информации, необходимой для применения и сопровождения ПС. На создание этой документации приходится большая доля стоимости ПС.

Эту документацию можно разбить на две группы :

- Документы управления разработкой ПС.
- Документы, входящие в состав ПС.

Документы управления разработкой ПС (*software process documentation*) управляют и протоколируют процессы разработки и сопровождения ПС, обеспечивая связи внутри коллектива раз-

работчиков ПС и между коллективом разработчиков и *менеджерами ПС* (*software managers*) - лицами, управляющими разработкой ПС. Эти документы могут быть следующих типов :

- *Планы, оценки, расписания.* Эти документы создаются менеджерами для прогнозирования и управления процессами разработки и сопровождения ПС.
- *Отчеты об использовании ресурсов в процессе разработки.* Создаются менеджерами.
- *Стандарты.* Эти документы предписывают разработчикам, каким принципам, правилам, соглашениям они должны следовать в процессе разработки ПС. Эти стандарты могут быть как международными или национальными, так и специально созданными для организации, в которой ведется разработка ПС.
- *Рабочие документы.* Это основные технические документы, обеспечивающие связь между разработчиками. Они содержат фиксацию идей и проблем, возникающих в процессе разработки, описание используемых стратегий и подходов, а также рабочие (временные) версии документов, которые должны войти в ПС.
- *Заметки и переписка.* Эти документы фиксируют различные детали взаимодействия между менеджерами и разработчиками.

Документы, входящие в состав ПС (*software product documentation*), описывают программы ПС как с точки зрения их применения пользователями, так и с точки зрения их разработчиков и сопроводителей (в соответствии с назначением ПС). Здесь следует отметить, что эти документы будут использоваться не только на стадии эксплуатации ПС (в ее фазах применения и сопровождения), но и на стадии разработки для управления процессом разработки (вместе с рабочими документами) - во всяком случае, они должны быть проверены (протестированы) на соответствие программам ПС. Эти документы образуют два комплекта с разным назначением:

- Пользовательская документация ПС (П-документация).
- Документация по сопровождению ПС (С-документация).

Пользовательская документация программных средств

Пользовательская документация ПС (*user documentation*) объясняет пользователям, как они должны действовать, чтобы применить разрабатываемое ПС . Она необходима, если ПС предполагает какое-либо взаимодействие с пользователями. К такой

документации относятся документы, которыми должен руководствоваться пользователь при *инсталляции* ПС (при установке ПС с соответствующей настройкой на среду применения ПС), при применении ПС для решения своих задач и при управлении ПС (например, когда разрабатываемое ПС будет взаимодействовать с другими системами). Эти документы частично затрагивают вопросы сопровождения ПС, но не касаются вопросов, связанных с модификацией программ.

В связи с этим следует различать две категории пользователей ПС: ординарных пользователей ПС и администраторов ПС. *Ординарный пользователь ПС* (е*nd-user*) использует ПС для решения своих задач (в своей предметной области). Это может быть инженер, проектирующий техническое устройство, или кассир, продающий железнодорожные билеты с помощью ПС. Он может и не знать многих деталей работы компьютера или принципов программирования. *Администратор ПС* (*system administrator*) управляет использованием ПС ординарными пользователями и осуществляет сопровождение ПС, не связанное с модификацией программ. Например, он может регулировать права доступа к ПС между ординарными пользователями, поддерживать связь с поставщиками ПС или выполнять определенные действия, чтобы поддерживать ПС в рабочем состоянии, если оно включено как часть в другую систему.

Состав пользовательской документации зависит от аудиторий пользователей, на которые ориентировано разрабатываемое ПС, и от режима использования документов. Под *аудиторией* здесь понимается контингент пользователей ПС, у которого есть необходимость в определенной пользовательской документации ПС . Удачный пользовательский документ существенно зависит от точного определения аудитории, для которой он предназначен. Пользовательская документация должна содержать информацию, необходимую для каждой аудитории. Под *режимом использования* документа понимается способ, определяющий, каким образом используется этот документ. Обычно пользователю достаточно больших программных систем требуются либо документы для изучения ПС (использование в виде *инструкции*), либо для уточнения некоторой информации (использование в виде *справочника*).

В соответствии с работами можно считать типичным следующий состав пользовательской документации для достаточно больших ПС:

- *Общее функциональное описание ПС*. Дает краткую характеристику функциональных возможностей ПС. Предна-

значено для пользователей, которые должны решить, насколько необходимо им данное ПС.

- *Руководство по инсталяции ПС*. Предназначено для администраторов ПС. Оно должно детально предписывать, как устанавливать системы в конкретной среде, в частности, должно содержать описание компьютерно-считываемого носителя, на котором поставляется ПС, файлы, представляющие ПС, и требования к минимальной конфигурации аппаратуры.
- *Инструкция по применению ПС.* Предназначена для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для ее изучения.
- *Справочник по применению ПС*. Предназначен для ординарных пользователей. Содержит необходимую информацию по применению ПС, организованную в форме удобной для избирательного поиска отдельных деталей.
- *Руководство по управлению ПС*. Предназначено для администраторов ПС. Оно должно описывать сообщения, генерируемые, когда ПС взаимодействует с другими системами, и как должен реагировать администратор на эти сообщения. Кроме того, если ПС использует системную аппаратуру, этот документ может объяснять, как сопровождать эту аппаратуру.

Как уже говорилось ранее (см. лекцию 4), разработка пользовательской документации начинается сразу после создания внешнего описания. Качество этой документации может существенно определять успех ПС. Она должна быть достаточно проста и удобна для пользователя (в противном случае это ПС, вообще, не стоило создавать). Поэтому, хотя черновые варианты (наброски) пользовательских документов создаются основными разработчиками ПС, к созданию их окончательных вариантов часто привлекаются профессиональные технические писатели. Кроме того, для обеспечения качества пользовательской документации разработан ряд стандартов , в которых предписывается порядок разработки этой документации, формулируются требования к каждому виду пользовательских документов и определяются их структура и содержание.

Документация по сопровождению программных средств

Документация по сопровождению ПС (*system documentation*) описывает ПС с точки зрения ее разработки. Эта

документация необходима, если ПС предполагает изучение того, как оно устроена (сконструирована), и модернизацию его программ. Как уже отмечалось, сопровождение - это продолжающаяся разработка. Поэтому в случае необходимости модернизации ПС к этой работе привлекается специальная команда разработчиков-сопроводителей. Этой команде придется иметь дело с такой же документацией, которая определяла деятельность команды первоначальных (основных) разработчиков ПС, - с той лишь разницей, что эта документация для команды разработчиков-сопроводителей будет, как правило, чужой (она создавалась другой командой). Чтобы понять строение и процесс разработки модернизируемого ПС, команда разработчиков-сопроводителей должна изучить эту документацию, а затем внести в нее необходимые изменения, повторяя в значительной степени технологические процессы, с помощью которых создавалось первоначальное ПС.

Документация по сопровождению ПС можно разбить на две группы:

3. документация, определяющая строение программ и структур данных ПС и технологию их разработки;
4. документацию, помогающую вносить изменения в ПС.

Документация первой группы содержит итоговые документы каждого технологического этапа разработки ПС. Она включает следующие документы:

- Внешнее описание ПС (Requirements document).
- Описание архитектуры ПС (description of the system architecture), включая внешнюю спецификацию каждой ее программы (подсистемы).
- Для каждой программы ПС - описание ее модульной структуры, включая внешнюю спецификацию каждого включенного в нее модуля.
- Для каждого модуля - его спецификация и описание его строения (design description).
- Тексты модулей на выбранном языке программирования (program source code listings).
- Документы установления достоверности ПС (validation documents), описывающие, как устанавливалась достоверность каждой программы ПС и как информация об установлении достоверности связывалась с требованиями к ПС.

Документы установления достоверности ПС включают, прежде всего, документацию по тестированию (схема тестирова-

ния и описание комплекта тестов), но могут включать и результаты других видов проверки ПС, например, доказательства свойств программ. Для обеспечения приемлемого качества этой документации полезно следовать общепринятым рекомендациям и стандартам .

Документация второй группы содержит

- *Руководство по сопровождению ПС* (system maintenance guide), которое описывает особенности реализации ПС (в частности, трудности, которые пришлось преодолевать) и как учтены возможности развития ПС в его строении (конструкции). В нем также фиксируются, какие части ПС являются аппаратно- и программно-зависимыми.

Общая проблема сопровождения ПС - обеспечить, чтобы все его представления шли в ногу (оставались согласованными), когда ПС изменяется. Чтобы этому помочь, связи и зависимости между документами и их частями должны быть отражены в руководстве по сопровождению, и зафиксированы в базе данных управления конфигурацией.

Контрольные вопросы

1. Что такое менеджер программного средства?

2. Что такое ординарный пользователь программного средства?

3. Что такое администратор программного средства?

4. Что такое руководство по инсталляции программного средства?

5. Что такое руководство по управлению программным средством?

6. Что такое руководство по сопровождению программного средства?

2.3. Задание к лабораторной работе

Выберете тему создания программного продукта, в соответствии с пунктом 1.2 проведите документацию разрабатываемого программного средства.

Тема программного продукта, желательно, совпадает с темой ВКР.

К отчёту представить:

- документ в любом электронном формате, содержащий краткое описание пунктов (при отсутствии какого-либо пункта обоснуйте своё решение).

Отчёт о проделанной работе:
- собеседование по представленному документу.

2.4. Контрольные вопросы

1. Пользовательская документация программных средств.
2. Пользовательская документация программных средств ординарных пользователей ПС.
3. Пользовательская документация программных средств администратора.
4. ЕСПД функция, назначение, состав.
5. ГОСТ 19.101-77 ЕСПД. Виды программ и программных документов назначение, состав.
6. ГОСТ 19.105-78 ЕСПД. Общие требования к программным документам назначение, состав.
7. ГОСТ 19.201-78 ЕСПД. Техническое задание. Требования к содержанию и оформлению назначение, состав.
8. ГОСТ 19.503-79 ЕСПД. назначение, состав.
9. ГОСТ 19.404-79 ЕСПД. назначение, состав.

3. ЛАБОРАТОРНАЯ РАБОТА №3: ТЕХНОЛОГИЧЕСКИЕ ЭТАПЫ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

3.1. Технологические этапы создания ПС

Инструменты разработки программных средств.

В процессе разработки программных средств в той или иной мере используется компьютерная поддержка процессов разработки ПС. Это достигается путем представления хотя бы некоторых программных документов ПС(прежде всего, программ) на компьютерных носителях данных (например, дискетах)и предоставлению в распоряжение разработчика ПС специальных ПС или включенных в состав компьютера специальных устройств, созданных для какой-либо обработки таких документов. В качестве такого специального ПС можно указать компилятор с какого-либо языка программирования. Компилятор избавляет разработчика ПС от необходимости писать программы на языке компьютера, который для разработчика ПС был бы крайне неудобен, - вместо этого он составляет программы на удобном ему языке программирования, которые соответствующий компилятор автоматически переводит на язык компьютера. В каче- стве специального устрой-

ства,поддерживающего процесс разработки ПС, может служит эмулятор какого-либо языка.Эмулятор позволяет выполнять (интерпретировать) программы на языке, отличном от языка компьютера, поддерживающего разработку ПС, например на языке компьютера,для которого эта программа предназначена.

ПС, предназначенное для поддержки разработки других ПС, будем называть *программным инструментом* разработки ПС, а устройство компьютера, специально предназначенное для поддержки разработки ПС,будем называть *аппаратным инструментом* разработки ПС.

Инструменты разработки ПС могут использоваться в течении всего жизненного цикла ПС для работы с разными программными документами. Так текстовый редактор может использоваться для разработки практически любого программного документа. С точки зрения функций, которые инструменты выполняют при разработке ПС, их можно разбить на следующие четыре группы: ·

редакторы,·

анализаторы,·

преобразователи,·

инструменты, поддерживающие процесс выполнения программ.

Редакторы поддерживают конструирование (формирование) тех или иных программных документов на различных этапах жизненного цикла. Как уже упоминалось, для этого можно использовать один какой-нибудь универсальный текстовый редактор. Однако, более сильную поддержку могут обеспечить специализированные редакторы: для каждого вида документов - свой редактор. В частности, на ранних этапах разработки в документах могут широко использоваться графические средства описания (диаграммы, схемы и т.п.). В таких случаях весьма полезными могут быть графические редакторы. На этапе программирования (кодирования) вместо текстового редактора может оказаться более удобным синтаксически управляемый редактор, ориентированный на используемый язык программирования.

Анализаторы производят либо *статическую*обработку документов, осуществляя различные виды их контроля, выявление определенных их свойств и накопление статистических данных (например, проверку соответствия документов указанным стандартам), либо *динамический* анализ программ (например, с целью выявление распределения времени работы программы по программным модулям).

Преобразователи позволяют автоматически приводить до-

кументы к другой форме представления (например, форматеры) или переводить документ одного вида к документу другого вида (например, конверторы или компиляторы), синтезировать какой-либо документ из отдельных частей и т.п.

Инструменты, поддерживающие процесс выполнения программ, позволяют выполнять на компьютере описания процессов или отдельных их частей, представленных в виде, отличном от машинного кода, или машинный код с дополнительными возможностями его интерпретации. Примером такого инструмента является эмулятор кода другого компьютера. К этой группе инструментов следует отнести и различные отладчики. По-существу, каждая система программирования содержит программную подсистему периода выполнения, которая выполняет наиболее типичные для языка программирования программные фрагменты и обеспечивает стандартную реакцию на возникающие при выполнении программ исключительные ситуации (такую подсистему мы будем называть *исполнительной поддержкой*), - также можно рассматривать как инструмент данной группы.

Инструментальные среды разработки и сопровождения программных средств.

В настоящее время с каждой системой программирования связываются не отдельные инструменты (например, компилятор), а некоторая логически связанная совокупность программных и аппаратных инструментов поддерживающих разработку и сопровождение ПС на данном языке программирования или ориентированных на какую-либо конкретную предметную область. Такую совокупность будем называть *инструментальной средой разработки и сопровождения ПС*. Для таких инструментальных сред характерно,во-первых, использование как программных, так и аппаратных инструментов, и,во-вторых, определенная ориентация либо на конкретный язык программирования,либо на конкретную предметную область.

Инструментальная среда не обязательно должна функционировать на том компьютере, на котором должно будет применяться разрабатываемое с помощью ее ПС. Часто такое совмещение бывает достаточно удобным (если только мощность используемого компьютера позволяет это): не нужно иметь дело с компьютерами разных типов, в разрабатываемую ПС можно включать компоненты самой инструментальной среды. Однако, если компьютер, на котором должно применяться ПС, недоступен для разработчиков этого ПС (например, он постоянно занят другой работой, которую нельзя прерывать, или он находится еще в стадии

разработки), либо неудобен для разработки ПС, либо мощность этого компьютера недостаточна для обеспечения функционирования требуемой инструментальной среды, то применяется так называемый *инструментально-объектный подход*. Сущность его заключается в том, что ПС разрабатывается на одном компьютере, называемым *инструментальным*, а применяться будет на другом компьютере, называемым *целевым* (или *объектным*).

Различают три основных класса инструментальных средразработки и сопровождения ПС (рис.1): ·
среды программирования, ·
рабочие места компьютерной технологии,·
инструментальные системы технологии программирования.

Среда программирования предназначена в основном для поддержки процессов программирования (кодирования), тестирования и отладки ПС. *Рабочее место компьютерной технологии* ориентировано на поддержку ранних этапов разработки ПС (спецификаций) и автоматической генерации программ по спецификациям. *Инструментальная система технологии программирования* предназначена для поддержки всех процессов разработки и сопровождения в течение всего жизненного цикла ПС и ориентирована на коллективную разработку больших программных систем с длительным жизненным циклом. Для таких систем стоимость сопровождения обычно превышает стоимость разработки.

Рис. 1. Основные классы инструментальных сред разработки и сопровождения ПС.

Инструментальные среды программирования.

Инструментальные среды программирования содержат прежде всего текстовый редактор, позволяющий конструировать программы на заданном языке программирования, инструменты,

позволяющие компилировать или интерпретировать программы на этом языке, а также тестировать и отлаживать полученные программы. Кроме того, могут быть и другие инструменты, например,для статического или динамического анализа программ. Взаимодействуют эти инструменты между собой через обычные файлы с помощью стандартных возможностей файловой системы.

Различают следующие классы инструментальных сред программирования (см. рис. 2): ·

среды общего назначения,·

языково-ориентированные среды.

Инструментальные среды программирования *общего назначения* содержат набор программных инструментов, поддерживающих разработку программ на разных языках программирования (например, текстовый редактор, редактор связей или интерпретатор языка целевого компьютера) и обычно представляют собой некоторое расширение возможностей используемой операционной системы. Для программирования в такой среде на каком-либо языке программирования потребуются дополнительные инструменты, ориентированные на этот язык (например, компилятор).

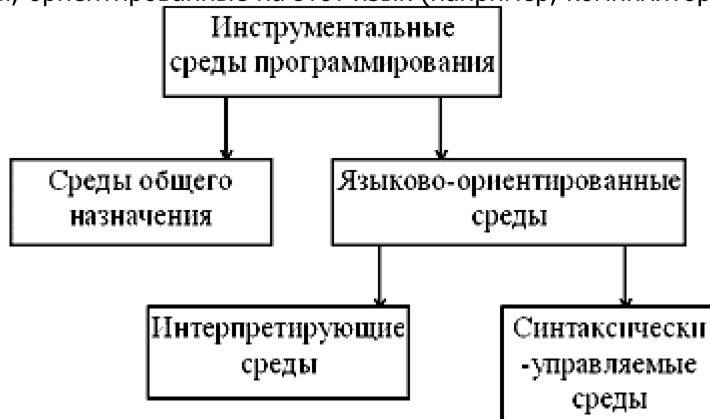


Рис.2. Классификация инструментальных сред программирования.

Языково-ориентированная инструментальная среда программирования предназначена для поддержки разработки ПС на каком-либо одном языке программирования и знания об этом языке существенно использовались при построении такой среды. Вследствие этого в такой среде могут быть доступны достаточно мощные возможности, учитывающие специфику данного языка. Такие среды разделяются на два подкласса: ·

интерпретирующие среды, ·

синтаксически-управляемые среды.

Интерпретирующая инструментальная среда программирования обеспечивает интерпретацию программ на данном языке программирования, т.е. содержит прежде всего интерпретатор языка программирования, на который эта среда ориентирована. Такая среда необходима для языков программирования интерпретирующего типа (таких, как Лисп), но может использоваться и для других языков (например, на инструментальном компьютере). *Синтаксически-управляемая*инструментальная среда программирования базируется на знании синтаксиса языка программирования, на который она ориентирована. В такой среде вместо текстового используется синтаксически-управляемый редактор, позволяющий пользователю использовать различные шаблоны синтаксических конструкций (в результате этого разрабатываемая программа всегда будет синтаксически правильной). Одновременно с программой такой редактор формирует (в памяти компьютера)ее синтаксическое дерево, которое может использоваться другими инструментами.

Понятие компьютерной технологии разработки программных средств и ее рабочие места.

Имеются некоторые трудности в выработке строгого определенияCASE-технологии (компьютерной технологии разработки ПС). CASE - это аббревиатура от английского Computer-Aided Software Engineering (Компьютерно-Помогаемая Инженерия Программирования). Но без помощи (поддержки) компьютера ПС уже давно не разрабатываются (используется хотя бы компилятор). В действительности, в это понятие вкладывается более узкий (специальный) смысл, который постепенно размывается (как это всегда бывает, когда какое-либо понятие не имеет строгого определения). Первоначально под CASE понималась инженерия ранних этапов разработки ПС (определение требований, разработка внешнего описания и архитектуры ПС) с использованием программной поддержки (программных инструментов). Теперь под CASE может пониматься и инженерия всего жизненного цикла ПС (включая и его сопровождение), но только в том случае, когда программы частично или полностью генерируются по документам, полученным на указанных ранних этапах разработки. В этом случае CASE-технология стала принципиально отличаться от ручной (традиционной) технологии разработки ПС: изменилось не только содержание технологических процессов, но и сама их совокупность.

В настоящее время *компьютерную технологию* разработки ПС можно характеризовать использованием·

программной поддержки для разработки графических требований и графических спецификаций ПС, ·

автоматической генерации программ на каком-либо языке программирования или в машинном коде (частично или полностью), ·

программной поддержки прототипирования.

Говорят также, что компьютерная технология разработки ПС является "безбумажной", т.е. рассчитанной на компьютерное представление программных документов. Однако, уверенно отличить ручную технологию разработки ПС от компьютерной по этим признакам довольно трудно. Значит, самое существенное в компьютерной технологии не выделено.

Главное отличие ручной технологии разработки ПС от компьютерной заключается в следующем. Ручная технология ориентирована на разработку документов, одинаково понимаемых разными разработчиками ПС, тогда как компьютерная технология ориентирована на обеспечение семантического понимания (интерпретации) документов программной поддержкой компьютерной технологии. Компьютерное представление документов еще не означает такого их понимание. Тогда как семантическое понимание документов дает программной поддержке возможность автоматической генерации программ, а необходимость обеспечения такого понимания делает желательными и различные графические формы входных документов. Именно это позволяет рационально изменить и саму совокупность технологических процессов разработки и сопровождения ПС.

Из проведенного обсуждения сущности компьютерной технологии можно понять и связанные с ней изменения в жизненном цикле ПС. Если при использовании ручной технологии основные усилия по разработке ПС делались на этапах собственно программирования (кодирования) и отладки (тестирования),то при использовании компьютерной технологии - на ранних этапах разработки ПС(определения требований и функциональной спецификации). При этом существенно изменился характер документации: вместо целой цепочки неформальных документов,ориентированной на передачу информации от заказчика (пользователя) к различным категориям разработчикам, формируются прототип ПС, поддерживающий выбранный пользовательский интерфейс, и формальные функциональные спецификации,достаточные для автоматического синтеза (генерации) программ ПС (или хотя бы значительной их части). При этом появилась возможность автоматической генерации части документации,

необходимой разработчикам и пользователям. Вместо ручного программирования (кодирования) - автоматическая генерация программ, что делает не нужной автономную отладку и тестирование программ: вместо нее добавляется достаточно глубокий автоматический семантический контроль документации.Существенно изменяется и характер сопровождения ПС: все изменения разработчиком-сопроводителем вносятся только в спецификации (включая и прототип), остальные изменения в ПС осуществляются автоматически.

С учетом сказанного *жизненный цикл ПС с использованием компьютерной технологии* можно представить следующей схемой (рис.3).

Рис. 3. Жизненный цикл программного средства при использовании компьютерной технологии.

Прототипирование позволяет заменить косвенное описание

взаимодействия между пользователем и ПС при ручной технологии (при определении требований к ПС и внешнем описании ПС) прямым выбором пользователем способа и стиля этого взаимодействия с фиксацией всех необходимых деталей. По-существу, на этом этапе производится точное описание пользовательского интерфейса, понятное программной поддержке компьютерной технологии, причем с ответственным участием пользователя: разработчик показывает пользователю на мониторе различные возможности и пользователь выбирает приемлемые для него варианты, пользователь с помощью разработчика вводит обозначения обрабатываемых им информационных объектов и операций над ними, он выбирает способ доступа к ним и связывает их с различными окнами,меню, виртуальными клавиатурами и т.п. Все это базируется на наличие в программной поддержке компьютерной технологии настраиваемой оболочки с обширной библиотекой заготовок различных фрагментов и деталей экрана. В результате указанных действий определяется оболочка ПС - верхний управляющий уровень ПС.Такое прототипирование, по-видимому, является лучшим способом преодоления барьера между пользователем и разработчиком.

Разработка спецификаций распадается на несколько разных процессов. Если исключить начальный этап разработки спецификаций (определение требований), то в этих процессах используются методы,приводящие к созданию формализованных документов, т. е. используются формализованные языки спецификаций. При этом широко используются графические методы спецификаций, приводящие к созданию различных схем и диаграмм, которые достаточно формализованно определяют структуру информационной среды и структуру управления ПС. К таким структурам привязываются фрагменты описания данных и программ, представленные на алгебраических языках спецификаций (например,использующие операционную или аксиоматическую семантику), или логических языках спецификаций (базирующихся на логическом подходе к спецификации программ).Такие спецификации позволяют в значительной степени или полностью автоматически генерировать программы.

Инструментальные системы технологии программирования.

Для компьютерной поддержки разработки и сопровождения больших ПС с продолжительным жизненным циклом используются инструментальные системы технологии программирования. *Инструментальная система технологии программирования* - это ин-

тегрированная совокупность программных и аппаратных инструментов, поддерживающая все процессы разработки и сопровождения больших ПС в течение всего его жизненного цикла в рамках определенной технологии. Из этого определения вытекают следующие основные черты этого класса компьютерной поддержки: ·

комплексность, ·
ориентированность на коллективную разработку, ·
технологическая определенность, ·
интегрированность.

Комплексность компьютерной поддержки означает, что она охватывает все процессы разработки и сопровождения ПС и что продукция этих процессов согласована и взаимоувязана. Тем самым, система в состоянии обеспечить, по-крайней мере, контроль полноты (комплектности)создаваемой документации (включая набор программ) и согласованности ее изменения (версионности). Тот факт, что компьютерная поддержка охватывает и фазу сопровождения ПС, означает, что система должна поддерживать работу сразу с несколькими вариантами ПС, ориентированными на разные условия применения ПС и на разную связанную с ним аппаратуру, т.е. должна обеспечивать управление конфигурацией ПС.

Ориентированность на коллективную разработку означает,что система должна поддерживать управление (management) работой коллектива и для разных членов этого коллектива обеспечивать разные права доступа к различным фрагментам продукции технологических процессов.

Технологическая определенность компьютерной поддержки означает, что ее комплексность ограничивается рамками какой-либо конкретной технологии программирования. Инструментальные системы технологии программирования представляют собой достаточно большие и дорогие ПС, чтобы как-то была оправданна их инструментальная перегруженность. Поэтому набор включаемых в них инструментов тщательно отбирается с учетом потребностей предметной области,используемых языков и выбранной технологией программирования.

Интегрированность компьютерной поддержки означает·
интегрированность по данным, ·
интегрированность по пользовательскому интерфейсу, ·
интегрированность по действиям (функциям),

Интегрированность по данным означает, что инструменты действуют в соответствии с фиксированной информационной

схемой(моделью) системы, определяющей зависимость различных используемых в системе фрагментов данных (информационных объектов) друг от друга. *Интегрированность по пользовательскому интерфейсу* означает, что все инструменты объединены единым пользовательским интерфейсом. *Интегрированность по действиям* означает,что, во-первых, в системе имеются общие части всех инструментов и, во-вторых,одни инструменты при выполнении своих функций могут обращаться к другим инструментам.

С учетом обсужденных свойств инструментальных систем технологии программирования можно выделить три их основные компоненты:·

база данных разработки (репозиторий),·

инструментарий, ·

интерфейсы.

Репозиторий - центральное компьютерное хранилище информации, связанной с проектом (разработкой) ПС в течении всего его жизненного цикла. *Инструментарий* - набор инструментов, определяющий возможности, предоставляемые системой коллективу разработчиков. Обычно этот набор является открытым: помимо минимального набора (*встроенные инструменты*),он содержит средства своего расширения (*импортированными инструментами*),- и структурированным, состоящим из некоторой общей части всех инструментов (*ядра*)и структурных (иногда иерархически связанных) классов инструментов. *Интерфейсы*разделяются на пользовательский и системные. *Пользовательский* интерфейс обеспечивает доступ разработчикам к инструментарию (командный язык и т.п.),реализуется *оболочкой* системы. Системные интерфейсы обеспечивают взаимодействие между инструментами и их общими частями. Системные интерфейсы выделяются как архитектурные компоненты в связи с открытостью системы - их обязаны использовать новые (*импортируемые*) инструменты, включаемые в систему.

Самая общая архитектура инструментальных систем технологии программирования представлена на рис. 4.

Различают два класса инструментальных систем технологии программирования: инструментальные системы поддержки проекта и языково-зависимые инструментальные системы. *Инструментальная система поддержки проекта* - это открытая система, способная поддерживать разработку ПС на разных языках программирования после соответствующего ее расширения программными инструментами, ориентированными на выбранный язык. Такая система содержит ядро (обеспечивающее, в частности, доступ к

репозиторию), набор инструментов, поддерживающих управление (management) разработкой ПС,независимые от языка программирования инструменты, поддерживающие разработку ПС(текстовые и графические редакторы, генераторы отчетов и т.п.), а также инструменты расширения системы. *Языково-зависимая инструментальная система* -это система поддержки разработки ПС на каком-либо одном языке программирования,существенно использующая в организации своей работы специфику этого языка. Эта специфика может сказываться и на возможностях ядра (в том числе и на структуре репозитория), и на требованиях к оболочке и инструментам. Примером такой системы является среда поддержки программирования на Аде (APSE).

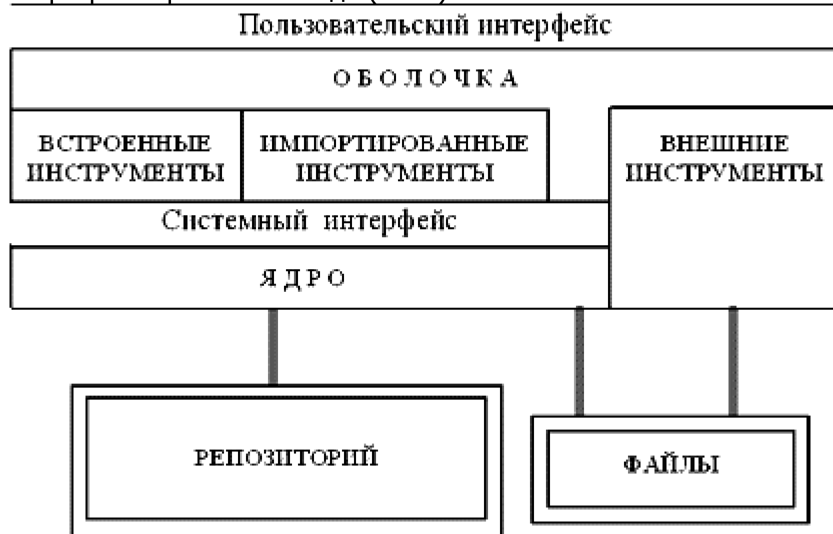


Рис. 4. Общая архитектура инструментальных систем технологии программирования.

3.2. Финансирование проекта

При подготовке бизнес-плана может быть использовано как универсальное, специальное, так и специализированное программное обеспечение.

Программное обеспечение для бизнес-планирования:

- Универсальное;
- Специализированное;
- Пакеты офисных программ;
- Независимые прикладные программы;

–Для создания бизнес-плана целиком;
–Для расчета финансовой части бизнес-плана;
–Текстовые редакторы;
–Электронные таблицы;
–Самостоятельные программные продукты;
–Модули, макросы и шаблоны для Ms Office;
–Статистические программы.

Специальное

К универсальному программному обеспечению относят текстовые редакторы и электронные таблицы, как в рамках офисных пакетов, так и отдельных, самостоятельных программ. К первым относятся Microsoft Office, а так же офисные пакеты от Sun Microsystems – Star Office, офисный пакет от канадской фирмы Corel Corp. - WordPerfect Office, пакет офисных программ OpenOffice.org (распространяется на бесплатной основе), созданный в рамках проекта Open Source Projects и другие.

Ко второй группе универсального программного обеспечения относят независимые текстовые редакторы и электронные таблицы. Наиболее популярными программными продуктами среди текстовых редакторов являются PolyEdit, AbiWord, PatriotXP, CryptEdit. Для подготовки финансовой части бизнес-плана можно использовать – Formula One, Tabad, SuperCalc и др.

Необходимо отметить, что если бизнес-план создается в офисном пакете, то нет необходимости в какой-либо дополнительной программе. Текстовая часть создается в текстовом редакторе пакета, а расчетно-аналитическая – в электронных таблицах того же офисного пакета. При этом данные из электронных таблиц могут быть свободно импортированы в текстовый редактор. В случае использования независимых программных продуктов необходимо либо обеспечить или совместимость между приложениями, либо расчетно-аналитическую часть бизнес-плана создавать отдельно (без импортирования расчетно-аналитических данных в текстовый редактор).

Как показала практика, в большинстве случаев разработка бизнес-плана с использованием универсального программного обеспечения ведется в офисном пакете Microsoft Office. Этот обстоятельство говорит о том, что пользователи считают его наиболее удобным как для подготовки текстовой, так и расчетно-аналитической составляющих бизнес-плана. В некоторых случаях в нем же разрабатываются и презентации (для привлечения и

первичного ознакомления инвесторов и партнеров).

Для подготовки расчетно-аналитической части бизнес-плана могут быть использованы и специальное программное обеспечение для статистических вычислений и анализа, такое как SPSS, StatSoft Statistica, Statit Professional и др.. Статистические программы обладают мощным аппаратом статистических процедур, благодаря которому, можно быстро обработать исходные данные и получить информацию, необходимую при бизнес-планировании, принятии решений и подготовки финансовой части бизнес-плана.

Работа со специализированными программными продуктами, как правило, осуществляется в четыре этапа. На первом этапе производится анализ условий разработки и осуществления проекта, формирование и ввод необходимых исходных данных для проведения расчетов, анализа и если позволяет программное обеспечение, то и текстовой составляющей бизнес-плана. На втором этапе с помощью инструментария программного обеспечения осуществляется автоматический расчет финансово-экономических показателей, формируются инвестиционно-финансовая отчетность проекта. Затем проводится финансово-экономический анализ привлекательности инвестиционного проекта, анализ его чувствительности к колебаниям конъюнктуры рынка и изменениям макроэкономических условий деятельности. На заключительном этапе программное обеспечение позволяет автоматически подготовить или финансовую часть бизнес-плана, или ТЭО, или бизнес-план целиком в зависимости от возможностей программы.

На данный момент на российском рынке специализированного программного обеспечения для расчета и анализа финансово-инвестиционной части бизнес-плана и бизнес-плана целиком существует более десятка как отечественных, так и зарубежных программных продуктов.

Среди отечественных можно выделить следующие:

- Project Expert консалтинговой фирмы «Про-Инвест Консалтинг»;
- Инэк-Аналитик консалтинговой фирмы «Инэк»;
- ТЭО-Инвест Института проблем управления РАН;
- Альт-Инвест исследовательско-консультационной фирмы «Альт»;
- Business Plan PL консалтинговой фирмы ЗАО «Общество финансового и экономического развития предприятий РОФЭР»;
- «Мастерская бизнес-планирования» фирмы ООО «Корпоративный Менеджмент»;

- «Мастер проектов» консалтинговой фирмы «Консультационная группа "Воронов и Максимов"»;

и другие.

Среди наиболее известных зарубежных программ можно выделить:

- COMFAR (Computer Model for Feasibility Analysis and Reporting), разработчиком которой является Комитет по промышленному развитию при ООН (UNIDO).

Представленные программные продукты позволяют:

- детально описать и провести финансово-экономический анализ проекта;
- разрабатывать различные варианты, схемы финансирования и привлечения инвестиций и выбрать наиболее оптимальный вариант;
- проводить анализ коммерческой эффективности инвестиционного проекта в целом;
- смоделировать различные сценарии развития инвестиционного проекта, варьируя значения параметров, влияющих на его финансовые результаты;
- сформировать бюджет инвестиционного проекта с учетом изменений внешний среды организации (инфляции, налогового окружения, ставки рефинансирования ЦБ РФ);
- произвести анализ чувствительности к изменению различных исходных параметров на эффективность проекта;
- в зависимости от возможности конкретной программы подготовить к печати документацию - инвестиционно-финансовую часть бизнес-плана, ТЭО или бизнес план целиком.

Кроме указанных характеристик, у представленных программных продуктов имеются и другие специальные возможности.

Рассмотрим общие достоинства, недостатки и возможности универсального, специального и специализированного программного обеспечения для разработки бизнес-плана.

Сравнение характеристик универсального, специального и специализированного программного обеспечения для бизнес-планирования представлено в Приложение 1.

СПЕЦИАЛИЗИРОВАННОЕ ПРОГРАММНОЕ ОБЕСПЕЧЕНИ ДЛЯ РАЗРАБОТКИ БИЗНЕС-ПЛАНОВ

1. Для разработки бизнес-плана целиком (текстовая и расчетно-аналитическая составляющие).
2. Для подготовки инвестиционно-финансовой части бизнес-

плана и ТЭО (расчетно-аналитическая составляющая).

3. Самостоятельные программные продукты:

- Модули, макросы и шаблоны для Ms Office;
- Модули и макросы для Ms Excel;
- Project Expert;
- COMFAR;
- Business Plan PL;
- Мастерская бизнес-планирования;
- Инэк-Аналитик;
- ТЭО-Инвест;
- Альт-Инвест;
- Мастер проектов.

Приобретать дорогостоящее специальное программное обеспечения – статистические программные продукты только для разработки одного бизнес-планов или только для бизнес–планирования – нецелесообразно и затратно. Если данное программное обеспечение имеется у организации или оно будет востребовано после приобретения (не только для бизнес-планирования, но и для статистических расчетов, анализа, обработки результатов исследований), то оно может использовано для разработки расчетно-аналитической части бизнес-плана.

Выбор в пользу универсального программного обеспечения (наиболее вероятно, что это будет офисный пакет Microsoft Office или бесплатная альтернатива - OpenOffice.org), может быть сделан исходя из следующих факторов:

- бизнес-планирование для организации – это или разовая акция;
- инвестор требует разработать бизнес-план в соответствии со своей методикой;
- нет возможности приобрести специализированное программное обеспечение, из–за недостаточности финансовых ресурсов;
- бизнес-план рассчитан в первую очередь на внутреннее использование;
- проект не соответствует стандартной схеме «покупка оборудования – производство»
- в проекте присутствует четко выраженная специфика, которая не может быть учтена в специализированных программах для бизнес-планирования;
- нет достаточного ресурса времени на обучение специали-

зированной программой для бизнес-планирования;

- разработчик бизнес-плана имеет достаточный успешный опыт самостоятельной разработки бизнес-плана без применения специализированного оборудования.

Выбор в пользу специализированного программного обеспечения может быть сделан в случае если:

- бизнес-план рассчитан на привлечение внешних инвестиций;
- инвестор рекомендует специальное программное обеспечения для разработки бизнес-плана;
- бизнес-планирование производится периодически или постоянно т.е. стало частью управленческой культуры организации;
- необходимо разработать бизнес-план в соответствии с международными стандартами и методиками;
- есть достаточные финансовые и временные ресурсы для приобретения программного обеспечения и обучение персонала.

Среди специализированных программных продуктов для разработки бизнес-планов организаций можно порекомендовать такие программы как Мастерская бизнес-планирования и Business Plan PL, реализующего проекты внутри России. Программу Project Expert можно рекомендовать для крупного и среднего бизнеса, а так же тем, кто занимается разработкой бизнес-планов и ТЭО с привлечением международных инвестиций.

Специализированное программное обеспечение для подготовки инвестиционно-финансовой части бизнес-плана и ТЭО, разработанное в виде модулей и макросов для Ms Exce,l в большей степени ориентировано на экспертов и профессиональных консультантов. Методики расчетов и оценок в данных программах учитывают требования международных и российских стандартов. Однако к недостаткам можно отнести отсутствие возможности подготовить бизнес-план целиком, то есть не только его расчетно-аналитическую составляющую, но и текстовую.

Применение зарубежных специализированных программ для бизнес-планирования, например COMFAR, затруднено, в следствие того, что они в недостаточной степени отражают российское налоговое законодательство, без учета которого разработать бизнес-план хоть и возможно, однако его реализация в условиях российской экономики будет затруднена.

Необходимо отметить, что для обеспечения гибкости и надежности бизнес-планирования целесообразно применять в комплексе универсальное и специализированное программное обеспечение для бизнес-планирования, что обеспечить комплекс-

ность решения и улучшить качество итогового документа - бизнес-плана.

С учетом требований к разработке программы развития (бизнес–плана), а также по результатам проведенного анализа нами было рекомендовано к использованию в зависимости от финансовых возможностей организации воспользоваться:

1). При отсутствии достаточного финансирования - воспользоваться программным пакетом Microsoft Office с использованием для подготовки текстовой части – текстовым редактором Microsoft Office Word и для подготовки расчетно-аналитической части – электронными таблицами Microsoft Office Excel, т.е. подготавливать бизнес-план полностью в ручную.

2). При наличии финансирования – выбрать из таких программных продуктов как «Мастерская бизнес-планирования», либо «Project Expert».

Рекомендуется составление финансовой части бизнес-планов с помощью программного продукта Project Expert фирмы "ПРО-ИНВЕСТ КОНСАЛТИНГ", или пакетов "Альт-Инвест" фирмы "Альт", "АНАЛИТИК" - фирмы "ИНЭК". Необходимо отметить, что само по себе использование указанных или иных программных продуктов еще не гарантирует составления бизнес-плана на качественном уровне,

PROJECT EXPERT ПОМОЖЕТ:

Project Expert дает возможность проанализировать несколько вариантов достижения целей развития предприятия и выбрать из них оптимальный. При этом можно оценить запас прочности бизнеса как производную риска изменения важнейших факторов, влияющих на реализацию проекта. Программа также позволяет оценить, как исполнение бизнес-плана повлияет на эффективность деятельности предприятия, рассчитать срок окупаемости проекта, спрогнозировать общие показатели эффективности для группы инвестиционных проектов, финансируемых из общего бюджета.

Project Expert позволяет создать безупречный бизнес-план предприятия, соответствующий международным стандартам (МСФО), подготовить предложения для региональной инвестиционной программы и/или стратегического инвестора, определив для каждого из участников общий экономический эффект от реализации инвестиционного проекта и эффективность инвестиций в него.

С Project Expert можно разработать схему финансирования инвестиционного проекта предприятия с учетом будущих потреб-

ностей в денежных средствах на основе прогноза движения денежных средств на всем периоде планирования, выбрать источники и условия привлечения средств для реализации бизнес-плана, оценить возможные сроки и графики возврата кредита. Вы также сможете спроектировать структуру капитала предприятия и оценить стоимость бизнеса.

Project Expert позволяет проанализировать планируемую структуру затрат и прибыльность отдельных подразделений и видов продукции, определить минимальный объем выпуска продукции и предельные издержки, подобрать производственную программу и оборудование, схемы закупок и варианты сбыта.

Project Expert поможет проконтролировать выполнение бизнес-плана предприятия, сравнивая в ходе реализации его плановые и фактические показатели.

Project Expert позволяет гибко учитывать изменения в экономическом окружении и оперативно отражать изменения. Программа рекомендована к использованию Минэкономики России и структурами регионального уровня как стандартный инструмент для разработки бизнес-планов предприятий. В основу Project Expert положена методика UNIDO по оценке инвестиционных проектов и методика финансового анализа, определенная международными стандартами IAS .

При построении в Project Expert финансовой модели, разработав на ее основе бизнес-план компании, автоматически создается хорошо структурированный и понятный отчет.

Благодаря возможностям динамического обмена данными с Excel , передачи отчетов в Word и сохранения их в формате HTML , Project Expert может использоваться и как самостоятельная аналитическая программа, и как составная часть информационно-аналитической системы предприятия.

Система выпускается в нескольких версиях: от Standard – для небольших предприятий – до Professional и PIC - Holding – для крупных корпораций и холдингов – и существует в локальном и сетевом вариантах.

Среди пользователей Project Expert свыше 4500 организаций:

1) государственные учреждения: Минэкономразвития России, Российский фонд федерального имущества, Администрации Ямало-Ненецкого и Ханты-Мансийского автономных округов, Краснодарского края, Томской области и др.

2) банки: Автобанк, Сбербанк РФ, Московский индустриальный банк, Банк Менатеп СПб, Международный банк развития,

Россельхозбанк.

3) предприятия: Лукойл-Пермь, ГАММА, РАО ЕЭС России, ПФГ Росвагонмаш, НК ЮКОС, ТЭК Итера, ТД Русавтопром, Группа "Сибирский алюминий", "Протек", "АвтоВАЗ", "ТЕТРА ПАК", "Хьюлетт-Паккард", "Алмазы России-Саха", "ТИГИ-КНАУФ", "Объединенная металлургическая компания", Клинский пивокомбинат и др.

4) образовательные учреждения: МГУ, ГУ ВШЭ, МГТУ им. Н.Э. Баумана, Российская экономическая академия им. Г.В. Плеханова, Финансовая академия при Правительстве РФ и др.

Зарегистрированным пользователям компания предоставляет:

- консультации ведущих специалистов компании в режиме "горячей линии" по телефону и электронной почте;
- информационную on - line поддержку на сайте компании;
- электронный учебник;
- возможность сертификации по программному продукту;
- поставку новых версий по льготным ценам;
- регулярную информацию о новых разработках компании.

3.3. Задание к лабораторной работе

В соответствии с исходными данными (ЛР 1) разработайте технологические этапы создания разрабатываемого программного продукта.

К отчёту представить:

- документ в любом электронном формате, содержащий:

А) краткое описание технологических этапов создания ПС (при отсутствии какого-либо пункта обоснуйте своё решение);

Б) краткое описание и обоснование принятой бизнес–плана.

Отчёт о проделанной работе:

- собеседование по представленному документу.

3.4. Контрольные вопросы

1. Функции, которые выполняют при разработке ПС редакторы.

2. Функции, которые выполняют при разработке ПС анализаторы.

3. Функции, которые выполняют при разработке ПС преобразователи.

4. Назначение Языково-ориентированной инструментальной среды программирования.

5. Назначение интерпретирующей среды программирования.

6. Назначение синтаксически-управляемой среды программирования.

7. Понятие и характеристика компьютерной технологии разработки.

8. Программная поддержка разработки графических требований и графических спецификаций ПС.

9. Автоматическая генерация программ на каком-либо языке программирования или в машинном коде (частично или полностью).

10. Программная поддержка прототипирования.

11. Схема жизненного цикла ПС с использованием компьютерной технологии.

4. ЛАБОРАТОРНАЯ РАБОТА №4: СТРУКТУРНО-МОДУЛЬНОЕ ПРОЕКТИРОВАНИЕ HIPO-СХЕМЫ

4.1. Структурно-модульное проектирование

Основные подходы при программировании:

- Нисходящее проектирование;
- Модульное программирование;
- Структурное программирование.

Нисходящее проектирование

Метод нисходящего проектирования предполагает последовательное разложение общей функции обработки данных на простые функциональные элементы ("сверху-вниз").

В результате строится иерархическая схема, отражающая состав и взаимоподчиненность отдельных функций, которая носит название *функциональная структура алгоритма (ФСА)* приложения.

Последовательность действий по разработке функциональной структуры алгоритма приложения:

- определяются цели автоматизации предметной области и их иерархия *(цель-подцель);*
- устанавливается состав приложений (задач обработки), обеспечивающих реализацию поставленных целей;
- уточняется характер взаимосвязи приложений и их основные характеристики (информация для решения задач, время и периодичность решения, условия выполнения и др.);
- определяются необходимые для решения задач функции обработки данных;
- выполняется декомпозиция функций обработки до необходимой структурной сложности, реализуемой предполагаемым инструментарием.

Подобная структура приложения (рис. 1) отражает наиболее важное - состав и взаимосвязь функций обработки информации для реализации приложений, хотя и не раскрывает логику выполнения каждой отдельной функции, условия или периодичность их вызовов.

Разложение должно носить строго *функциональный* характер, т.е. отдельный элемент ФСА описывает *законченную содержательную функцию* обработки информации, которая предполагает определенный способ реализации на программном уровне.

Функции ввода-вывода информации рекомендуется отделять от функций вычислительной или логической обработки данных.

По частоте использования функции делятся на:

- однократно выполняемые;
- повторяющиеся.

Степень детализации функций может быть различной, но иерархическая схема должна давать представление о составе и структуре взаимосвязанных функций и общем алгоритме обработки данных. Широко используемые функции приобретают ранг стандартных (встроенных) функций при проектировании внутренней структуры программного продукта.

Пример. Некоторые функции, например Ф2, далее неразложимы на составляющие: они предполагают непосредственную программную реализацию.

Другие функции, например Ф1, Фm, могут быть представлены в виде структурною объединения более простых функций, например Ф11, Ф12 и т.д. Для всех функций-компонентов осуществляется самостоятельная программная реализация; состав-

ные функции (типа Ф1, Фm) реализуются как программные модули, управляющие функциями-компонентами. например, в виде программ-меню.

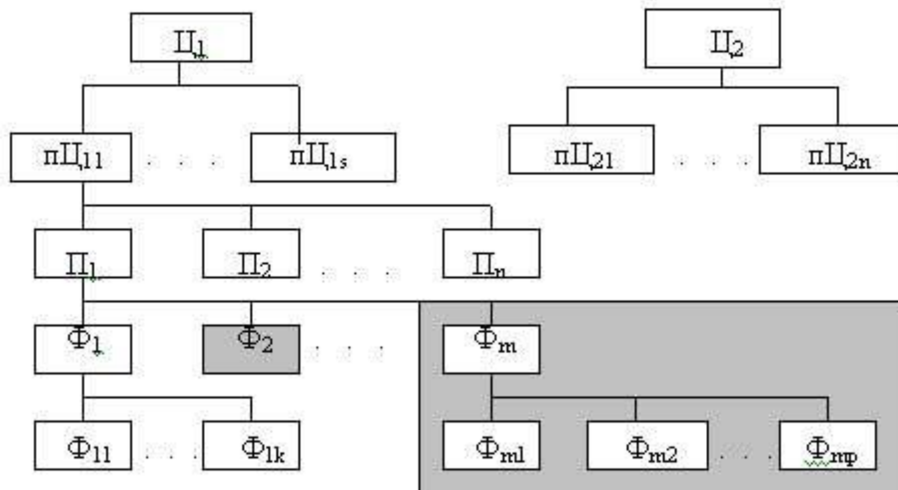


Рис. 1. Функциональная структура приложения:
Ц - цель; пЦ - подцель; П - приложение; Ф - функция

МОДУЛЬНОЕ ПРОГРАММИРОВАНИЕ

Свойства модуля

Модульное программирование основано на понятии **модуля** - логически взаимосвязанной совокупности функциональных элементов, оформленных в виде отдельных программных модулей.

Модуль характеризуют:

один вход и один выход - на входе программный модуль получает определенный набор исходных данных, выполняет содержательную обработку и возвращает один набор результатных данных, т.е. реализуется стандартный принцип IPO (Input - Process - Output) - *вход-процесс-выход;*

функциональная завершенность - модуль выполняет перечень регламентированных операций для реализации каждой отдельной функции в полном составе, достаточных для завершения начатой обработки;

логическая независимость - результат работы программного модуля зависит только от исходных данных, но не зависит от работы других модулей;

слабые информационные связи с другими программными

модулями - обмен информацией между модулями должен быть по возможности минимизирован;

обозримый по размеру и сложности программный элемент.

Таким образом, модули содержат определение доступных для обработки данных, операции обработки данных, схемы взаимосвязи с другими модулями.

Каждый модуль состоит из спецификации и тела. *Спецификации* определяют правила использования модуля, а *тело* - способ реализации процесса обработки.

Модульная структура программных продуктов

Принципы модульного программирования программных продуктов во многом сходны с принципами нисходящего проектирования. Сначала определяются состав и подчиненность функций, а затем - набор программных модулей, реализующих эти функции.

Однотипные функции реализуются одними и теми же модулями. Функция верхнего уровня обеспечивается **главным** модулем; он управляет выполнением нижестоящих функций, которым соответствуют **подчиненные** модули.

При определении набора модулей, реализующих функции конкретного алгоритма, необходимо учитывать следующее:

каждый модуль вызывается на выполнение вышестоящим модулем и, закончив работу, возвращает управление вызвавшему его модулю;

принятие основных решений в алгоритме выносится на максимально "высокий" по иерархии уровень;

для использования одной и той же функции в разных местах алгоритма создается один модуль, который вызывается на выполнение по мере необходимости. В результате дальнейшей детализации алгоритма создается *функционально-модульная* схема (ФМС) алгоритма приложения, которая является основой для программирования (рис. 2).

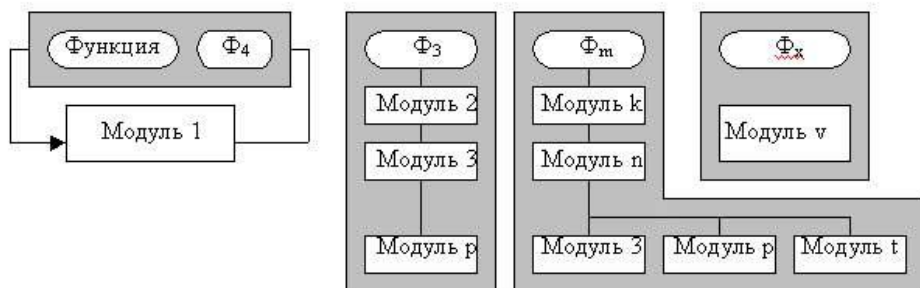


Рис. 2. Функционально-модульная структура приложения

Пример. Некоторые функции могут выполняться с помощью одного и того же программного модуля (например, функции Ф1 и Ф2).

Функция Ф3 реализуется в виде последовательности выполнения программных модулей.

Функция Фm реализуется с помощью иерархии связанных модулей.

Модуль n управляет выбором на выполнение подчиненных модулей.

Функция Фх реализуется одним программным модулем.

Состав и вид программных модулей, их назначение и характер использования в программе в значительной степени определяются инструментальными средствами. Например, применительно к средствам СУБД отдельными модулями могут быть:

экранные формы ввода и/или редактирования информации базы данных;

отчеты генератора отчетов;

макросы;

стандартные процедуры обработки информации;

меню, обеспечивающее выбор функции обработки и др.

Алгоритмы большой сложности обычно представляются с помощью схем двух видов:

обобщенной схемы алгоритма - раскрывает общий принцип функционирования алгоритма и основные логические связи между отдельными модулями на уровне обработки информации (ввод и редактирование данных, вычисления, печать результатов и т.п.);

детальной схемы алгоритма представляет содержание каждого элемента обобщенной схемы с использованием *управляющих структур* в блок-схемах алгоритма, *псевдокода либо алгоритмических языков* высокого уровня.

Наиболее часто детально проработанные алгоритмы изображаются в виде блок-схем согласно требованиям структурного программирования; при их разработке используются условные обозначения согласно ГОСТ 19.003-80 ЕСПД (Единая система программной документации). Обозначения условные графические, ГОСТ 19.002-80 ЕСПД. Схемы алгоритмов и программ. Правила обозначения.

СТРУКТУРНОЕ ПРОГРАММИРОВАНИЕ

Структурное программирование основано на модульной

структуре программного продукта и типовых **управляющих структурах** алгоритмов обработки данных различных программных модулей (рис. 3).

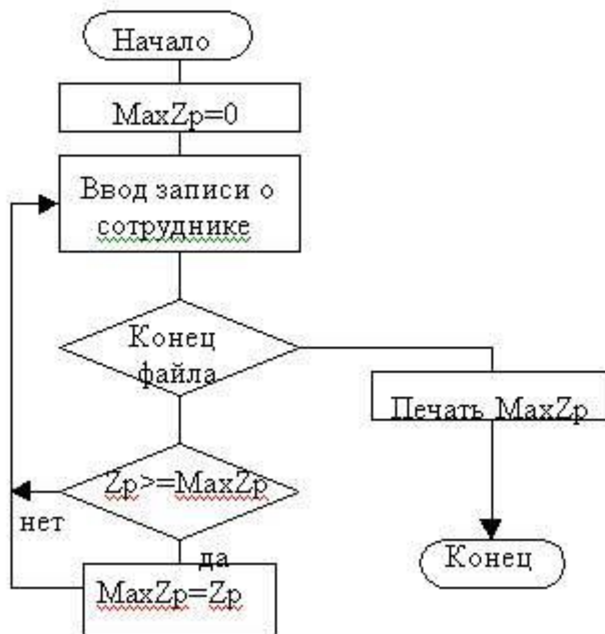


Рис. 3. Блок-схема алгоритма поиска в базе данных.

В любой типовой структуре блок, кроме условного, имеет только один вход и выход, безусловный переход на блок с нарушением иерархии запрещен (оператор типа *GoTo* в структурном программировании не используется). Виды основных управляющих структур алгоритма приведены в табл.1.

Пример. Алгоритм поиска в базе данных сведений о максимальном окладе сотрудников (рис. 4).

Таблица 1. Управляющие структуры алгоритмов

Типы управляющей структуры	Применение управляющей структуры
Последовательность Блок 1 Блок 2 Конец	Последовательность включает фиксированный перечень блоков (операторов). Каждый очередной блок обрабатывается после завершения предыдущего без дополнительных условий. Для изменения порядка обработки блоков редактируется последовательность выполняемых
Альтернатива (условие выбора) Начало Да Условие Нет Альтернатива1 Альтернатива2 Конец	В блоке Условие содержится условие выбора альтернативы обработки. Каждая альтернатива выполняется 1 раз; выполнение одной из двух альтернатив - обязательно. Развитие данного типа структуры является множественная альтернатива, когда последовательно проверяются условия выполнения определенных альтернатив. Если очередное условие истинно, обрабатывается соответствующая ему альтернатива, после чего происходит выход. В противном случае - переход к проверке условия следующей альтернативы. Если ни одно из условий не выполнилось, происходит выход.

Цикл ("пока") Начало Условие Нет Да Тело цикла Конец	В блоке Условие задается условие тела цикла - определенной обработки. Если условие не выполняется, цикл прерывается и осуществляется выход. Условие может содержать счетчик повторений тела цикла либо логическое условие. Тело цикла - произвольная последовательность блоков (операторов) обработки

4.2. HIPO-схемы

Технология HIPO (Hierarchical Input Process Output Diagrams) - многоуровневая технология проектирования и документирования программных моделей, основанная на использовании системы шаблонов, бланков и типовых диаграмм. **HIPO-диаграммы (Hierarchy Input-Processing-Output)**

Данный метод используется после подготовки структурной схемы модулей и направлен на доработку взаимодействия отдельных элементов модуля. При этом рассматриваются входные и выходные данные, а также процессы их преобразования. Общий вид данных схем можно представить так:

Вход	Процесс обработки	Выход
Файл CLIENT (атрибуты, запись)	Подготовка файла	
Запись на корректировку ⇒ (атрибуты одной записи)	Анализ корректности записи ⇒ Изменение файла CLIENT ⇒	Сообщение об ошибке Файл CLIENT с изменениями

Пример применения технологии

Ход работы:

Формулировка задачи:

- с помощью HIPO-технологии составить внешние спецификации для комплекса программ решения следующей задачи: «Решение задач по физике. Законы идеального газа и уравнение состояния».

Описание требований к проекту.

Проект представляет собой обучающую систему, содержащую теоретический материал по теме о законах идеального газа и об уравнении состояния. Также в составе программного продукта должен содержаться модуль для решения физических задач, основанных на уравнении состояния идеального газа.

Кроме этого, в программе должна быть предусмотрена возможность тестирования пользователя с целью оценки уровня его знаний в данной области.

Данный программный продукт рассчитан в основном на использование учениками средних общеобразовательных школ, поэтому еще одним требованием является разработка «дружественного» интерфейса с пользователем; программа должна быть максимально проста в управлении, и содержать лишь необходимые функции, плюс исчерпывающую справку по самой программе.

Схема состава разложения и IPO-диаграммы.

Далее приводится альбом связных схем, составленный по HIPO-технологии и включающий:

-оглавление альбома документации и условные обозначения;

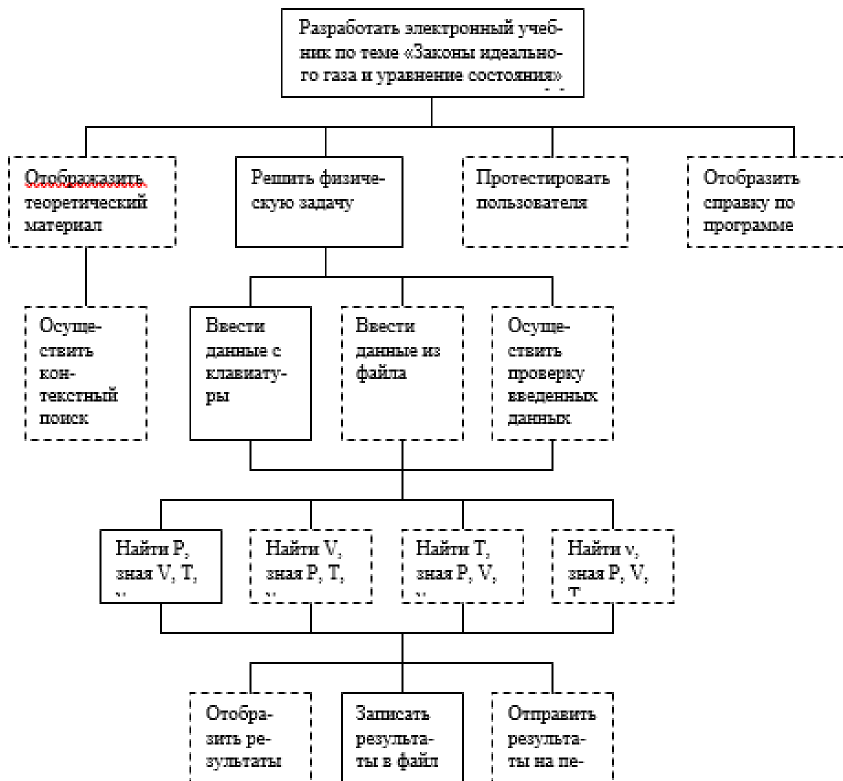
-схему состава разложения;

-набор IPO-диаграмм, реализующих элементы структуры.

Исходное описание системы Физика Схема Оглавление Номер: 0.0	**Оглавление альбома** документов и условные обозначения	Лист: 1 Автор: Васильцов А. Отдел: каф. ПОИС Дата: 4.03.2006

Обозначение	Наименование	Лист	Примечание
0.0	Оглавление альбома документации и условные обозначения	1	
0.1	Схема состава разложения	2	
1.1	Разработать электронный учебник по теме «Законы идеального газа и уравнение состояния»	3	
2.2	Решить физическую задачу	4	
3.2	Ввести данные с клавиатуры	5	
4.1	Найти P, зная V, T, v	6	
5.2	Сохранить результаты в файл	7	

Исходное описание Системы Физика Схема Состава Номер: 0.1	Схема состава разложения	Лист: 2 Автор: Васильцов А. Отдел: каф. ПОИС Дата: 4.03.2006



Исходное описание Системы Физика Схема Программа Номер: 1.1	Разработать электронный учебник по теме «Законы идеального газа и уравнение состояния»	Лист: 3 Автор: Васильцов А. Отдел: каф.ПОИС Дата: 4.03.2006

При разработке крупных программных продуктов, требующих привлечения существенных материальных и человеческих ресурсов правильная декомпозиция сущностей поставленной задачи имеет огромное значение и влияние на результат (как на качество результата, так и на скорость его достижения). Поэтому не

следует пренебрегать методикой составления внешних спецификаций для корректного взаимодействия между группами разработчиков, решающими каждая свою задачу, а также между разработчиками в каждой из групп, реализующими различные функции единой задачи.
HIPO-технология, благодаря своей четкой стандартизации и наглядности, способна значительно ускорить процесс составления внешних спецификаций (как залог эффективного и согласованного взаимодействия группы разработчиков при создании общего продукта), а строгое структурирование призвано без особых усилий обнаруживать и устранять ошибки, неточности и недостатки анализа.
В результате HIPO-разложения данного гипотетического создаваемого продукта можно обнаружить такие преимущества конкретной декомпозиции предметной области:
1) наблюдается частичная информационная независимость между отдельно разрабатываемыми частями программы, особенно на верхних уровнях, что безусловно положительно влияет на общий результат и время его достижения за счет отсутствия необходимости согласования типов и форматов данных, передаваемых между модулями; 2) построенная схема состава разложения имеет относительно «квадратную» форму (5 уровней в глубину и 4 уровня в ширину), из чего можно сделать вывод об относительной сбалансированности в соотношении «время-человеч.ресурсы». Конечно, о данном балансе нет смысла говорить в таких критических случаях, если либо время выполнения проекта ограничено, либо количество человек, которые можно одновременно задействовать в создании проекта, также ограничено (в этих двух случаях невозможно судить об оптимальности этого соотношения исходя из составленной схемы состава разложения), однако подразумевается, что ни один из этих показателей не является критически строго ограниченным.

4.3. Задание к лабораторной работе

На основании исходных данных (ЛР 1,2) проведите структурно-модульное проектирование и разработайте HIPO-схему создаваемого программного продукта.

К отчёту представить:

- документ в любом электронном формате содержащий структурно-модульное пред- ставление продукта и HIPO-

схемы.

Отчёт о проделанной работе:

- собеседование по представленному документу.

4.4. Контрольные вопросы

1. Метод нисходящего проектирования.
2. Модульное программирование.
3. Модульная структура программных продуктов
4. Назначение обобщенной схемы.
5. Назначение детальной схемы алгоритма.
6. Структурное программирование.

5. ЛАБОРАТОРНАЯ РАБОТА №5: ЭТАПЫ СОПРОВОЖДЕНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

5.1. Теория

Основная задача любого успешного проекта заключается в том, чтобы на момент запуска системы и в течение всего времени ее эксплуатации можно было обеспечить:

- требуемую функциональность системы и степень адаптации к изменяющимся условиям ее функционирования;
- требуемую пропускную способность системы;
- требуемое время реакции системы на запрос;
- безотказную работу системы в требуемом режиме, иными словами – готовность и доступность системы для обработки запросов пользователей;
- простоту эксплуатации и поддержки системы;
- необходимую безопасность.

Производительность является главным фактором, определяющим эффективность системы. Хорошее проектное решение служит основой высокопроизводительной системы.

Проектирование информационных систем охватывает три основные области:

- проектирование объектов данных, которые будут реализованы в базе данных;
- проектирование программ, экранных форм, отчетов, которые будут обеспечивать выполнение запросов к данным;
- учет конкретной среды или технологии, а

именно: топологии сети, конфигурации аппаратных средств, используемой архитектуры (файл-сервер или клиент-сервер), параллельной обработки, распределенной обработки данных и т.п.

В реальных условиях проектирование – это поиск способа, который удовлетворяет требованиям функциональности системы средствами имеющихся технологий с учетом заданных ограничений.

К любому проекту предъявляется ряд абсолютных требований, например максимальное время разработки проекта, максимальные денежные вложения в проект и т.д. Одна из сложностей проектирования состоит в том, что оно не является такой структурированной задачей, как анализ требований к проекту или реализация того или иного проектного решения.

Считается, что сложную систему невозможно описать в принципе. Это, в частности, касается систем управления предприятием. Одним из основных аргументов является изменение условий функционирования системы, например директивное изменение тех или иных потоков информации новым руководством. Еще один аргумент – объемы технического задания, которые для крупного проекта могут составлять сотни страниц, в то время как технический проект может содержать ошибки. Возникает вопрос: а может, лучше вообще не проводить обследования и не делать никакого технического проекта, а писать систему «с чистого листа» в надежде на то, что произойдет некое чудесное совпадение желания заказчика с тем, что написали программисты, а также на то, что все это будет стабильно работать?

Если разобраться, то так ли уж непредсказуемо развитие системы и действительно ли получить информацию о ней невозможно? Вероятно, представление о системе в целом и о предполагаемых (руководством) путях ее развития можно получить посредством семинаров. После этого разбить сложную систему на более простые компоненты, упростить связи между компонентами, предусмотреть независимость компонентов и описать интерфейсы между ними (чтобы изменение одного компонента автоматически не влекло за собой существенного изменения другого компонента), а также возможности расширения системы и «заглушки» для нереализуемых в той или иной версии системы функций. Исходя из подобных элементарных соображений описание того, что предполагается реализовать в информационной системе, уже не кажется столь нереальным. Можно придерживаться классических подходов к разработке информационных систем,

один из которых – схема «водопада» (рис. 1) – описан ниже. Кратко будут рассмотрены и некоторые другие подходы к разработке информационных систем, где использование элементов, описанных в схеме «водопада», также допустимо. Какого подхода из описываемых ниже придерживаться (и есть ли смысл придумывать собственный подход) – в какой-то мере дело вкуса и обстоятельств.

Рис. 1. Схема «водопада»

Жизненный цикл программного обеспечения представляет собой модель его создания и использования. Модель отражает его различные состояния, начиная с момента возникновения необходимости в данном ПО и заканчивая моментом его полного выхода из употребления у всех пользователей. Известны следующие модели жизненного цикла:

- Каскадная модель. Переход на следующий этап означает полное завершение работ на предыдущем этапе.
- Поэтапная модель с промежуточным контролем. Разработка ПО ведется итерациями с циклами обратной связи между этапами. Межэтапные корректировки позволяют уменьшить трудоемкость процесса разработки по сравнению с каскадной моделью; время жизни каждого из этапов растягивается на весь период разработки.
- Спиральная модель. Особое внимание уделяется начальным этапам разработки – выработке стратегии, анализу и проектированию, где реализуемость тех или иных технических решений проверяется и обосновывается посредством создания прототипов (макетирования). Каждый виток спирали предполагает создание некой версии продук-

та или какого-либо его компонента, при этом уточняются характеристики и цели проекта, определяется его качество и планируются работы следующего витка спирали.

Ниже мы рассмотрим некоторые схемы разработки проекта.

«Водопад» - схема разработки проекта

Очень часто проектирование описывают как отдельный этап разработки проекта между анализом и разработкой. Однако в действительности четкого деления этапов разработки проекта нет – проектирование, как правило, не имеет явно выраженного начала и окончания и часто продолжается на этапах тестирования и реализации. Говоря об этапе тестирования, также следует отметить, что и этап анализа, и этап проектирования содержат элементы работы тестеров, например для получения экспериментального обоснования выбора того или иного решения, а также для оценки критериев качества получаемой системы. На этапе эксплуатации уместен разговор и о сопровождении системы.

Ниже мы рассмотрим каждый из этапов, подробнее остановившись на этапе проектирования.

Стратегия

Определение стратегии предполагает обследование системы. Основная задача обследования – оценка реального объема проекта, его целей и задач, а также получение определений сущностей и функций на высоком уровне.

На этом этапе привлекаются высококвалифицированные бизнес-аналитики, которые имеют постоянный доступ к руководству фирмы; этап предполагает тесное взаимодействие с основными пользователями системы и бизнес-экспертами. Основная задача взаимодействия – получить как можно более полную информацию о системе (полное и однозначное понимание требований заказчика) и передать данную информацию в формализованном виде системным аналитикам для последующего проведения этапа анализа. Как правило, информация о системе может быть получена в результате бесед или семинаров с руководством, экспертами и пользователями. Таким образом определяются суть данного бизнеса, перспективы его развития и требования к системе.

По завершении основной стадии обследования системы технические специалисты формируют вероятные технические подходы и приблизительно рассчитывают затраты на аппаратное обеспечение, закупаемое программное обеспечение и разработку нового программного обеспечения (что, собственно, и предполагается проектом).

Результатом этапа определения стратегии является документ, где четко сформулировано, что получит заказчик, если согласится финансировать проект; когда он получит готовый продукт (график выполнения работ); сколько это будет стоить (для крупных проектов должен быть составлен график финансирования на разных этапах работ). В документе должны быть отражены не только затраты, но и выгода, например время окупаемости проекта, ожидаемый экономический эффект (если его удается оценить).

В документе обязательно должны быть описаны:

- ограничения, риски, критические факторы, влияющие на успешность проекта, например время реакции системы на запрос является заданным ограничением, а не желательным фактором;
- совокупность условий, при которых предполагается эксплуатировать будущую систему: архитектура системы, аппаратные и программные ресурсы, предоставляемые системе, внешние условия ее функционирования, состав людей и работ, которые обеспечивают бесперебойное функционирование системы;
- сроки завершения отдельных этапов, форма сдачи работ, ресурсы, привлекаемые в процессе разработки проекта, меры по защите информации;
- описание выполняемых системой функций;
- будущие требования к системе в случае ее развития, например возможность работы пользователя с системой с помощью Интернета и т.п.;
- сущности, необходимые для выполнения функций системы;
- интерфейсы и распределение функций между человеком и системой;
- требования к программным и информационным компонентам ПО, требования к СУБД (если проект предполагается реализовывать для нескольких СУБД, то требования к каждой из них, или общие требования к абстрактной СУБД и список рекомендуемых для данного проекта СУБД, которые удовлетворяют заданным условиям);
- что не будет реализовано в рамках проекта.

Выполненная на данном этапе работа позволяет ответить на вопрос, стоит ли продолжать данный проект и какие требования заказчика могут быть удовлетворены при тех или иных условиях. Может оказаться, что проект продолжать не имеет смысла,

например из-за того, что те или иные требования не могут быть удовлетворены по каким-то объективным причинам. Если принимается решение о продолжении проекта, то для проведения следующего этапа анализа уже имеются представление об объеме проекта и смета затрат.

Следует отметить, что и на этапе выбора стратегии, и на этапе анализа, и при проектировании независимо от метода, применяемого при разработке проекта, всегда следует классифицировать планируемые функции системы по степени важности. Один из возможных форматов представления такой классификации – MoSCoW – предложен в Clegg, Dai and Richard Barker, Case Method Fast-track: A RAD Approach, Adison-Wesley, 1994.

Эта аббревиатура расшифровывается так: Must have – необходимые функции; Should have – желательные функции; Could have – возможные функции; Won't have – отсутствующие функции.

Функции первой категории обеспечивают критичные для успешной работы системы возможности.

Реализация функций второй и третьей категорий ограничивается временными и финансовыми рамками: разрабатываем то, что необходимо, а также максимально возможное в порядке приоритета число функций второй и третьей категорий.

Последняя категория функций особенно важна, поскольку необходимо четко представлять границы проекта и набор функций, которые будут отсутствовать в системе.

Анализ

Этап анализа предполагает подробное исследование бизнес-процессов (функций, определенных на этапе выбора стратегии) и информации, необходимой для их выполнения (сущностей, их атрибутов и связей (отношений)). На этом этапе создается информационная модель, а на следующем за ним этапе проектирования – модель данных.

Вся информация о системе, собранная на этапе определения стратегии, формализуется и уточняется на этапе анализа. Особое внимание следует уделить полноте переданной информации, анализу информации на предмет отсутствия противоречий, а также поиску неиспользуемой вообще или дублирующейся информации. Как правило, заказчик не сразу формирует требования к системе в целом, а формулирует требования к отдельным ее компонентам. Уделите внимание согласованности этих компонентов.

Аналитики собирают и фиксируют информацию в двух вза-

имосвязанных формах:

- функции – информация о событиях и процессах, которые происходят в бизнесе;
- сущности – информация о вещах, имеющих значение для организации и о которых что-то известно.

Двумя классическими результатами анализа являются:

- иерархия функций, которая разбивает процесс обработки на составные части (что делается и из чего это состоит);
- модель «сущность-связь» (Entry Relationship model, ER-модель), которая описывает сущности, их атрибуты и связи (отношения) между ними.

Эти результаты являются необходимыми, но не достаточными. К достаточным результатам следует отнести диаграммы потоков данных и диаграммы жизненных циклов сущностей. Довольно часто ошибки анализа возникают при попытке показать жизненный цикл сущности на диаграмме ER.

Ниже мы рассмотрим три наиболее часто применяемые методологии структурного анализа:

- диаграммы «сущность-связь» (Entity-Relationship Diagrams, ERD), которые служат для формализации информации о сущностях и их отношениях;
- диаграммы потоков данных (Data Flow Diagrams, DFD), которые служат для формализации представления функций системы;
- диаграммы переходов состояний (State Transition Diagrams, STD), которые отражают поведение системы, зависящее от времени; диаграммы жизненных циклов сущностей относятся именно к этому классу диаграмм.

ER-диаграммы

ER-диаграммы (рис. 2) используются для разработки данных и представляют собой стандартный способ определения данных и отношений между ними. Таким образом, осуществляется детализация хранилищ данных. ER-диаграмма содержит информацию о сущностях системы и способах их взаимодействия, включает идентификацию объектов, важных для предметной области (сущностей), свойств этих объектов (атрибутов) и их отношений с другими объектами (связей). Во многих случаях информационная модель очень сложна и содержит множество объектов.

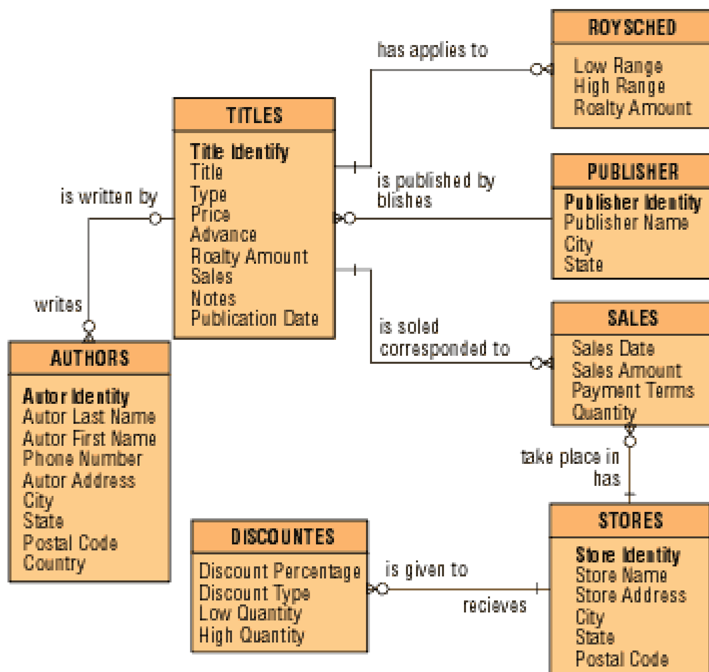


Рис. 2. Пример ER-диаграммы

Сущность изображается в виде прямоугольника, вверху которого располагается имя сущности (например, TITLES). В прямоугольнике могут быть перечислены атрибуты сущности; атрибуты ER-диаграмм, набранные полужирным шрифтом1, являются ключевыми (так Title Identity – ключевой атрибут сущности TITLES, остальные атрибуты ключевыми не являются).

Отношение изображается линией между двумя сущностями (синие линии на рисунке).

Одиночная линия справа (рис. 3) означает «один», «птичья лапка» слева - «многие», а отношение читается вдоль линии, например «один ко многим». Вертикальная черта означает «обязательно», кружок - «не обязательно», например для каждого издания в TITLE обязательно должен быть указан издатель в PUBLISHERS, а один издатель в PUBLISHERS может выпускать несколько наименований изданий в TITLES. Следует отметить, что связи всегда комментируются (надпись на линии, изображающей связь).

Рис. 3. Элемент ER-диаграммы

Приведем также пример (рис. 4) изображения рефлексивного отношения «сотрудник», где один сотрудник может руководить несколькими подчиненными и так далее вниз по иерархии должностей.

Следует обратить внимание на то, что такое отношение всегда является необязательным, в противном случае это будет бесконечная иерархия.

Рис. 4. ER-диаграмма рефлексивного отношения

Атрибуты сущностей могут быть ключевыми – они выделяются полужирным шрифтом; обязательными – перед ними ставится знак «*», то есть их значение всегда известно, необязательными (optional) – перед ними ставится O, то есть значения этого атрибута в какие-то моменты могут отсутствовать или быть неопределенными.

Дуги

Если сущность имеет набор взаимоисключающих отношений с другими сущностями, то говорят, что такие отношения находятся в дуге. Например, банковский счет может быть оформлен или для юридического лица, или для физического лица. Фрагмент ER-диаграммы для такого типа отношений приведен на рис. 5.

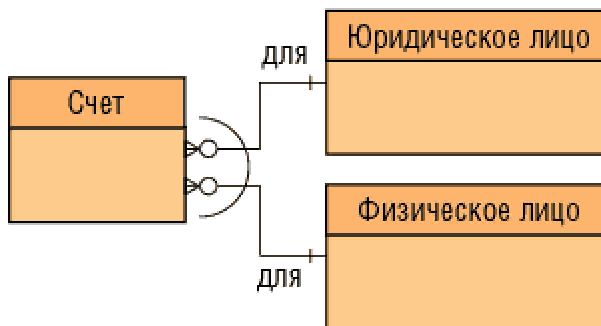


Рис. 5. Дуга

В этом случае атрибут ВЛАДЕЛЕЦ сущности СЧЕТ имеет особое значение для данной сущности – сущность делится на типы по категориям: «для физического лица» и «для юридического

лица». Полученные в результате сущности называют подтипами, а исходная сущность становится супертипом. Чтобы понять, нужен супертип или нет, надо установить, сколько одинаковых свойств имеют различные подтипы. Следует отметить, что злоупотребление подтипами и супертипами является довольно распространенной ошибкой. Изображают их так, как показано на рис. 6.

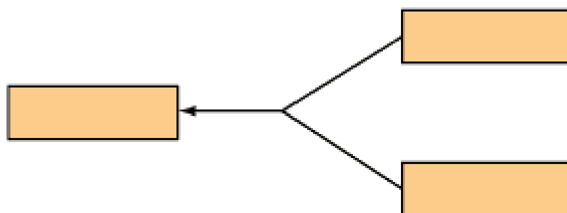


Рис. 6. Подтипы (справа) и супертип (слева)

Нормализация

Чтобы не допустить аномалий при обработке данных, используют нормализацию. Принципы нормализации для объектов информационной модели в точности такие же, как и для моделей данных.

Допустимые типы связей. При ближайшем рассмотрении связи типа «один к одному» (рис. 7) почти всегда оказывается, что A и B представляют собой в действительности разные подмножества одного и того же предмета или разные точки зрения на него, просто имеющие отличные имена и по-разному описанные связи и атрибуты.

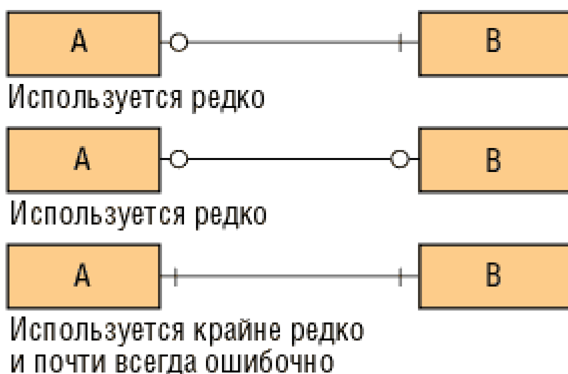


Рис. 7. Связи «один к одному»

Связи «многие к одному» представлены на рис. 8.

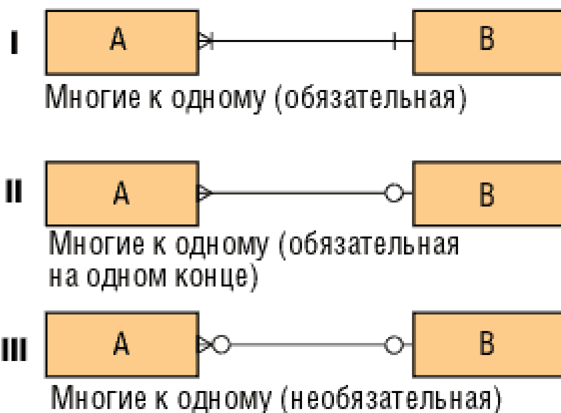


Рис. 8. Связи «многие к одному»

I – достаточно сильная конструкция, предполагающая, что вхождение сущности В не может быть создано без одновременного создания по меньшей мере одного связанного с ним вхождения сущности А.

II – это наиболее часто встречающаяся форма связи. Она предполагает, что каждое и любое вхождение сущности А может существовать только в контексте одного (и только одного) вхождения сущности В. В свою очередь, вхождения В могут существовать как в связи с вхождениями А, так и без нее.

III – применяется редко. Как А, так и В могут существовать без связи между ними.

Связи «многие ко многим» представлены на рис. 9.

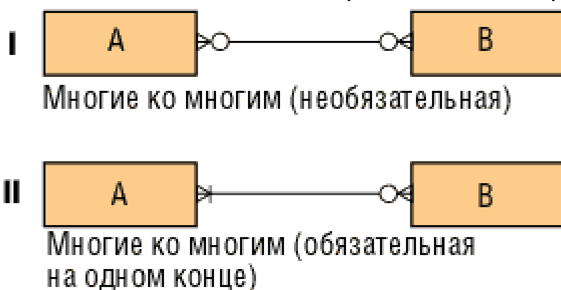


Рис. 9. Связи «многие ко многим»

I – такая конструкция часто имеет место в начале этапа анализа и означает связь – либо понятую не до конца и требующую дополнительного разрешения, либо отражающую простое коллективное отношение – двунаправленный список.

II – применяется редко. Такие связи всегда подлежат дальнейшей детализации.

Рассмотрим теперь рекурсивные связи (рис. 10).

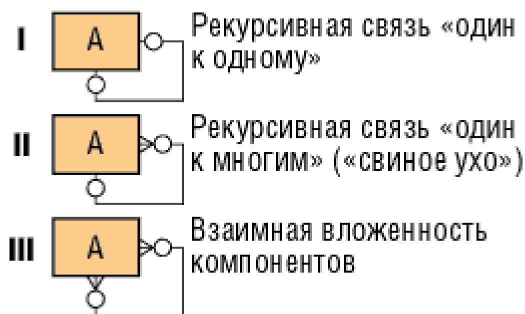


Рис. 10. Рекурсивные связи

I – редко, но имеет место. Отражает связи альтернативного типа.

II – достаточно часто применяется для описания иерархий с любым числом уровней.

III – имеет место на ранних этапах. Часто отражает структуру «перечня материалов» (взаимная вложенность компонентов). Пример: каждый КОМПОНЕНТ может состоять из одного и более (других) КОМПОНЕНТОВ и каждый КОМПОНЕНТ может использоваться в одном и более (других) КОМПОНЕНТОВ.

Недопустимые типы связей. К недопустимым типам связей относятся следующие: обязательная связь «многие ко многим» (рис. 11) и ряд рекурсивных связей (рис. 12).

Рис. 11. Недопустимые связи «многие ко многим»

Обязательная связь «многие ко многим» в принципе невозможна. Такая связь означала бы, что ни одно из вхождений A не может существовать без B, и наоборот. На деле каждая подобная конструкция всегда оказывается ошибочной.

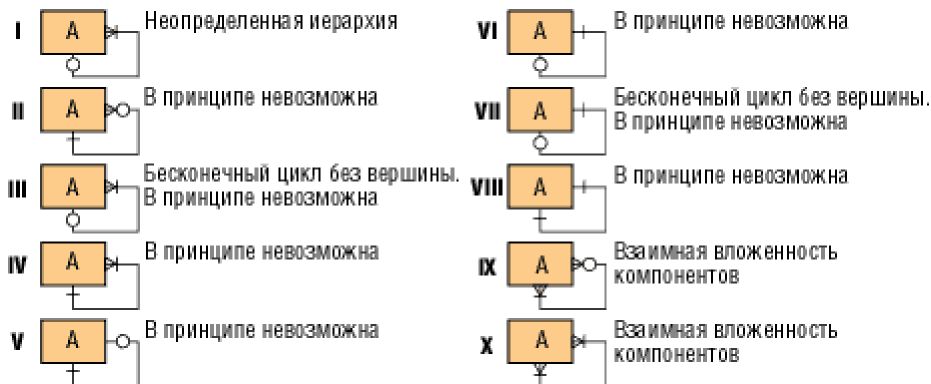


Рис. 12. Недопустимые рекурсивные связи

Диаграммы потоков данных

Логическая DFD (рис. 13) показывает внешние по отношению к системе источники и стоки (адресаты) данных, идентифицирует логические функции (процессы) и группы элементов данных, связывающие одну функцию с другой (потоки), а также идентифицирует хранилища (накопители) данных, к которым осуществляется доступ. Структуры потоков данных и определения их компонентов хранятся и анализируются в словаре данных. Каждая логическая функция (процесс) может быть детализирована с помощью DFD нижнего уровня; когда дальнейшая детализация перестает быть полезной, переходят к выражению логики функции при помощи спецификации процесса (мини-спецификации). Содержимое каждого хранилища также сохраняют в словаре данных, модель данных хранилища раскрывается с помощью ER-диаграмм.

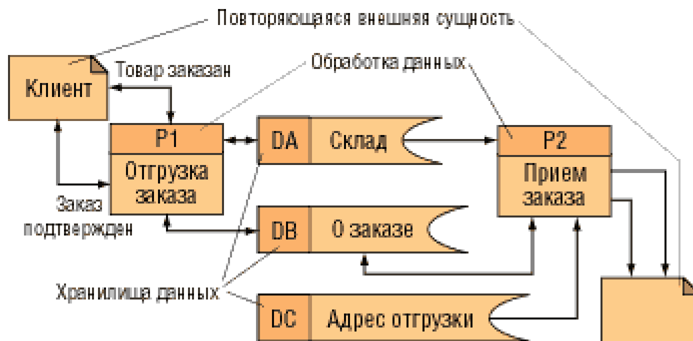


Рис. 13. Пример DFD

В частности, в DFD не показываются процессы, которые

управляют собственно потоком данных и не приводятся различия между допустимыми и недопустимыми путями. DFD содержат множество полезной информации, а кроме того:

- позволяют представить систему с точки зрения данных;
- иллюстрируют внешние механизмы подачи данных, которые потребуют наличия специальных интерфейсов;
- позволяют представить как автоматизированные, так и ручные процессы системы;
- выполняют ориентированное на данные секционирование всей системы.

Потоки данных используются для моделирования передачи информации (или даже физических компонентов) из одной части системы в другую. Потоки на диаграммах изображаются именованными стрелками, стрелки указывают направление движения информации. Иногда информация может двигаться в одном направлении, обрабатываться и возвращаться в ее источник. Такая ситуация может моделироваться либо двумя различными потоками, либо одним двунаправленным.

Процесс преобразует входной поток данных в выходной в соответствии с действием, задаваемым именем процесса. Каждый процесс должен иметь уникальный номер для ссылок на него внутри диаграммы. Этот номер может использоваться совместно с номером диаграммы для получения уникального индекса процесса во всей модели.

Хранилище данных (data storage) позволяет на ряде участков определять данные, которые будут сохраняться в памяти между процессами. Фактически хранилище представляет «срезы» потоков данных во времени. Информацию, которую оно содержит, можно использовать в любое время после ее определения, при этом данные могут выбираться в произвольном порядке. Имя хранилища должно идентифицировать его содержимое. В случае когда поток данных входит (выходит) в (из) хранилище и его структура соответствует структуре хранилища, он должен иметь то же самое имя, которое нет необходимости отражать на диаграмме.

Внешняя сущность (терминатор) представляет сущность вне контекста системы, являющуюся источником или приемником системных данных. Ее имя должно содержать существительное, например «Клиент». Предполагается, что объекты, представленные такими узлами, не должны участвовать ни в какой обработке.

Диаграммы изменения состояний STD

Жизненный цикл сущности относится к классу STD-

диаграмм (рис. 14). Эта диаграмма отражает изменение состояния объекта с течением времени. Например, рассмотрим состояние товара на складе: товар может быть заказан у поставщика, поступить на склад, храниться на складе, проходить контроль качества, может быть продан, забракован, возвращен поставщику. Стрелки на диаграмме показывают допустимые изменения состояний.

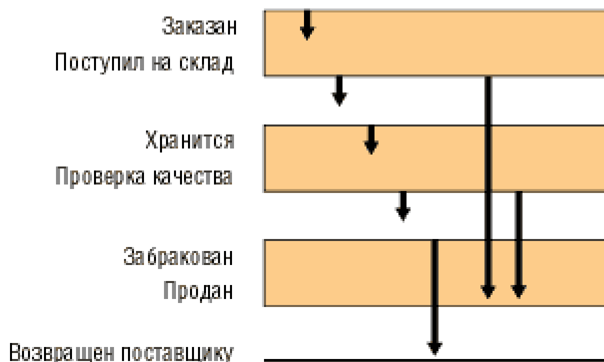


Рис. 14. Пример диаграммы жизненного цикла

Существует несколько различных вариантов изображения подобных диаграмм, на рисунке приведен лишь один из них.

Некоторые принципы проверки качества и полноты информационной модели (источник – Richard Barker, Case Method: Entity Relationship Modelling, Addison-Wesley, 1990)

Если вы хотите создать качественную модель, то придется прибегать к помощи аналитиков, хорошо владеющих CASE-технологией. Однако это не означает, что построением и контролем информационной модели должны заниматься только аналитики. Помощь коллег также может оказаться весьма полезной. Привлекайте их к проверке поставленной цели и к детальному изучению построенной модели как с точки зрения логики, так и с точки зрения учета аспектов предметной области. Большинство людей легче находят недостатки в чужой работе.

Регулярно представляйте вашу информационную модель или ее отдельные фрагменты, относительно которых у вас возникают сомнения, на одобрение пользователей. Особое внимание уделяйте исключениям из правил и ограничениям.

Качество сущностей

Основной гарантией качества сущности является ответ на вопрос, действительно ли объект является сущностью, то есть важным объектом или явлением, информация о котором должна

храниться в базе данных.

Список проверочных вопросов для сущности:

- Отражает ли имя сущности суть данного объекта?
- Нет ли пересечения с другими сущностями?
- Имеются ли хотя бы два атрибута?
- Всего атрибутов не более восьми?
- Есть ли синонимы/омонимы данной сущности?
- Сущность определена полностью?
- Есть ли уникальный идентификатор?
- Имеется ли хотя бы одна связь?
- Существует ли хотя бы одна функция по созданию, поиску, корректировке, удалению, архивированию и использованию значения сущности?
- Ведется ли история изменений?
- Имеет ли место соответствие принципам нормализации данных?
- Нет ли такой же сущности в другой прикладной системе, возможно, под другим именем?
- Не имеет ли сущность слишком общий смысл?
- Достаточен ли уровень обобщения, воплощенный в ней?

Список проверочных вопросов для подтипа:

- Отсутствуют ли пересечения с другими подтипами?
- Имеет ли подтип какие-нибудь атрибуты и/или связи?
- Имеют ли они все свои собственные уникальные идентификаторы или наследуют один на всех от супертипа?
- Имеется ли исчерпывающий набор подтипов?
- Не является ли подтип примером вхождения сущности?
- Знаете ли вы какие-нибудь атрибуты, связи и условия, отличающие данный подтип от других?

Качество атрибутов

Следует выяснить, а действительно ли это атрибуты, то есть описывают ли они тем или иным образом данную сущность.

Список проверочных вопросов для атрибута:

- Является ли наименование атрибута существительным единственного числа, отражающим суть обозначаемого атрибутом свойства?
- Не включает ли в себя наименование атрибута имя сущности (этого быть не должно)?
- Имеет ли атрибут только одно значение в каждый момент времени?
- Отсутствуют ли повторяющиеся значения (или группы)?
- Описаны ли формат, длина, допустимые значения, алго-

ритм получения и т.п.?
- Не может ли этот атрибут быть пропущенной сущностью, которая пригодилась бы для другой прикладной системы (уже существующей или предполагаемой)?
- Не может ли он быть пропущенной связью?
- Нет ли где-нибудь ссылки на атрибут как на «особенность проекта», которая при переходе на прикладной уровень должна исчезнуть?
- Есть ли необходимость в истории изменений?
- Зависит ли его значение только от данной сущности?
- Если значение атрибута является обязательным, всегда ли оно известно?
- Есть ли необходимость в создании домена для этого и ему подобных атрибутов?
- Зависит ли его значение только от какой-то части уникального идентификатора?
- Зависит ли его значение от значений некоторых атрибутов, не включенных в уникальный идентификатор?

Качество связи

Нужно выяснить, отражают ли связи действительно важные отношения, наблюдаемые между сущностями.

Список проверочных вопросов для связи:

- Имеется ли ее описание для каждой участвующей стороны, точно ли оно отражает содержание связи и вписывается ли в принятый синтаксис?
- Участвуют ли в ней только две стороны?

Не является ли связь переносимой?

- Заданы ли степень связи и обязательность для каждой стороны?
- Допустима ли конструкция связи?

Не относится ли конструкция связи к редко используемым?

- Не является ли она избыточной?
- Не изменяется ли она с течением времени?
- Если связь обязательная, всегда ли она отражает отношение к сущности, представляющей противоположную сторону?

Для исключающей связи:

- Все ли концы связей, покрываемые исключающей дугой, имеют один и тот же тип обязательности?
- Все ли из них относятся к одной и той же сущности?
- Обычно дуги пересекают разветвляющиеся концы – что вы можете сказать о данном случае?

- Связь может покрываться только одной дугой. Так ли это?
- Все ли концы связей, покрываемые дугой, входят в уникальный идентификатор?

Функции системы

Часто аналитикам приходится описывать достаточно сложные бизнес-процессы. В этом случае прибегают к функциональной декомпозиции, которая показывает разбиение одного процесса на ряд более мелких функций до тех пор, пока каждую из них уже нельзя будет разбить без ущерба для смысла. Конечный продукт декомпозиции представляет собой иерархию функций, на самом нижнем уровне которой находятся атомарные с точки зрения смысловой нагрузки функции. Приведем простой пример (рис. 15) такой декомпозиции. Рассмотрим простейшую задачу выписки счета клиенту при отпуске товара со склада при условии, что набор товаров, которые хочет приобрести клиент, уже известен (не будем рассматривать в данном примере задачу выбора товаров).

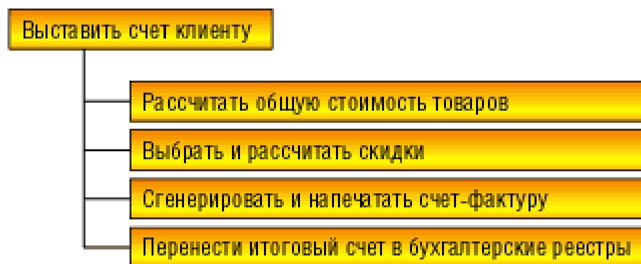


Рис. 15. Пример декомпозиции

Очевидно, что операция выбора и расчета скидок может быть также разбита на более мелкие операции, например на расчет скидок за приверженность (клиент покупает товары в течение долгого времени) и на расчет скидок за количество покупаемого товара. Атомарные функции описываются подробно, например с помощью DFD и STD. Очевидно, что такое описание функций не исключает и дополнительное словесное описание (например, комментарии).

Следует отметить, что на этапе анализа следует уделить внимание функциям анализа и обработки возможных ошибок и отклонений от предполагаемого эталона работы системы. Следует выделить наиболее критичные для работы системы процессы и обеспечить для них особенно строгий анализ ошибок. Обработка ошибок СУБД (коды возврата), как правило, представляет собой обособленный набор функций или одну-единственную функцию.

Уточнение стратегии

На этапе анализа происходит уточнение выбранных для конечной реализации аппаратных и программных средств. Для этого могут привлекаться группы тестирования, технические специалисты. При проектировании информационной системы важно учесть и дальнейшее развитие системы, например рост объемов обрабатываемых данных, увеличение интенсивности потока запросов, изменение требований надежности информационной системы.

На этапе анализа определяются наборы моделей задач для получения сравнительных характеристик тех или иных СУБД, которые рассматривались на этапе определения стратегии для реализации информационной системы. На этапе определения стратегии может быть осуществлен выбор одной СУБД. Данных о системе на этапе анализа уже намного больше, и они более подробны. Полученные данные, а также характеристики, переданные группами тестирования, могут показать, что выбор СУБД на этапе определения стратегии был неверным и что выбранная СУБД не может удовлетворять тем или иным требованиям информационной системы. Такие же данные могут быть получены относительно выбора аппаратной платформы и операционной системы. Получение подобных результатов инициирует изменение данных, полученных на этапе определения стратегии, например пересчитывается смета затрат на проект.

Выбор средств разработки также уточняется на этапе анализа. В силу того что этап анализа дает более полное представление об информационной системе, чем оно было на этапе определения стратегии, план работ может быть скорректирован. Если выбранное на предыдущем этапе средство разработки не позволяет выполнить ту или иную часть работ в заданный срок, то принимается решение об изменении сроков (как правило, это увеличение срока разработки) или о смене средства разработки. Осуществляя выбор тех или иных средств, следует учитывать наличие высококвалифицированного персонала, который владеет выбранными средствами разработки, а также наличие администраторов выбранной СУБД. Эти рекомендации также будут уточнять данные этапа выбора стратегии (совокупность условий, при которых предполагается эксплуатировать будущую систему).

Уточняются также ограничения, риски, критические факторы. Если какие-либо требования не могут быть удовлетворены в информационной системе, реализованной с использованием СУБД и программных средств, выбранных на этапе определения стратегии, то это также инициирует уточнение и изменение получаемых

данных (в конечном итоге сметы затрат и планов работ, а возможно, и изменение требований заказчика к системе, например их ослабление). Более подробно описываются те возможности, которые не будут реализованы в системе.

Определение стратегии тестирования

Как отмечалось ранее, на этапе анализа привлекаются группы тестирования, например для получения сравнительных характеристик предполагаемых к использованию аппаратных платформ, операционных систем, СУБД, иного окружения. Кроме того, на данном этапе определяется план работ по обеспечению надежности информационной системы и ее тестирования. Для любых проектов целесообразным является привлечение тестеров на ранних этапах разработки, в частности на этапе анализа и проектирования. Если проектное решение оказалось неудачным и это обнаружено слишком поздно – на этапе разработки или, что еще хуже, на этапе внедрения в эксплуатацию, - то исправление ошибки проектирования может обойтись очень дорого. Чем раньше группы тестирования выявляют ошибки в информационной системе, тем ниже стоимость сопровождения системы. Время на тестирование системы и на исправление обнаруженных ошибок следует предусматривать не только на этапе разработки, но и на этапе проектирования.

Для автоматизации тестирования следует использовать системы отслеживания ошибок (bug tracking). Это позволяет иметь единое хранилище ошибок, отслеживать их повторное появление, контролировать скорость и эффективность исправления ошибок, видеть наиболее нестабильные компоненты системы, а также поддерживать связь между группой разработчиков и группой тестирования (уведомления об изменениях по e-mail и т.п.). Чем больше проект, тем сильнее потребность в bug tracking.

Проектирование

На этапе проектирования формируется модель данных. Проектировщики в качестве исходной информации получают результаты анализа. Конечным продуктом этапа проектирования являются:

- схема базы данных (на основании ER-модели, разработанной на этапе анализа);
- набор спецификаций модулей системы (они строятся на базе моделей функций).

Если проект небольшой, то в качестве аналитиков, проектировщиков и разработчиков могут выступать одни и те же люди. Возникает вопрос: насколько вообще актуальна передача резуль-

татов самому себе? Думаем, что актуальна. Представьте себе, что вы передаете данные кому-либо, кто мало знает о системе. Зачастую это помогает, например, найти не описанные вообще, нечетко описанные, противоречиво описанные компоненты системы.

Все спецификации должны быть точными. План тестирования системы дорабатывается также на этом этапе разработки. Во многих проектах результаты этапа проектирования оформляются единым документом, который называют технической спецификацией. В нем также описывают принятый подход к решению каких-либо сложных технических вопросов.

Журнал проектирования

При проектировании возникает необходимость регистрировать все обсуждаемые варианты и окончательные решения. Не секрет, что проектировщики порой меняют первоначальные решения. Это может происходить потому, что со временем участники проекта забывают аргументы в пользу принятого решения. Подобную информацию можно хранить в репозитарии используемого CASE-средства, в текстовых файлах, просто на бумаге. Журнал проектирования является полезным материалом для новых членов групп проектировщиков, а также для разработчиков и тестировщиков.

Такой журнал может вестись как на этапе анализа, так и на этапе разработки и тестирования.

Планирование этапа проектирования

Тщательное планирование важно для любого проекта. Это входит в обязанности руководителя проекта и руководителя группы проектирования (консультации с аналитиками в этом случае будут обязательными). Это позволяет:

- Разбить глобальную задачу на небольшие, независимые задачи. Такими задачами легче управлять, такие задачи легче реализовывать.
- Определить контрольные даты (этапы сдачи), которые позволят определить, насколько успешно продвигается проект, какие направления отстают, какие недогружены, какие работают успешно. Это позволяет обнаружить отставание от сроков сдачи и вовремя предотвратить авралы.
- Определить зависимости между задачами, а также последовательность завершения задач.
- Прогнозировать загрузку персонала, наем временных работников, привлечение других групп разработчиков, привлечение консультантов (если это необходимо).

- Получить четкое представление о том, когда можно начать этап реализации.
- Получить четкое представление о том, когда можно начать этап опытной эксплуатации.

Перепланирование

Заказчики всегда хотят, чтобы план выполнения работ оставался неизменным. На практике этого редко удается достичь в полном объеме. Определенным компромиссом здесь может стать неизменность установленных сроков сдачи компонентов системы в эксплуатацию.

Задачи проектирования.

Ранние стадии

Рассмотрение результатов анализа

Это собственно процесс передачи информации от аналитиков проектировщикам. На практике это итерактивный процесс. У проектировщиков неизбежно будут возникать вопросы к аналитикам, и наоборот. Информация о системе будет постоянно уточняться. При разработке схемы базы данных может измениться информационная модель, полученная на этапе анализа, например потому, что имеющееся проектное решение нестабильно либо медленно работает при реализации его посредством выбранной СУБД или в силу иных причин. Проверить, охватывает ли анализ все бизнес-процессы системы (то есть осуществить проверку на полноту), проектировщики не в состоянии, но проверку информационной модели на непротиворечивость и корректность проектировщики провести могут. Это позволяет отследить ошибки в информационной модели и не повторить их в модели данных. Если результаты хранятся в репозитарии CASE-средства, то такая проверка на корректность может быть произведена автоматически.

Семинары

Ранние стадии проектирования сопряжены с нудной и утомительной работой. Проектировщики и аналитики должны достигнуть полного понимания требований заказчика. Семинары являются быстрым и эффективным способом обмена информацией. Хороший способ убедиться в том, правильно ли проектировщики понимают назначение той или иной подсистемы, - взять один или несколько сценариев бизнес-процессов и проиграть их. Это можно сделать в форме простой диаграммы потока данных, причем необходимо указать не только автоматизированные, но и ручные функции.

Критические участки

Критические участки системы изучаются при первом обсле-

довании системы и уточняются на этапе анализа. Термин «критические» может означать жизненно важные как для нормального функционирования информационной системы с точки зрения бизнеса (например, время простоя автоматизированной линии изготовления материала X не должно превышать одной минуты), так и для успешной реализации и приемки проекта. Критические с точки зрения бизнеса участки информационной системы хорошо подходят для макетирования. На основе этих макетов (работы макетирования выполняют проектировщики и группы тестирования) тестеры дают оценку качества как информационной модели, так и модели данных. Также макетирование позволяет показать, какие требования и какими средствами могут быть выполнены, а какие требования – не могут.

Часто на этапе проектирования выявляются критические участки, которые не были очевидными на этапе анализа. Это влечет за собой необходимость уточнения информационной модели. Часто это связано с особенностями реализации тех или иных возможностей в выбранной СУБД. Некоторые функции, которые на этапе анализа выглядят простыми, могут стать очень сложными, когда дело дойдет до физической реализации. Например:

- В выбранной СУБД отсутствует эффективный механизм сканирования деревьев, а при анализе выявлено большое количество справочников и выбраны интерфейсы представления в виде деревьев, кроме того это понравилось заказчику, а СУБД при большом справочнике работает слишком медленно.
- Другая распространенная неприятность – неполно реализованная ссылочная целостность. В СУБД не реализованы каскадные модификации, в информационной модели нормализованные отношения предполагают наличие каскадных удалений и обновлений. Реализация же таких механизмов посредством триггеров оказалась слишком медленной, и уровень каскадирования триггеров ниже, чем уровень каскадных операций, определенных в информационной модели.

Такие моменты могут инициировать не только изменение информационной модели, но и смену СУБД.

Между критическими участками проекта и его рисками существует тесная связь. Критический участок разработки (например, срыв сроков первого этапа сдачи проекта заказчику) может стоить целого проекта. Причинами срыва могут быть как ошибки проектирования, так и нехватка персонала.

Оценка ограничений

Ограничениями, известными с момента обследования бизнес-процессов, являются смета затрат и сроки внедрения. Могут быть и другие ограничения, например допуск персонала к той или иной информации (группы аналитиков к информации о бизнес-процессах в фирме, ограничения доступа к секретной информации и т.п.).

Решения относительно выбора аппаратной платформы, как правило, необратимы, поскольку тесно связаны со сметой затрат и наличием обслуживающего персонала. Например, решения на платформе RS/6000 и Intel с точки зрения сметы затрат выглядят одинаково, но персонала, способного квалифицированно обслуживать RS/6000, нет, и руководство не согласно оплатить обучение сотрудников, хотя решение на основе RS/6000 обладает более высокой масштабируемостью. Это может послужить причиной выбора платформы Intel. Аналогичные причины могут влиять и на выбор операционной системы.

Если проект является расширением или модернизацией существующей информационной системы, то число унаследованных ограничений также может быть большим. На этапе проектирования осуществляется обязательная проверка требований к информационной системе в свете выявленных ограничений. Менять платформу, операционную систему или СУБД на этапе реализации сложно, а на этапе опытной эксплуатации практически невозможно (на это просто не хватит времени, если не произойдет чудо). Чем большее количество компонентов системы уже реализовано, тем сложнее произвести подобную замену. Большинство СУБД сейчас работают на нескольких аппаратных платформах и нескольких операционных системах, но если есть участки кода проекта, зависимые от операционной системы или аппаратной платформы, то их изменение может обойтись очень дорого. В данной статье мы не будем обсуждать вопросы переносимости кода, поскольку они выходят за рамки этого цикла.

Если какие-либо требования не могут быть удовлетворены в принципе, принимается решение о доведении этого факта до сведения спонсоров проекта (руководства фирмы). Обнаружение неработоспособности системы в процессе эксплуатации ничем хорошим обернуться не может, особенно если до руководства фирмы дойдет информация о том, что невыполнимость требований была известна заранее.

Определение целевой архитектуры

Под выбором архитектуры мы понимаем и выбор платфор-

мы (платформ), и выбор операционной системы (операционных систем). В системе могут работать несколько компьютеров на разных аппаратных платформах и под управлением различных операционных систем. Если к автоматизации того или иного бизнеса до вас уже приложили руки, причем неоднократно, вы можете обнаружить настоящий «зверинец» платформ и операционных систем. Перенос ПО на ту или иную платформу – процесс не безболезненный, да и управление разнородной сетью может также стать делом проблемным. Если же обстоятельства таковы, что ПО на рабочих местах конечных пользователей должно работать под управлением нескольких операционных систем (ОС), то следует обязательно выделить зависимые от ОС участки кода и жестко описать интерфейсы обмена компонентов информационной системы, сделав их независимыми от ОС. При написании кода модулей, работающих под управлением нескольких ОС, следует ориентироваться на ту из них, которая обладает наиболее жесткими требованиями.

Кроме определения платформы следует выяснить следующее:

- Будет ли это архитектура «файл-сервер» или «клиент-сервер».
- Будет ли это 3-уровневая архитектура со следующими слоями: сервер, ПО промежуточного слоя (сервер приложений), клиентское ПО.
- Будет ли база данных централизованной или распределенной. Если база данных будет распределенной, то какие механизмы поддержки согласованности и актуальности данных будут использоваться.
- Будет ли база данных однородной, то есть будут ли все серверы баз данных продуктами одного и того же производителя (например, все серверы только Oracle или все серверы только DB2 UDB). Если база данных небудет однородной, то какое ПО будет использовано для обмена данными между СУБД разных производителей (уже существующее или разработанное специально как часть проекта).
- Будут ли для достижения должной производительности использоваться параллельные серверы баз данных (например, Oracle Parallel Server, DB2 UDB и т.п.).

Эти решения часто принимаются до начала этапа проектирования (на этапе анализа). На этапе проектирования полезно еще раз рассмотреть все причины выбора той или иной архитек-

туры, провести тесты производительности и надежности критических участков информационной системы. Это позволит избежать тяжело устраняемых ошибок проектирования. Довольно часто на этапе проектирования возникают непредвиденные технические проблемы, например аналитики не учитывают группу пользователей, имеющих доступ к информационной системе посредством текстовых терминалов. Это ошибка анализа, но выявляется она только на этапе проектирования. Такие проблемы должны решаться вместе с аналитиками, которые инициируют изменение информационной модели.

Если в среде используется сеть, то на этапе проектирования необходимо определить требуемые уровни сервиса сети и спроектировать ее топологию. Необходимо также провести тесты сети, чтобы увидеть, обеспечивает ли существующая сеть должную пропускную способность и имеется ли резерв пропускной способности сети. Если результат отрицательный, то следует четко описать необходимые изменения аппаратного обеспечения и топологии сети.

Выделение потенциальных узких мест в информационной системе

Если заказчик заявит, что производительность системы не имеет никакого значения, примите это замечание с юмором. Это означает лишь то, что время ответа системы на запрос не является (или не кажется заказчику в данный момент) критическим. Попробуйте спросить, приемлемо ли время ответа системы, равное одному часу или одному дню. Вряд ли ответ на этот вопрос будет положительным.

Производительность важна для любой информационной системы. Узким местом называют момент падения производительности системы. Конкретный ответ на вопрос, где узкие места данной системы, может дать лишь специальное направленное тестирование. Но это не означает, что оценка потенциальных узких мест невозможна. Одним из хороших методов является график нагрузки на систему в течение дня, недели, месяца и т.п. Можно построить диаграмму, на которой будет отражено время работы тех или иных бизнес-процессов, а также требуемое для данного бизнес-процесса время ответа системы. Такие диаграммы помогают выявить момент, когда нагрузка будет наиболее интенсивной. Количество пользователей, одновременно работающих с тем или иным компонентом, отражается на диаграмме посредством весового коэффициента (рис. 1).

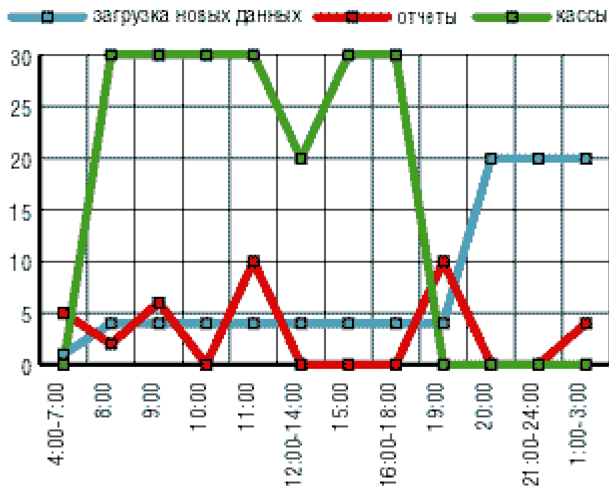


Рис. 1. Пример диаграммы активности системы

В приведенном примере явно видны 3 пика активности системы, максимальный из которых приходится на 11 часов. Использован тип диаграммы с накоплением.

А в диаграмме, представленной на рис. 2, видна активность касс в течение рабочего дня и повышение активности загрузки данных в нерабочее время. В такие диаграммы следует также добавлять вес, отражающий сложность бизнес-процесса, например в данном примере самый высокий весовой коэффициент будут иметь отчеты. Оценка весов определяется особенностями каждого конкретного бизнеса – где-то она может быть высокой, где-то низкой.

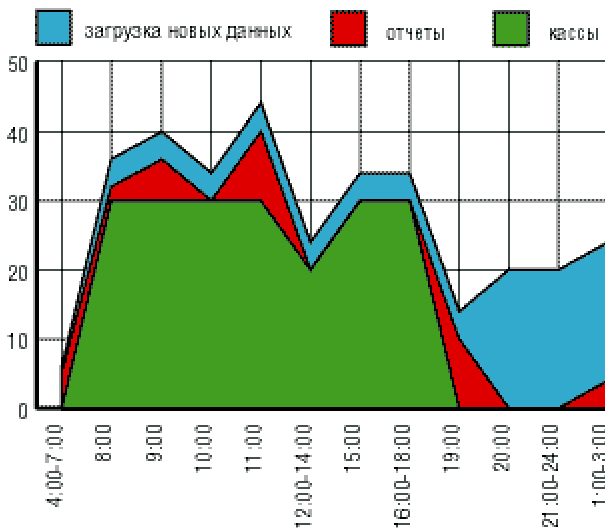


Рис. 2. Еще один пример диаграммы активности системы

Ответ на вопрос, насколько потенциальные узкие места являются реальными, может дать только тестирование. Здесь оправданно применение специальных средств моделирования сценариев приложений. Следует отметить, что оценка точности детектирования узкого места в системе очень зависит от объема обрабатываемых данных. Следует уделить внимание генерации тестовых данных и проверке узких мест уже на этих данных. Часто информационная система не сразу выходит на проектную мощность, как правило, она работает некоторое время в режиме первоначального накопления информации, которое может продолжаться и несколько дней, и несколько месяцев. Как правило, предполагаемый порог объема обрабатываемых данных известен на этапе анализа, но реальный объем физических данных можно точно оценить только на этапе проектирования. Если сгенерировать предполагаемый объем тестовых данных нельзя (не хватает мощности техники или есть иные причины), то тесты проводят на меньшем объеме данных и пытаются построить оценки поведения системы на реальном объеме данных.

Более точно узкие места системы оцениваются на этапе разработки. Здесь уже есть реализованные компоненты системы. Средства автоматизации тестирования (например, LoadRunner, WinRunner и др.) позволяют отследить операции, которые выполняет то или иное приложение (но данные средства могут отследить далеко не все возможные типы приложений и то, насколько

они подходят для тестирования вашего проекта, - это решение такого же порядка, что и выбор средства разработки приложения), автоматически сгенерировать сценарий запуска имитаторов работы реальных приложений и построить оценки узких мест системы.

Продукты третьих фирм

На этапе проектирования оценивают возможность и эффективность использования продуктов третьих форм в разработке данной информационной системы. Например, существует задача выполнения некоторого набора работ (определенных пакетных заданий и т.п.) по заданному графику. Далеко не всегда целесообразно включать в проект создание утилиты контроля запуска приложений, поскольку есть масса утилит, выполняющих эти операции, в том числе и свободно распространяемых. Существует и другая причина, по которой с ПО третьих фирм следует хотя бы ознакомиться. Не факт, что в мировой практике решения задач, подобных вашей, не встречаются. Если реализации третьих фирм известны, то следует с ними ознакомиться хотя бы для того, чтобы не повторять неудачные решения и взять на заметку удачные. Вероятно, какой-либо из существующих продуктов может быть интегрирован в создаваемую вами информационную систему. Для этого, возможно, потребуется создать интерфейс обмена данными между ПО третьей фирмы и вашим. Следует оценить целесообразность как разработки собственного компонента, так и интеграции уже готового аналогичного компонента.

Использование CASE-средств

CASE-средства предоставляют много преимуществ. На одной чаше весов будет автоматизация работы, предоставляемая CASE, а на другой – ненавистная задача преобразования результатов анализа в формат этого CASE (если для формализации результатов анализа использовался другой CASE-инструмент или не использовался никакой). Некоторые CASE-средства позволяют непосредственно перейти к проектированию, а к анализу можно вернуться путем обратного проектирования. К сожалению, при использовании обратного проектирования в CASE-средстве создается весьма вредная иллюзия того, что данные анализа регистрируются, хотя на самом деле этого практически никогда не происходит, поскольку информация, содержащаяся в спроектированной структуре, отличается от результатов анализа. Некоторые полезные данные получить можно, но построить полную картину вряд ли удастся.

Инфраструктура

Для проектирования и реализации необходимы аппаратные ресурсы и специальное программное обеспечение. Кроме того, требуется механизм, позволяющий контролировать создаваемую документацию и код. Эти вопросы лучше решать на ранних стадиях проектирования, а не на стадии разработки. Мы поговорим об этом ниже, в разделе «Проектирование процессов и кода». При групповой разработке вам понадобятся средства контроля согласованности кода. Если разработка идет под разными платформами (аппаратная платформа и ОС), то хорошим решением может оказаться PVCS. Для платформ Windows 98, NT, 2000 может оказаться приемлемым решение, предлагаемое Microsoft – MS Source Save. Кроме того, многие средства разработки также предоставляют возможности контроля исходного кода.

Проектирование базы данных

Здесь мы изложим задачи, касающиеся проектирования реальной базы данных.

Построение модели данных

Работа проектировщиков базы данных в значительной степени зависит от качества информационной модели. Информационная модель не должна содержать никаких непонятных конструкций, которые нельзя реализовать в рамках выбранной СУБД. Следует отметить, что информационная модель создается для того, чтобы на ее основе можно было построить модель данных, то есть должна учитывать особенности реализации выбранной СУБД. Если те или иные особенности СУБД не позволяют отразить в модели данных то, что описывает информационная модель, значит, надо менять информационную модель, так как производитель СУБД вряд ли будет оперативно менять собственно СУБД ради вашего конкретного проекта (хотя и такие, правда единичные, случаи имели место).

Построение логической и физической моделей данных является основной частью проектирования базы данных. Полученная в процессе анализа информационная модель сначала преобразуется в логическую, а затем в физическую модель данных. После этого для разработчиков информационной системы создается пробная база данных. С ней начинают работать разработчики кода. В идеале к моменту начала разработки модель данных должна быть устойчива. Проектирование базы данных не может быть оторвано от проектирования модулей и приложений, поскольку бизнес-правила могут создавать объекты в базе данных, например серверные ограничения (constraints), а также хранимые процедуры и триггеры, - в этом случае часто говорят, что часть биз-

нес-логики переносится в базу данных. Проектирование модели данных для каждой СУБД содержит свои особенности, проектные решения, которые дают хороший результат для одной СУБД, но могут оказаться совершенно неприемлемыми для другой. Ниже перечислим задачи, которые являются общими для проектирования моделей данных:

- выявление нереализуемых или необычных конструкций в ER-модели и в определениях сущностей;
- разрешение всех дуг, подтипов и супертипов;
- изучение возможных, первичных, внешних ключей, описание ссылочной целостности (в зависимости от реализации декларативно или с использованием триггеров);
- проектирование и реализация денормализации базы данных в целях повышения производительности;
- определение части бизнес-логики, которую следует реализовать в базе данных (пакеты, хранимые процедуры);
- реализация ограничений (ограничений и триггеров), отражающих все централизованно определенные бизнес-правила, генерация ограничений и триггеров;
- определение набора бизнес-правил, которые не могут быть заданы как ограничения в базе данных;
- определение необходимых индексов, кластеров (если таковые реализованы в СУБД), определение горизонтальной фрагментации таблиц (если это реализовано в СУБД);
- оценка размеров всех таблиц, индексов, кластеров;
- определение размеров табличных пространств и особенностей их размещения на носителях информации, определение спецификации носителей информации для промышленной системы (например, тип raid-массивов, их количество, какие табличные пространства на них размещаются), определение размеров необходимых системных табличных пространств (например, системного каталога, журнала транзакций, временного табличного пространства и т.п.);
- определение пользователей базы данных, их уровней доступа, разработка и внедрение правил безопасности доступа, аудита (если это необходимо), создание пакетированных привилегий (в зависимости от реализации СУБД это роли или группы), синонимов;
- разработка топологии базы данных в случае распределенной базы данных, определение механизмов доступа к удаленным данным.

Подробнее на каждом из перечисленных пунктов мы остановимся в части «Схема базы данных».

Создание базы данных для разработчика

Чаще всего базу данных создает администратор баз данных – если он есть; в противном случае это приходится делать проектировщикам. Физическая база данных нужна разработчикам информационной системы для разработки кода, а проектировщикам для проверки их идей. Проектировщики и разработчики могут работать как с одной и той же схемой, так и с разными схемами. В процессе разработки проекта, как правило, создается несколько версий схемы базы данных. Следует обязательно вести журнал изменений схемы (вручную или в репозитарии case) и жестко контролировать версии схемы.

Проектирование процессов и кода

Параллельно с проектированием схемы базы данных требуется выполнить проектирование процессов, чтобы получить спецификации всех модулей. Если часть бизнес-логики хранится в базе данных (ограничения, триггеры, хранимые процедуры), то оба эти процесса проектирования тесно связаны. Главная цель проектирования заключается в отображении функций, полученных на этапе анализа, в модули информационной системы. Определения модулей раскрываются в технической спецификации программ. Возможно, что некоторые атомарные функции, полученные на этапе анализа, вообще не будут отображены в какие-либо модули, а будут преобразованы в ручные процедуры или принципы работы.

Выбор средств разработки

Не следует откладывать выбор средств разработки на самый последний момент. Если проектировщик не слишком хорошо представляет себе набор средств, которые будут использованы для разработки проекта, то следует для начала составить перечень возможных средств, затем провести консультации с техническими специалистами (хорошо знающими средства-кандидаты), оценить, с какими средствами персонал уже работал, а какие являются для них абсолютно новыми. Часто выбор средств разработки определяет именно фактор квалификации персонала.

Отображение функций на модули

На этапе анализа уже разработан перечень функций, которые будут реализованы. На этапе проектирования этот перечень еще раз анализируется и корректируется. Однозначное соответствие между функцией и модулем вряд ли возможно. Дело в том, что на этапе анализа функции организованы по бизнес-

категориям, а на этапе проектирования их придется реорганизовывать для упрощения разработки. Проектировщики могут принять решение объединить несколько функций, обладающих общими свойствами, или выделить какое-то общее свойство (или их набор) в отдельный модуль, а также разбить сложную функцию на несколько модулей. Подробнее эти вопросы рассматриваются в разделе «Спецификации функций».

Интерфейсы программ

При проектировании модулей определяют разметку меню, вид окон, горячие клавиши и связанные с ними вызовы. Существуют два вида перемещения по программам:

- с контекстом, когда целевая экранная форма частично или полностью заполняется автоматически данными, связанными с теми, что находятся в исходной экранной форме;
- без контекста, когда целевая экранная форма не заполняется вовсе или частично заполняется автоматически данными, не связанными с теми, что находятся в исходной экранной форме.

Часто автоматически заполняемые данные экранной формы группируют (располагают рядом), а перемещение по заполняемым пользователем полям организуют так, как это делал бы сам пользователь, работая с реальным бумажным документом. Такие интерфейсы воспринимаются пользователем легче, и он намного быстрее осваивает новое ПО.

Интегрирование и наследование механизмов обмена данными

Информационная система редко разрабатывается с нуля. Чаще проектировщики сталкиваются с задачами наследования данных из старых систем, которые уже выполняют какие-либо задачи автоматизации бизнеса. Такие системы могут на начальном этапе быть интегрированы в новую систему и постепенно заменяться новыми, более современными модулями. Этот подход может навязываться руководством фирмы для того, чтобы ускорить ввод новой информационной системы. Следует рассмотреть все плюсы и минусы такой постепенной интеграции (минусов, как правило, оказывается больше). Одну операцию придется делать в любом случае: переносить ценные данные, хранящиеся в старой информационной системе, в новую, то есть проектировать механизмы конвертации данных. Возможно, что придется делать конвертацию данных не только из старой системы в новую, но и обратно (полную или частичную), поскольку возможен вариант развития событий, при котором старая и новая информационные си-

стемы будут работать параллельно – хотя бы в период опытной эксплуатации новой системы.

Кроме вопросов наследования собственно данных из старых информационных систем, возможно, вам придется также решать задачи взаимодействия вашего ПО с продуктами третьих фирм. В этом случае вам следует изучить интерфейсы обмена данными ПО других разработчиков и обеспечить должный уровень поддержки этих интерфейсов в разрабатываемой информационной системе.

Определение спецификаций модулей

Это основная часть функционального проектирования. Здесь решаются следующие задачи:

- преобразование функциональных определений анализа в реализуемые модули;
- спецификации, которые выражают функциональные возможности каждого модуля в физических категориях;
- определение средств разработки для каждого модуля (или выделенных групп модулей), если используются несколько средств разработки в одном проекте;
- определение последовательности реализации модулей и зависимостей модулей.

Спецификации модулей различают по степени детализации и содержанию даже в рамках одного проекта. Определяют, сколько времени требуется для того, чтобы сгенерировать тот или иной модуль, сколько необходимо на тестирование того или иного модуля, а также на тестирование совокупности сгенерированных модулей. Кроме того, следует разработать специальные метрики – шаблоны, которые позволяют оценить, сколько времени потребуется на создание исходного кода модуля. Для ускорения процесса разработки следует рассмотреть возможность использования генераторов исходного кода – это целесообразно, если вам предстоит разработать большое количество несложных модулей, а время разработки ограничено. Следует использовать шаблоны кода для устранения рутинных операций. Подробнее эти вопросы рассматриваются в разделе, посвященном спецификациям функций, - вы найдете его в одной из следующих статей данного цикла.

Заключительные стадии проектирования

Проектирование процесса тестирования

Проектирование процесса тестирования, как правило, следует за процессом функционального проектирования и проектирования схемы базы данных. На этом этапе можно использовать сложные схемы тестирования, а можно ограничиться и простыми.

Здесь мы приведем некоторые принципы, которых нужно придерживаться при проектировании любой информационной системы.

Когда генерация модуля завершена, выполняют автономный тест, который преследует две основные цели:

- обнаружение отказов модуля (жестких сбоев);
- соответствие модуля спецификации (наличие всех необходимых функций, отсутствие лишних функций).

После того как автономный тест прошел успешно, группа сгенерированных модулей проходит тесты связей, которые должны отследить взаимное влияние модулей.

Далее группа модулей тестируется на надежность работы, то есть проходят, во-первых, тесты имитации отказов системы, а во-вторых, тесты наработки на отказ. Первая группа тестов показывает, насколько хорошо система восстанавливается после сбоев программного обеспечения, отказов аппаратного обеспечения. Вторая группа тестов определяет степень устойчивости системы при штатной работе и позволяет оценить время безотказной работы системы. В комплект тестов устойчивости должны входить тесты, имитирующие пиковую нагрузку на систему.

Затем весь комплект модулей проходит системный тест – тест внутренней приемки продукта, показывающий уровень его качества. Сюда входят тесты функциональности и тесты надежности системы.

Последний тест информационной системы – приемо-сдаточные испытания. Такой тест предусматривает показ информационной системы заказчику и должен содержать группу тестов, моделирующих реальные бизнес-процессы, чтобы показать соответствие реализации требованиям заказчика.

Требования к безопасности, доступу, обслуживанию системы

Каждая информационная система содержит определенные требования к защите от несанкционированного доступа, к регистрации событий системы, аудиту, резервному копированию, восстановлению информации, которые в начале проектирования должны быть формализованы аналитиками. Проектировщики строят стратегию безопасности системы. В частности, ими должны быть определены категории пользователей системы, которые имеют доступ к тем или иным данным посредством соответствующих компонентов. Кроме того, определяются объекты и субъекты защиты. Следует отметить, что стратегия безопасности не ограничивается только ПО – это должен быть целый комплекс мер и правил ведения бизнеса. Нужно четко определить, какой

уровень защиты данных необходим для каждого из компонентов информационной системы, и выделить критичные данные, доступ к которым строго ограничен. Пользователи информационной системы регистрируются, поэтому проектируются модули, отвечающие за идентификацию и аутентификацию пользователя. В большинстве СУБД реализована дискреционная защита данных, то есть регламентирован доступ к объектам данных (например, к таблицам, представлениям). Если же требуется ограничение доступа собственно к данным (к отдельным записям в таблице, к отдельным полям записи в таблице и т.п.), то следует реализовать мандатную защиту. Проектировщики должны иметь четкое представление о том, какой уровень защиты той или иной единицы информации является необходимым, а какой достаточным.

Вопросы восстановления, хранения резервных копий базы данных, архивов базы данных относятся к мероприятиям поддержки бесперебойного функционирования информационной системы. Необходимо внимательно изучить возможности, предоставляемые СУБД, а затем проанализировать, как следует использовать возможности СУБД для обеспечения требуемого уровня бесперебойной работы системы.

Составление спецификаций

Результаты проектирования отражаются в документе – функциональной спецификации. Этот документ пишется для заказчика, чтобы получить его санкцию на завершение проектирования и начало разработки, и обычно не содержит большого количества технических деталей. Второй документ – техническая спецификация, являющаяся основным документом для разработчиков моделей и групп тестирования, и здесь описаны детали проекта. Если использовались CASE-средства, то техническая спецификация обязательно содержит ряд отчетов из репозитария.

Полнота проектирования

Перед началом разработки модулей нужно еще раз проверить полноту проектирования. Один из полезных инструментов – матрица использования таблиц схемы базы данных по модулям.

Переход к реализации

Итак, начата реализация модулей. Означает ли это, что работа проектировщиков на этом завершена полностью? На практике это далеко не так. Довольно часто разработчик сталкивается с медленно работающими или не реализуемыми в данной схеме запросами. Подобные ситуации инициируют изменение модели данных, а значит, и информационной модели. Однако изменение информационной модели производится не только по этой при-

чине. Хорошему проектировщику необходим практический опыт работы с аппаратным и программным обеспечением – вот одна из причин участия проектировщиков в составе групп разработчиков. Нередко ведущие сотрудники групп разработчиков одновременно являются проектировщиками.

Как можно использовать проектировщиков на этапе разработки? Приведем некоторые примеры:

- Проектировщик, написавший спецификацию модуля, проводит семинар с разработчиками и демонстрирует необходимые прототипы (семинар предполагает диалог двух сторон).
- Когда модуль передан разработчику, проектировщик может участвовать в его пересмотре, а также выполнять контрольные функции по реализации проектных решений.
- Для крупных проектов характерно поэтапное выполнение работ, так что вполне вероятно, что после завершения реализации группы модулей и сдачи очередного этапа процесс проектирования будет продолжен для новой группы модулей.
- Проектировщики должны обеспечить быстрое реагирование на возможные изменения требований заказчика, поскольку своевременная обработка такой информации является их обязанностью. Кроме того, необходимо и участие системных аналитиков, так как именно они общаются с заказчиком проекта.

Схема базы данных

Схема базы данных содержит описание всех объектов базы данных: пользователей, их привилегий, таблиц, представлений, индексов, кластеров, ограничений, пакетов, хранимых процедур, триггеров и т.п. При этом создаются не только определения этих объектов, но и сами объекты, с которыми потом работают разработчики.

ER-модель и ее отображение на схему данных

Результат этапа анализа – построение информационной модели. Казалось бы, дело это простое: сущности становятся таблицами, а атрибуты сущностей – столбцами таблиц; ключи становятся первичными ключами, для возможных ключей определяется ограничение unique, внешние ключи становятся декларациями ссылочной целостности. Аналитики, как правило, не вникают в особенности реализации той или иной СУБД, поэтому при проектировании схемы базы данных проектировщик сталкивается с конструкциями в информационной модели, которые не реализуе-

мы или трудно реализуемы в выбранной СУБД.

Приведем несколько примеров ограничений реализации СУБД:

- В информационной модели описаны три сущности – A, B, C. Сущности B и C содержат внешние ключи, ссылающиеся на сущность A. В СУБД поддерживается возможность определения внешнего ключа только для первичного ключа, а для возможного ключа определить декларативную ссылочную целостность нельзя. В этом случае отображение ER-модели на физическую модель данных невозможно без изменения информационной модели.
- В информационной модели описан внешний ключ с каскадным удалением и модификацией. В СУБД поддерживаются внешние ключи только для варианта действия no action (то есть каскадные изменения в явном виде не поддерживаются). Реализация ссылочной целостности посредством триггеров ограничена уровнем каскадирования триггеров (например, 32 вызовами триггера). В этом случае потребуется также изменение информационной модели.
- В информационной модели определен атрибут, представляющий собой строку длиной в 500 символов. По этому атрибуту часто осуществляется поиск в информационной системе; объем данных велик. В СУБД можно индексировать строки символов не длиннее чем 128 или 256 символов. Если осуществлять поиск без индекса, то время ответа информационной системы существенно превышает допустимое, вследствие чего придется изменить описание сущности.
- В информационной модели описана сущность A, которая содержит по крайней мере два атрибута BLOB (например, требуется отдельно хранить и звук и изображение). В СУБД невозможно создать таблицу с двумя атрибутами BLOB и в этом случае нужно изменить описание сущности. Из отношения A исключаются два атрибута BLOB, и добавляется один атрибут AK типа integer. Добавляются две дополнительные сущности - A' и A''. Каждая из них будет содержать один атрибут BLOB и один ключевой атрибут K типа integer, который станет внешним ключом у каждого из новых отношений и будет ссылаться на атрибут AK в отношении A (тип внешнего ключа – on delete cascade, on update cascade).

- Жизненный цикл сущности определен в информационной модели соответствующей диаграммой. В описании сущности отсутствует атрибут, который отражает изменение состояния сущности. В этой ситуации проектировщики добавляют атрибут status, для которого определяется ограничение допустимых значений (из списка допустимых состояний), а изменение состояния сущности описывается триггером, проверяющим допустимость сочетания нового и старого значения атрибута.
- Две диаграммы потока данных описывают различные бизнес-процессы, работающие над одними и теми же данными. Допустим, первая диаграмма описывает выписку товара со склада, а вторая – сложный отчет, отражающий состояние склада. Один процесс интенсивно модифицирует данные, второй работает в режиме чтения данных, но требует согласованности данных в течение длительного времени. Каждый из процессов описывается транзакцией над данными. В СУБД уровни изолированности транзакций реализованы так, что читающие транзакции конфликтуют с модифицирующими транзакциями. Это приводит к остановке выписки товаров со склада на время выполнения отчета, что неприемлемо для заказчика. Здесь может потребоваться очень серьезное изменение информационной модели.

Подобных примеров, когда не только ER-модель, но и другие продукты анализа не могут быть перенесены автоматически на модель данных, можно привести множество. Каждый такой случай инициирует изменение информационной модели. Решение проблемы определяется возможностями СУБД, выбранной для реализации проекта. Если проблем, не разрешаемых в рамках данной СУБД, накапливается очень много, то проектировщики могут поставить вопрос о смене СУБД. Такой вопрос поднимается именно на стадии проектирования, поскольку если уже разработчики столкнутся с подобными проблемами, то цена смены СУБД будет выше. Ясно, что одинаковых СУБД не бывает: то, что хорошо работает в одной, может плохо работать или вообще не работать в другой, несмотря на уверения производителей СУБД в поддержке стандартов SQL. Что касается хранимых процедур и триггеров, то здесь вообще трудно говорить о поддержке SQL92/PSM.

Вопросы производительности информационной системы также влияют на отображение ER-модели на модель данных. За счет мощного сервера баз данных можно добиться большей ско-

рости реакции системы, но мощность аппаратного комплекса ограниченна. Производительность системы в целом зависит в том числе и от нормализации. Часто до 80% запросов к базе данных являются выборками данных, а соединение по тому или иному атрибуту относится к затратным операциям, в первую очередь соединение по нечисловым атрибутам. Увеличить производительность системы можно посредством введения избыточной информации – денормализации. Следует отметить, что решение об этом не принимается на основе одной ER-модели – требуется внимательно проанализировать потоки данных. Критичные процессы являются хорошими кандидатами для денормализации: по времени выполнения, по частоте выполнения, по большому объему обрабатываемых данных, по частоте изменения обрабатываемой информации, по явному приоритету. Часто к денормализации прибегают в целях ускорения выполнения отчетов. Для проверки эффективности той или иной денормализации привлекаются тестеры.

Типы данных

Как правило, СУБД поддерживают небольшой набор базовых типов данных: числовые типы (целые, вещественные с плавающей и фиксированной точкой), строки (символов и байт), дата и время (или комбинированный тип datetime), BLOB (и его разновидности, например BLOB-поля для хранения только текста). В информационной модели каждому атрибуту соответствует домен. Поскольку не все реализации СУБД поддерживают домены, то в этом случае при определении модели данных ограничения домена описывают как ограничения столбца таблицы (если такое возможно); в частности используют check constraints, триггеры. Следует отметить, что при определении типов столбцов таблиц нужно учитывать, какие типы данных поддерживаются в словаре данных СУБД. Например, в Oracle ключевые слова integer, smallint, real поддерживаются транслятором SQL, но в словаре данных им соответствуют number(38), number(38), float(63), так как Oracle хранит данные в двоично-десятичном формате с плавающей точкой, а не в двоичном формате с плавающей точкой, и 38-восьмизначное число никак нельзя назвать словом smallint.

СУБД поддерживают два вида строковых типов: с фиксированной длиной (например, char), когда хранится ровно столько символов, сколько указано в описании атрибута, и с переменной длиной (например, varchar), когда хранится реальная длина значения атрибута, а концевые пробелы строки усекаются. Семантика сравнения строк в СУБД также различная, и если ваше мнение

о сравнении строк расходится с тем, как это реализовано в СУБД, то придется смириться с этим как с особенностью СУБД. Например (описано поведение Oracle 7.x), если сравниваются значения A равное 'ab' и B равное 'ab' двух атрибутов типа varchar разной длины, то sql сообщит, что $A \neq B$. Чтобы избежать подобных «фокусов», нужно, в частности, следить за тем, чтобы приложение не вставляло незначащие концевые пробелы в значения атрибутов этих типов.

Индексы, кластеры

В правильно спроектированной базе данных каждая таблица содержит первичный ключ, что означает наличие индекса. В большинстве СУБД используются индексы $B^* - tree$. Отметим, что если используется составной индекс, то поиск по всем атрибутам, входящим в индекс, начиная со второго, будет медленным. Допустим, определен индекс index1(id1, id2), в этом случае поиск значений, удовлетворяющих условию id2=1, будет медленным (не исключено, что оптимизатор вообще не будет использовать этот индекс для обработки данного условия и будет принято решение о полном сканировании данных), а поиск значений, удовлетворяющих условию id1=1 and id2=1, будет быстрым. Данные особенности следует учитывать при определении индексов в схеме базы данных, а именно:

- индексировать нужно атрибуты, по которым наиболее часто осуществляется поиск или соединение. Наличие индекса замедляет операции модификации, но ускоряет поиск;
- наличие индекса обязательно, если для атрибута или набора атрибутов указано ограничение unique. Такие индексы СУБД создает автоматически, если в описании таблицы указаны ограничения unique;
- индекс может быть использован для выборки данных в заданном порядке. В этом случае не вызывается процесс сортировки ответа, а используется уже готовый индекс;
- атрибуты, входящие во внешний ключ, также следует индексировать, если СУБД не делает эту операцию автоматически при декларации внешнего ключа;
- в некоторых СУБД поддерживаются bitmap-индексы, которые очень эффективны при поиске на равенство, но для поиска на $<, \leq, \geq, >$ этот тип индексов не годится;
- в некоторых СУБД поддерживаются хеш-индексы, например для кластеров. Такие индексы эффективно использу-

ются при поиске на равенство.

Кластеризация – это попытка разместить рядом в одном физическом блоке данных те строки, доступ к которым осуществляется при помощи одинаковых значений ключа. Индексные кластеры, например, удобно использовать для хранения родительской и дочерних строк таблиц, связанных ссылочной целостностью. Кластеры удобно определять для тех наборов атрибутов, соединение по которым проводится наиболее часто, поскольку это увеличивает скорость поиска. Следует отметить, что в реализациях СУБД существуют жесткие ограничения на количество кластеров для таблицы, как правило, это один кластер. Особенности реализации кластеров в СУБД необходимо учитывать при проектировании критичных по времени выполнения модулей. Нужно обратить внимание, насколько сильно влияет наличие кластера на производительность DML-операций. Чаще всего это оказывает отрицательное влияние, которое в некоторых реализациях распространяется на DML-операции над любой таблицей базы данных, а не только над той, для которой определен кластер. Эти особенности СУБД также следует учитывать при проектировании.

Для того чтобы выбрать тот или иной тип индекса, требуется внимательно изучить руководство администратора СУБД. Оптимизатор SQL использует различные типы доступа к данным при обработке запросов, и индексирование существенно влияет на выбор оптимизатора.

Приведем некоторые способы доступа к данным на примере выборки select id, name from xtable where id=10:

- Таблица не индексирована. В этом случае применяется полное сканирование, которое не является эффективным, если объем данных большой, в таблице много данных, не удовлетворяющих условию, размер кортежа существенно превосходит размер атрибута id.
- Атрибут id индексирован; тип индекса - $B^* - tree$. Тогда применяется индексное сканирование; полное сканирование может быть выбрано только в случае, если объем данных, удовлетворяющих данному условию, является большим (эта информация анализируется статистическим оптимизатором) и сравним с количеством записей в таблице. Индексное сканирование применяется лишь тогда, когда размер индексированных атрибутов меньше размера кортежа и все необходимые для обработки запроса данные могут быть получены из индекса; в остальных случаях может быть применено полное сканирование.

- Атрибут id входит в составной индекс, и является первым (лидирующим) атрибутом индекса, при этом тип индекса - $B^* - tree$. Аналогично предыдущему примеру, применяется индексное сканирование.
- Атрибут id входит в составной индекс, он не первый атрибут индекса; тип индекса - $B^* - tree$. Индексное сканирование применяется лишь тогда, когда размер индексированных атрибутов меньше размера кортежа и все необходимые для обработки запроса данные могут быть получены из индекса; в остальных случаях применяется полное сканирование.
- Атрибут id индексирован; тип индекса – хеш. Здесь применяется индексное сканирование для поиска на =; если бы условие было id <=10, то применение хеш-индекса для такого поиска не эффективно.
- Атрибут id индексирован; тип индекса – bitmap. Здесь применяется индексное сканирование для поиска на =; если бы условие было id <=10, то применение bitmap-индекса для такого поиска не эффективно.
- Атрибут id является ключом хеш-кластера. В этом случае применяется алгоритм хеширования при поиске блока данных для чтения. При хорошем алгоритме и правильном размере кластера поиск может быть осуществлен за одно чтение; при ошибках в выборе алгоритма и блока кластера это может составить до нескольких тысяч операций чтения блоков.
- Атрибут id является ключом индексного кластера. Здесь применяется индексное сканирование, почти аналогично случаю с индексом $B^* - tree$.
- Таблица кластеризована, id не является ни кластерным ключом, ни лидирующим в составном индексе $B^* - tree$. В этой ситуации для кластера применяется полное сканирование.

Мы привели только некоторые правила выполнения операции поиска в зависимости от наличия и типа индекса. В реализации используемой вами СУБД могут быть приняты иные принципы. Подробности использования типов сканирования при поиске данных даются в руководстве по настройке СУБД и в руководстве администратора СУБД.

А почему бы не проиндексировать все, если индексный по-

иск быстрее полного сканирования? Очевидно, что индекс занимает место на диске, вопрос в том – сколько. Например, индексируется атрибут integer – это 4 байта. Но в $B^* - tree$ кроме собственно значения ключа в индексе хранятся и внутренний идентификатор кортежа, и некоторая служебная информация, так что все вместе может составлять 4-8 байт. Чтобы точно посчитать эту величину для используемой вами СУБД, следует обратиться к руководству администратора: посмотрите размер идентификаторов ROWID (Oracle), RID (DB2) и т.д., а также размер страницы индекса (как правило, это 4 Кбайт).

При выборе стратегии индексации следует придерживаться двух простых принципов:

- чем больше индексов, тем выше затраты на выполнение DML-операций. По грубым оценкам затрат: если принять за 1 работу по вставке строки в таблицу, то работа по вставке той же записи в один индекс равна 2 или 3 (для разных СУБД);
- в $B^* - tree$ любое значение может быть найдено за такое количество операций чтения, сколько уровней у дерева (дерево трех уровней для значений integer, например, содержит порядка 533 731 324 ключей, если страница дерева 4 Кбайт). Такие индексы отлично используются при поиске на =, <, >, <=, >=, between, и достаточно хорошо модифицируются. В bitmap-индексах содержатся готовые битовые векторы, отражающие вхождение или невхождение значения в ответ при поиске на равенство, но такие индексы плохо модифицируются и больше подходят для хранилищ данных, например для индексирования вхождения слов в текстовый документ. Хеш-индексы позволяют осуществлять поиск на равенство, хеш-функция используется для поиска блока кластеризованных данных, содержащего нужные значения. Если алгоритм хеш-функции хорош и размер кластера указан верно, то поиск может быть осуществлен за одно чтение. Эти индексы, как правило, используют для создания кластеров.

Обратите внимание, хранит ли СУБД в индексах NULL. Если NULL в индексе не хранится, то вероятность использования полного сканирования для атрибутов без декларации not NULL резко повышается. Если NULL хранится в индексе (обычно его считают самым большим или самым маленьким при построении $B^* - tree$ и специальным значением при построении bitmap и хеш-индексов),

то выясните, для каких операций поиска индексы будут использованы оптимизатором SQL. Эта информация, как правило, содержится в руководстве администратора. Можно проверить и экспериментально, создав тестовую таблицу с объемом данных примерно 20 тыс. записей (чтобы оптимизатор не выбирал полное сканирование по причине малого объема) и выполнив исследуемый запрос, а затем произвести explain плана запроса (если подобный сервис предоставляется СУБД).

Временные данные

Временными данными, или временными рядами, называют данные, содержащие дату и время. Неправильная обработка таких данных в некоторых СУБД может служить одной из основных причин низкой функциональности и производительности информационной системы. Временные ряды не очень хорошо вписываются в двухмерную реляционную модель. SQL поддерживает соединения, не основанные на равенстве, но большинство разработчиков СУБД ограничиваются эквисоединением. Для временных данных часто приходится соединять таблицы на основе перекрытия одного диапазона дат другим. В SQL не существуют операции, которая позволяла бы задать такое соединение непосредственно.

Ряд атрибутов, например курс валюты или цена товара, изменяются во времени. Такие атрибуты действительны по дате, то есть актуальны только в течение определенного интервала времени, например дня для курса валюты.

Если СУБД позволяет обрабатывать многомерные данные, то обработка временных рядов может использовать эти механизмы и тогда время будет являться одним из измерений. Подобные многомерные процессоры применяются для обработки геофизических и географических данных. В таких системах используются индексы $R-tree$, $C-tree$ и их клоны.

Приведем пример обработки цены товара (код товара, начальная дата, конечная дата, цена):

```
create table prices
(id integer, date_from date not null,
date_to date, price decimal not null,
constraint p_range check date_from < date_to);
```

Отметим, что здесь в отношении не задан первичный ключ, а сама задача определения ключа в таких отношениях отличается сложностью. Известно, что момент изменения цены заранее не известен, этим и объясняется отсутствие ограничения not null для атрибута date_to:

```
select price from prices
```

```
where id = :PRODUCT_CODE
and date_from < :WHEN_DATE
and date_to >=
nvl(:WHEN_DATE, to_date('01/12/4721', 'DD/MM/YYYY');
```

Здесь :PRODUCT_CODE и :WHEN_DATE обозначают переменные включающего языка, дата '01/12/4721' является самой большой из поддерживаемых СУБД (эта дата может быть и другой). Подобные операции лучше оформлять в виде хранимых процедур, функций или претранслированных запросов. В хранилищах данных часто обрабатываются архивные данные, для которых обработка временных рядов также актуальна.

Возникают также вопросы по поводу точности дат, например: они актуальны для курсов валют, биржевых сделок. Следует определить, достаточна ли точность даты до секунды или нет. Существует проблема и с тем, как зарегистрировать сделку, если цена акции постоянно меняется. Некоторые проектировщики в этом случае обрабатывают временной ряд, но есть и более простое решение – отследить цену акции на момент прохождения биржевой транзакции и сохранить эту цену как часть информации, которую модифицировала транзакция. Поэтому сначала следует рассмотреть простые варианты решения задач, и если ни один из них не подходит, то использовать временные ряды.

А также как нужно показывать не известную на текущий момент дату, например момент, когда цена товара перестанет быть актуальной? Сделайте это значение равным NULL или самым большим значением (определите его как default-значение для атрибута). Следует отметить, что значения default в большинстве реализаций СУБД работают только при вставке новых записей. Можно, конечно, создать триггер, который срабатывает после выполнения операции insert или update и преобразует null в наибольшее из допустимых значений даты. Но в этом случае, когда пользователь будет работать с подобной информацией, то может забыть, откуда взялась та или иная дата. Если вы спроектировали схему базы данных и запросы таким образом, то все приложения, работающие с выборками данных, должны отвечать следующим требованиям:

- принудительно конвертировать в null такие даты и не показывать их пользователю;
- использовать соответствующие представления, содержащие decode (или иное средство конвертации «больших» дат);
- вызывать хранимую процедуру, которая выполнит все

нужные преобразования.

Выше только что были перечислены возможные проектные решения. Отметим, что для эффективного поиска следует создать составной индекс с атрибутами (date_from, date_to), но не все СУБД будут использовать такой составной индекс, если для одного из атрибутов допустимы значения null. Поэтому довольно простая для аналитиков задача представления временных рядов может повлечь за собой множество неприятных моментов при проектировании.

Теперь рассмотрим проблему поиска первичного ключа для подобных отношений. Оказывается, что единственный разумный вариант предотвращения дубликатов таков: каждому единичному интервалу соответствует отдельная строка. Для биржевых операций этот интервал может быть равен, например, одной секунде, а в некоторых системах он еще меньше. У таких таблиц только одно преимущество – результат из них можно выбирать при помощи эквисоединения, однако объемы обрабатываемых данных велики.

Использование временных рядов, как правило, является одной из наиболее актуальных тем в разговоре с аналитиками. Какое решение будет лучшим – зависит от используемой СУБД и от ее особенностей, а именно: оптимизатора запросов, особенностей использования индексов, мощности SQL, хранимых процедур и триггеров.

Хранение объектов данных

Это одна из самых сложных задач проектирования схемы базы данных, для решения которой привлекаются администраторы баз данных. Универсальных решений, которые подошли бы для любой СУБД, не существует, так каждый производитель СУБД создает свой способ хранения и доступа к данным, считает его лучшим и очень им гордится. На проектирование схемы базы данных влияют следующие параметры, общие для большинства СУБД:

- размер табличных пространств для хранения таблиц;
- размер табличных пространств для хранения индексов;
- размер табличных пространств для хранения BLOB;
- кластеры и их параметры;
- размер словаря данных, включая код всех хранимых процедур, функций, триггеров, пакетов, статического SQL (реализован только в DB2);
- управляющие файлы;
- файлы журнала;
- интенсивность потока запросов, модифицирующих данные

и индексы;
- фалы временных табличных пространств (для хранения временных таблиц, которые строятся, например, при выполнении group by, а также других временных объектов);
- интенсивность потока запросов, инициирующих создание временных таблиц;
- потоки транзакций read-write, read-only, объем модифицируемых и считываемых ими данных, характеристики параллельной работы транзакций (какие и сколько их);
- количество приложений, работающих параллельно с базой данных;
- количество соединений с базой данных для каждого приложения;
- файлы параметров старта ядра СУБД;
- загрузочные модули ядра СУБД и утилиты СУБД;
- входные и выходные данные, генерируемые пользовательскими программами;
- скрипты управления СУБД.

Постарайтесь описать эти параметры хотя бы в общем виде, обсудите все вопросы с администратором баз данных и внимательно выслушайте его рекомендации. Если же его мнение может расходится с вашим представлением о том, как должна работать СУБД, попытайтесь понять его аргументы и запомните, с какими особенностями СУБД это связано. Все данные следует отразить в журнале проектирования. Следует иметь в виду, что многие нюансы размещения объектов данных и конфигурации сервера баз данных не могут быть учтены на этапе проектирования, так как требует полномасштабного тестирования. Конечно, избежать некоторых ошибок проектирования можно, но приготовьтесь к тому, что схема базы данных будет меняться и на этапе реализации, причем неоднократно.

Защита данных

Стратегия защиты определяется на этапе анализа, а на этапе проектирования предстоит реализовать эту стратегию, спроектировав соответствующие структуры в схеме базы данных и модули. Большинство СУБД имеют развитые средства дискреционной защиты, а ряд СУБД имеют встроенные подсистемы аудита, что освобождает от необходимости создания собственных средств защиты.

Обычно СУБД предоставляют набор пакетированных привилегий для управления данными, например: connect, которая разрешает соединение с базой данных; resource, которая дополни-

тельно разрешает создание собственных объектов базы данных, dba, которая позволяет выполнять функции администратора конкретной базы данных, и др. Дискреционная защита предполагает разграничение доступа к объектам данных (таблиц, представлений, и т.п.), а не собственно к данным, которые хранятся в этих объектах. Дискреционная защита также обеспечивает создание пользовательских пакетированных привилегий – ролей или групп привилегий. В этом случае набор привилегий на те или иные объекты данных назначается группе или роли, а затем эта группа или роль назначается пользователю; таким образом пользователь получает привилегии на выполнение тех или иных операций над объектами данных косвенно – через группу или роль.

В некоторых реализациях допускается выполнение определенного набора операций от имени другого пользователя и с его привилегиями – в частности вызовы пакетов и хранимых процедур. Пользователь, вызывающий такой объект, не может ни изменить пакет или хранимую процедуру, ни получить доступ к ее коду, но операции над данными, которые выполняются в хранимой процедуре или пакете, будут выполнены от имени владельца хранимой процедуры или пакета. То же самое относится и к триггерам.

Некоторые проектировщики очень любят использовать описанный выше метод защиты данных – это похоже на инкапсуляцию. Иногда проектировщики строят на каскадирующих вызовах хранимых процедур весьма сложные протоколы доступа к данным, имитируя мандатную защиту. Такой подход имеет свои преимущества и недостатки. С одной стороны, любой доступ к данным скрыт хранимой процедурой или пакетом, но с другой – в этом случае словарь данных сильно перегружен. Однако далеко не все реализации СУБД хорошо работают с курсорами в хранимых процедурах и пакетах, поскольку это вызывает чрезмерную загрузку процессора. Кроме того, в большинстве реализаций СУБД предложения SQL, выполняемые из хранимой процедуры или пакета, имеют более высокий приоритет, чем операции SQL, выполняемые из приложения пользователя. В связи с этим большое количество вызовов хранимых процедур может существенно замедлить выполнение запросов непосредственно из приложений пользователя. В любом случае, чтобы сделать защиту данных, «закрывая» любой запрос хранимой процедурой, требуется, чтобы СУБД имела достаточно развитый язык и позволяла, например, выполнять синтаксический разбор и чтобы внутри самой хранимой процедуры можно было строить как статические, так и дина-

мические предложения SQL.

К сфере защиты данных относятся также сохранность данных и восстановление их после сбоя системы. Для обеспечения бесперебойной работы часто применяют архивирование (в том числе инкрементное) базы данных и журнала транзакций, а в случае отказа системы при следующем старте операции над данными восстанавливают по журналу транзакций (например, производят их откат до определенного момента времени). Применяют также методы горячего резервирования, когда работают два сервера: основной, обрабатывающий запросы пользователей, и резервный, который продолжает работу основного сервера в случае его отказа. Состояние хранилищ данных на основном и резервном серверах согласовано и поддерживается СУБД автоматически, что позволяет проектировщикам не разрабатывать собственные механизмы репликации данных.

Работа серверов в режиме горячего резервирования не избавляет от необходимости хранения резервных копий данных, это может быть и не очевидно для аналитиков и не предусмотрено ими. Некоторые бизнес-процессы по своей природе требуют от информационной системы работы в режиме 24x7, и любой простой стоит очень дорого. В этих случаях работают две или три параллельные системы, и при отказе одного из серверов резервные серверы немедленно принимают управление на себя. Эффективным, но дорогостоящим способом реализации таких задач являются предоставляемые СУБД технологии симметричной репликации. Еще один вариант – архивирование журналов транзакций на резервном узле на специальное устройство и немедленный докат по этому журналу резервного узла в случае отказа основного. Разные СУБД предлагают разные механизмы реализации подобной бесперебойной работы, и для принятия верного проектного решения необходимы консультации проектировщиков с администраторами баз данных.

В простых ситуациях, когда информационная система используется в основном для операций чтения данных, а сами данные меняются редко, резервное копирование может вообще не требоваться, если данные одной такой системы могут быть легко восстановлены из данных других работающих систем. Достаточно будет обеспечить наличие образа базы данных (архив всех файлов базы данных, а также управляющих файлов – это должен быть снимок базы данных на определенный момент времени; проще всего такой снимок получить, остановив СУБД и сделав резервную копию всех указанных файлов).

В большинстве информационных систем необходимо обеспечение безотказной работы системы в ситуации, представляющей собой нечто среднее между описанными выше двумя крайними случаями. Проектировщики должны пересмотреть результаты анализа, исходя из ответов на следующие вопросы:

- каков график необходимой доступности системы для запросов пользователя (то есть когда система обязательно должна работать);
- допустимы ли вообще и когда допустимы периоды профилактического простоя системы;
- допустимы ли и когда допустимы периоды ограничения доступа к системе;
- какие данные после отказа системы нельзя получить из других источников (часто это ввод новых документов, например накладных, операции со счетами, заказы на телефонные переговоры, информация с автоматизированных датчиков и т.п.);
- если данные можно получить, то каков объем повторно вводимой информации;
- каково допустимое время восстановления системы после сбоя;
- имеется ли график пакетных суточных заданий;
- какие еще приложения, кроме информационной системы, работают на данном оборудовании;
- имеются ли резервные аппаратные средства на случай отказа основных;
- имеется ли запас мощности оборудования, на котором функционирует информационная система;
- какова скорость передачи данных при резервном копировании;
- имеются ли специальные отказоустойчивые носители для хранения резервных копий;
- имеется ли правило циклического использования резервных носителей;
- имеется ли специальное защищенное место для хранения резервных носителей.

Ответы на эти вопросы позволят более реально оценить ситуацию и уточнить требования заказчика, формализованные аналитики. Бывает, что заказ работоспособности системы в режиме 24x7 вовсе не является обоснованным и система простаивает, например, 50% времени. Если же требование 24x7 действительно отражает особенности данного бизнеса, то эти вопросы помогут

построить соответствующую стратегию защиты данных от сбоев. Качество построенной при проектировании стратегии защиты должно быть проверено тестерами, причем их работа по генерации и проведению тестов, имитирующих отказы оборудования, должна проводиться как на этапе проектирования, так и в течение всего этапа разработки – в целях раннего обнаружения дефектов стратегии защиты данных от сбоев.

Обмен данными с внешними системами

Большинство аналитиков считают задачи обмена с внешними системами чисто физическими, решенными априори и, как следствие, не уделяют этим вопросам внимания при анализе. Поэтому зачастую проектировщики вынуждены нести на себе всю нагрузку по созданию таких систем обмена.

Импорт и экспорт данных во внешние системы могут обеспечиваться как утилитами, поставляемыми в составе СУБД, так и специальными средствами обмена данными проектируемой информационной системы. Прежде чем писать код собственно системы обмена данными, внимательно изучите сервис, который предоставляет СУБД, и используйте его полностью. Если обмен данными идет между двумя однородными базами данных (одного производителя и одной версии), то средства импорта и экспорта данных во внутреннем формате данной СУБД могут быть использованы без ограничений. Если же обмен данными осуществляется между СУБД разных производителей, то для импорта и экспорта собственно данных можно использовать текстовые файлы. Следует при этом обратить особое внимание на особенности представления null, пустых строк символов и BLOB, так как здесь потенциально присутствует расхождение интерпретации содержимого файлов загрузчиками данных.

Импорт и экспорт схем может оказаться более сложным делом. Производители СУБД часто заявляют о поддержке стандартов ANSI, но на практике большинство баз данных не удовлетворяют им полностью. Синтаксисы операций создания таблиц существенно различаются, а синтаксисы определения хранимых процедур и триггеров большинства СУБД вообще несовместимы. Информация о схеме базы данных может быть получена из системного каталога посредством анализа данных в системных представлениях, но и здесь нет полной поддержки стандарта. Когда информационной системе требуется обмен данными между СУБД разных производителей, то, как правило, проектировщики принимают решение о создании специализированного средства. Даже если оно не будет полнофункциональным и не может быть при-

менено для решения задачи конвертации данных между любыми СУБД, то решить локальную задачу оно способно.

Интерфейсы обмена с внешними системами можно разбить на следующие категории:

- одноразовый импорт данных, унаследованных, как правило, из старой системы;
- периодический обмен данными между узлами информационной системы (внутренний обмен);
- периодический обмен данных с другими информационными системами (внешний обмен).

Если обмен данными должен осуществляться в режиме, близком к реальному времени, то это будет задача о распределенной базе данных, а не о простой передаче данных.

При анализе задач загрузки и выгрузки данных проектировщик должен рассмотреть:

- каким подсистемам нужен интерфейс выгрузки данных и каков должен быть интерфейс загрузки данных из внешней системы;
- каковы периодичность обмена данными и объем передаваемых данных;
- какая требуется степень синхронизации двух систем;
- каковы возможные методы транспортировки данных;

а также:

- согласовать формат данных для обмена;
- определить зависимости загрузки и выгрузки, например порядок выполнения операций;
- определить мероприятия, которые необходимо выполнить при сбое во время загрузки и выгрузки данных;
- сформулировать правила определения ошибочных записей (при загрузке);
- определить правила регистрации операций передачи и приема данных;
- определить график передачи данных (в большинстве информационных систем эти операции выполняются в ночное время);
- составить график разработки и тестирования собственных утилит или скриптов обмена данными;
- составить график разовой загрузки данных, наследуемых из старой системы, и подготовить методику проверки корректности этой операции.

Следует отметить, что при наследовании данных из старой системы проектировщикам не приходится надеяться на то, что

кто-то создаст утилиту, позволяющую достать данные из старой системы, - обычно это становится задачей самих проектировщиков новой системы. Может случиться так, что вам придется работать в жестких условиях, когда не будет возможности выделить время для тестирования новой программы извлечения данных. В этом случае нужно разработать набор тестовых данных. Если в старой системе имеется какое-то средство извлечения данных – используйте его; часто это самый разумный выход.

При загрузке данных из старой системы проектировщики могут столкнуться с большим объемом неочищенных данных – с нарушениями целостности данных, возникшими из-за сбоев системы, «заплаток» разработчиков, иных неприятностей. Возможно, что на вас будет оказано давление с тем, чтобы допустить наличие неочищенных данных в новой системе. Если не принять мер по очистке данных, то, вероятно, большинство спроектированных ограничений целостности нужно будет ослабить, чтобы загрузить хоть какую-то часть данных. Цена такой уступки достаточно высока: данные вы приняли, но ослабленные ранее ограничения уже нельзя восстановить, так как они уже нарушены (это отслеживается СУБД автоматически). Отсюда следует вывод: поддаваться давлению нельзя, так как несколько дней, потраченных на очистку данных, стоят так мало по сравнению с наличием в информационной системе данных, не обладающих элементарной целостностью.

Что делать с данными, которые содержат ошибки или не согласованы? Самое простое решение – пропускать такие данные, собирать их отдельно и анализировать. Здесь вас могут ждать некоторые проблемы: не все СУБД при загрузке данных их собственными утилитами позволяют в случае возврата кода ошибки указать запись, на которой произошел сбой. Если это так, то данные следует загружать небольшими порциями, чтобы можно было легче найти запись, которая повлекла сбой. Можно разместить данные с нарушениями целостности в отдельных таблицах, а потом обработать их. Подобную операцию (которую аналитики, как правило, не предусматривают) лучше автоматизировать посредством отдельного компонента. Проектировщикам придется либо озадачить аналитиков исследованием правил корректности данных, либо выполнить эту работу самим, причем необходима помощь опытных пользователей старой информационной системы. Здесь крайне важно найти данные, которые являются надежными, то есть те, которые с большой вероятностью указаны правильно. От таких данных и надо отталкиваться при создании программ

проверки корректности данных.

Далеко не все СУБД предоставляют средства, позволяющие выгрузить схему данных, хранимые процедуры и триггеры, но имеется достаточное количество ПО сторонних фирм, способного на это. Самым привлекательным будет ПО, осуществляющее выгрузку указанных объектов в виде скриптов, описывающих их создание. Если ПО позволяет сначала создать объекты данных без ограничений ссылочной целостности, а потом изменить их, добавив эти ограничения, то такой опцией рекомендуется пользоваться всегда. Дело в том, что большинство утилит выгрузки схемы опрашивают системный каталог и генерируют скрипт создания таблиц в том порядке, в котором имена таблиц встречаются в системном каталоге. Но вы не застрахованы от того, что имя таблицы-потомка будет считано раньше имени таблицы-предка. От исправления операций создания объектов данных вы также не застрахованы, так как у большинства СУБД разный синтаксис определения ограничений ссылочной целостности, серверных ограничений, индексов. Например, перечисленные ниже определения ссылочной целостности для таблицы tx:

create table t(id int primary key, name char(10);

create table tx(i int, j int references t(id));

create table tx(i int, j int references t(id) on delete no action on update no action);

create table tx(i int, j int, foreign key j references t(id) on delete no action on update no action);

create table tx(i int, j int, foreign key j references t on delete no action on update no action);

create table tx(i int, j int, constraint t_ref_cascade foreign key j references t(id) on delete no action on update no action);

все означают в точности одно и то же, но синтаксис у них разный. Для преобразования предложений SQL в требуемый формат могут быть созданы специальные утилиты. Поскольку в этом случае требуется разбор текста, то весьма нецелесообразно применение инструмента Perl. К сожалению, операция переноса хранимых процедур и триггеров не может быть в такой же степени автоматизирована вследствие существенных различий в языках их написания в разных СУБД. Некоторые СУБД совместимы по синтаксису хотя бы частично, но если этого нет, то большой доли ручной работы не избежать.

Как правило, процесс проектирования модулей кода ведется параллельно с проектированием схемы базы данных. Дело в

том, что немногие проектные решения схемы можно принять без согласования с проектными решениями модулей. Редко проектирование модулей состоит из единственного этапа, в большинстве случаев это процесс итерационный с постоянным уточнением тех или иных моментов.

Обработка иерархии функций

На этапе анализа была создана иерархия функций, которая дополняется диаграммами потока данных и изменения состояний. Как проще всего проверить, является ли функция в иерархии атомарной? Ответьте на вопрос: имеет ли смысл выполнение только части этой функции. Поскольку мы имеем дело с иерархией, вероятно наличие одной и той же функции в нескольких местах иерархии. Здесь кроется ошибка анализа: аналитик перепутал функцию и механизм.

Хорошо, если при описании функции аналитик укажет в скобках тот объект в ER-модели (сущность и атрибут), о котором идет речь, например «выбрать товары в накладной из списка товаров (ITEMS), хранящихся на складе (STORE)». Если подобных указаний нет, это станет хорошим упражнением для приемки отчетов аналитиков; заодно и убедитесь, что вы правильно понимаете друг друга. Параллельно проверьте, есть ли сущности, которые не используются ни в одной функции, - это поможет сделать матрицу «функции-атрибуты», отражающую факт использования в функциях атрибутов сущностей.

Как правило, не бывает взаимно однозначного отображения функций на модули. Если однозначных соответствий много, то вероятнее всего был выполнен не анализ, а собственно проектирование. Зачастую схожие по выполняемым действиям функции объединяют, даже если у них разный контекст. Некоторые сложные функции разбивают на более простые модули. Некоторые функции преобразуют в ограничения базы данных (constraints, или триггеры и хранимые процедуры). Каких-то общих способов отображения функций на модули не существует. Требуется время, терпение, опыт. Проектировщики часто меняют количество, состав модулей в течение процесса проектирования, и это правило, а не исключение.

Некоторые группы модулей присутствуют в любом проекте:

- работа с базой данных:
- обработка кодов возврата СУБД;
- установка соединения с СУБД и его параметры;
- выполнение некурсорного запроса (DDL- и DML-операции);

- выполнение выборки;
- вызов хранимой процедуры и обработка ее ответа;
- операции начала и завершения транзакции;
- функциональные модули:
- модули, реализующие бизнес-процессы;
- модули, изолирующие обработку запросов к базе данных и интерфейсы, предоставляемые пользователю;
- системные модули:
- средства управления приложениями;
- средства обмена данными с внешними системами;
- планировщик пакетных заданий;
- менеджер печати документов;
- модуль резервного копирования;
- модуль архивирования;
- модуль автоматического восстановления при сбоях;
- средства администрирования пользователей системы (регистрация, назначение прав, внешняя аутентификация);
- средства создания нерегламентированных запросов к базе данных.

Один из приемов построения расширяемых систем состоит в создании независимости двух слоев: обработки запросов и интерфейса, предоставляемого пользователю. Можно специально кодировать каждый запрос данных, например именем функции и номером или как-нибудь иначе. Текст SQL-запроса скрыт от разработчиков интерфейсов; они имеют доступ только к возвращаемому запросом множеству – коду ошибки или выборке. Формат кода ошибки и выборки фиксируется. Это позволяет проектировщикам схемы базы данных изменять как ее, так и сам SQL-запрос, однако эти изменения не отражаются на процессах создания интерфейсов. Аналогичная независимость реализуется для слоя функций, обеспечивающих вызовы интерфейса, предоставляемого СУБД для выполнения запросов. Этот слой функций может использовать вызовы как native-интерфейса СУБД, так и стандартных интерфейсов, например ODBC.

Очень важен слой функций обработки ошибок. При выполнении запросов к базе данных всегда следует предусматривать обработку исключений, например нарушений ограничений целостности. Отдельно требуется предусмотреть обработку конфликтов транзакций и принудительный откат текущей транзакции вследствие разрешения конфликтов типа deadlock. Проектировщики должны хорошо понимать особенности многопользовательской работы, которые реализованы используемой в проекте СУБД.

Проектирование потоков транзакций и снижение конфликтов между ними – одна из самых сложных задач проектирования.

Для больших проектов важно наличие функций мониторинга работы приложений. Они служат в том числе и для оценки корректности работы самих приложений, а также для сбора статистики отказов системы и прогнозирования вероятных отказов. Такие функции отслеживают моменты перегрузки системы и автоматически разрешают их, уведомляя администратора о принятом решении. Подобные модули могут быть реализованы в виде локальных агентов, каждый из которых работает на одном сервере, а сами они обмениваются информацией друг с другом и центральной программой управления.

Управление исходным кодом

Для групповой разработки важны системы контроля исходного кода. Такие системы решают по крайней мере две задачи: хранение всех версий каждого экземпляра исходного кода (версии файлов) и разрешение конфликтов одновременного доступа разработчиков к одному экземпляру исходного кода (слияние исходных текстов, согласованность группы файлов проекта и т.п.)

Контролю подвергается не только собственно код модулей, но и код скриптов генерации схемы базы данных, а также собственно схема базы данных, которая может храниться в виде версий бинарных файлов, экспортированных во внутренний формат СУБД схем баз данных. Дело проектировщиков – предусмотреть наличие таких механизмов разработки, дело прикладных программистов – придерживаться установленных правил групповой разработки. Часто ведущие исполнители проектов не доверяют системам контроля исходного кода сливать правки в исходных текстах автоматически и частично или полностью контролируют этот процесс. Эта перестраховка во многих случаях себя оправдывает.

После того как основной код модуля будет создан и зарегистрирован в системе контроля исходных текстов, каждая правка в нем должна быть помечена датой и именем автора правки. Правки должны сопровождаться ясными комментариями, которые располагаются в начале файла исходного кода. Также в начале исходного кода (в комментариях) описывается: для чего данный файл исходного кода создан, основные его функции, к какой части информационной системы он относится, кто автор. Формат комментария к правке может быть таким

10.01.2000 Ivanov: authorization bug fixed (found by Petrov)

Это делается для того, чтобы через некоторое время можно

было понять, кто и зачем внес данную правку. Также это помогает корректировать слияние исходного кода, если система контроля обнаружила несовместность правок и не может разрешить ее сама.

Функции, структуры, наиболее важные переменные должны сопровождаться комментариями. Избегайте непонятных названий вида K1, Function10.

Для крупных подсистем следует зафиксировать интерфейс обмена с другими подсистемами – формат данных, передаваемых подсистеме и получаемых из нее, а также формат вызовов функций и методов подсистемы. Это позволит добиться относительной независимости подсистем и снизить влияние изменений кода одной подсистемы на другую. Такой подход облегчает взаимодействие разработчиков разных модулей. Документировать придется только интерфейс обмена, а не весь код модуля целиком. Зачастую разработчику, вызывающему функцию модуля, требуется знать, что нужно передать и как обработать результат, а что происходит внутри – знать не обязательно. Единственное, что его интересует, - это правильная инициализация модуля, правильный прием результатов его работы и правильная выгрузка модуля. Взаимодействие компонентов не должно быть произвольным. Контролировать эти правила создания исходного кода в большинстве случаев приходится вручную.

Следует отметить, что группа разработчиков должна иметь свой выделенный сервер баз данных, и, возможно, не один, а также выделенные рабочие места. Часто эти моменты далеко не очевидны руководству, и оно воспринимает это как совершенно ненужную трату средств. Сервер, обеспечивающий контроль исходного кода, также должен быть выделенным.

Размещение логики обработки

В отчетах аналитиков часто смешиваются три группы правил: правила для данных, процессов и интерфейса. На этапе проектирования эти правила предстоит выделить.

В правилах для данных формулируются условия, которым всегда должны удовлетворять данные. Такие правила безоговорочны; они могут быть неверными, но в этом случае они безоговорочно неверные. Правила данных определяются в схеме базы данных. Они могут контролироваться приложением только в том случае, когда мощности языка данных не хватает для их поддержки (например, поддержка сложных ограничений check, содержащих агрегатные функции, и т.п.).

Правила для процессов определяют, что должно и что не

должно делать приложение, и выводятся из модели функций, переданной аналитиками.

В правилах для интерфейсов устанавливается, что должен видеть конечный пользователь. Такие правила не касаются обработки данных, они выводятся из спецификации пользовательского интерфейса (например, к ним относится внешний вид экранных форм отчетов и документов).

Ниже приведем несколько правил, переданных аналитиками, и классифицируем их. Дело в том, что при проектировании группы правил не должны перемешиваться. Итак:

1. Только руководитель может санкционировать выплату вознаграждения.

Это правило только для процессов. В момент разрешения операции приложение должно проверить, есть ли у пользователя привилегии, разрешающие операцию, и нет ли привилегий, запрещающих операцию. Многие проектировщики пытаются реализовать такие правила, как правила данных. Но связь здесь зависит от времени, а не является постоянной (руководитель может быть смещен, подчиненный может стать руководителем, для временных групп работников это процесс постоянный).

2. Обновление записей о платежах запрещено.

Это правило для данных. Оно не изменяется во времени. Такое правило можно реализовать с помощью триггера before update, он должен возвращать ошибку вызвавшему приложению. Откат транзакции внутри триггера в этом случае возможен только при отсутствии сложных транзакций, операции которых инициируют вызов данного триггера.

3. Все коды валют должны раскрываться; рядом с кодом указывается полное название валюты.

Это правило для интерфейса. На первый взгляд оно разумно, но практика показывает, что в большинстве случаев пользователи оперируют кодами, а не полными названиями валют. В большинстве организаций есть система сокращений, ставших частью языка и понятных всем. Такие сокращения не требуют раскрытия.

4. Все торговые операции, выполненные в воскресенье, учитываются в бухгалтерской книге за следующий понедельник.

Подобные правила часто описываются аналитиками. На самом деле здесь два правила в одном, и их надо разделить. Первое гласит о том, что в бухгалтерских книгах не должно быть проводок, сделанных в воскресенье. Это правило данных. Оно может

быть реализовано ограничением check. Второе правило – для процессов. Оно объясняет, как откорректировать дату: чтобы дата бухгалтерской проводки была правильной для «воскресенья», требуется прибавить один день. Это позволяет избежать случая, когда в приложении появляется оператор insert с заведомо отвергаемыми данными. Обработку такой ситуации можно предоставить хранимой процедуре, если ее язык достаточно мощный и допустимы вызовы внешних функций (в данном случае проверка дня недели) или имеется встроенная функция обработки календаря. Если же решение в виде хранимой процедуры невозможно, то это преобразование должно выполнить само приложение (вызов библиотечной функции или соответствующая организация объектов – как больше нравится). Другое решение подобной задачи – добавление производного атрибута. Вместо одного атрибута, хранящего дату, получаем два: один хранит реальную дату, второй – откорректированную для «воскресенья»; последний является производным. Это допустимо, если подобная денормализация не влечет за собой много аномалий модификации.

5. Пособие не должно выплачиваться лицу, если его доход превышает 300 руб. на человека.

Это правило для процессов. Здесь не говорится о том, какие данные образуют доход, в разные периоды времени источники дохода разные для одного и того же лица, состав семьи также меняется с течением времени.

Точно такие же группы правил могут быть применены и для проектной документации. Эти группы могут отражать естественное разделение разработчиков на тех, кто реализует работу с базой данных, системные модули и интерфейсы пользователя.

Полное описание интерфейсов на этапе проектирования возможно, если требования пользователя достаточно четко определены. На практике реализация интерфейса пользователя является наиболее часто изменяемой частью исходного кода. На этапе проектирования следует описать наиболее общие правила интерфейса, например горячие клавиши, используемые одинаково во всех программах.

Следует отметить, что много проблем в интерфейсах пользователей создают сами проектировщики, если они неправильно выбирают макет и не учитывают разницы поведения системы на макете и на реальных данных. Наиболее частая ошибка проектирования интерфейса – отображение данных в форме для редактирования и блокирование их средствами СУБД до тех пор, пока пользователь не нажмет кнопку ОК. Еще одна распространенная

ошибка проектирования интерфейса – обработка длительных процессов, когда пользователь должен ждать ответа на запрос. Большинство проектировщиков не предусматривают единого вызова функции- обработчика события ожидания ответа. На макете запрос может проходить мгновенно, а в случае реальных данных это может составлять несколько минут. Если обработка ожидания ответа не предусмотрена (хотя бы в виде элементарного сообщения «подождите, идет обработка данных»), то пользователь думает, что приложение «зависло».

С точки зрения логики расположения правил они должны быть реализованы так:

- правила интерфейса реализуются во внешней системе, например в Delphi, генераторах экранных форм и отчетов, поставляемых в составе средств разработки СУБД;
- правила для процессов могут быть реализованы как процедуры, вызываемые из внешней системы;
- правила данных следует реализовать средствами СУБД с помощью ограничений целостности.

Следует отметить, что место правил интерфейса и правил данных всегда задано точно. Место правил процессов не всегда определено точно. Часть из них может быть реализована как вызовы утилит, поставляемых с СУБД или созданных другими группами разработчиков; часть может храниться в схеме как процедуры или пакеты; другие могут быть реализованы как библиотечные функции собственно информационной системы. Но любая процедура, реализующая правило процесса, должна быть отделена от кода, реализующего правила интерфейса, и не зависеть от него.

Вопросы проверки корректности ввода информации часто решаются проектировщиками неоднозначно. Как правило, пользователь настаивает на более детальном анализе ошибки ввода данных. Если вводимые значения выбираются из справочника, то здесь проблем меньше. Если они вводятся вручную, то возникает проблема предотвращения попадания в систему некорректных данных, например элементарных опечаток. Выбор значений из справочника – это форма реализации ограничений на уровне интерфейса. Проектировщики часто выбирают – делать проверку на уровне интерфейса или на уровне ограничений базы данных. Лучше реализовать оба правила: и интерфейсное – поскольку пользователь требует немедленной обратной связи, и правило данных – как дополнительную проверку корректности.

При проектировании интерфейсов старайтесь придерживаться стандартов де-факто. Ими могут быть приложения, часто

используемые пользователем, например вызов контекстной подсказки по F1 и т.п.

При работе нескольких пользователей с одними и теми же объектами данных проектировщикам приходится решать задачи совместного редактирования документов, например оформление заказа. Клиент не всегда точно определяет список товаров и их количество, а потому оформление заказа может требовать некоторого времени (и оно больше, чем чистое время заполнения экранной формы оператором). Проектировщики допускают распространенную ошибку решения интерфейса пользователя для таких задач: в экранной форме отображается ввод позиций заказа, которые выбираются из справочников; выбранные данные блокируются до тех пор, пока оператор не нажмет ОК. Это приводит к возникновению феномена «конвоя». А именно – несколько операторов после нажатия ОК начинают ждать разрешения конфликтов средствами СУБД, в то время как большое количество конфликтов спровоцировал именно интерфейс. В самом деле время редактирования формы намного превосходит время обмена данными приложения пользователя и СУБД, соединение с СУБД простаивает до 99% времени – то есть ждет запроса, тогда как блокировка данных остается. Чем дольше время блокировки, тем больше вероятность конфликта. Здесь у некоторых проектировщиков возникает идея обвинить СУБД во всех грехах: она же блокирует, а мы вроде и ни при чем. Предположим, что СУБД не блокирует редактируемые данные, и они вступают в силу только по нажатию кнопки ОК. Это, конечно, хорошо, но в течение времени редактирования другой пользователь мог изменить те же данные и зафиксировать свои изменения. СУБД транзакцию второго пользователя уже не пропустит – это типично для стратегии оптимистических блокировок. Возникает вопрос: если и так плохо, и так нехорошо, что делать? Проблема – в неверном взаимодействии интерфейса и обработки данных. Транзакция в любой СУБД начинается или явно, или по факту первого запроса данных. При описанном решении интерфейса транзакция оказывается слишком длинной. Ведь когда идет формирование списка заказываемых товаров, выполняются запросы к данным и транзакция уже начата. Можно принудительно разорвать операцию формирования заказа и его подтверждения. Здесь используют стандартные методы определения зон риска: так, в случае заказа товара это вход в зону потенциальной нехватки (например, заказали 90% товара, имеющегося на складе, - это следует отметить как сигнал потенциального риска продажи товара два раза). По кнопке ОК выпол-

няется подтверждение ранее зарезервированного товара и в результате вероятность конфликта снижается.

Аналогично решается задача одновременного редактирования двумя пользователями одного документа. Изменения пользователя (которые он сделал в экранной форме) запоминаются в специальном буфере данных; то, что он первоначально получил для редактирования, также запоминается в буфере начальных данных. По нажатию кнопки ОК выполняется попытка зафиксировать изменения пользователя. На уровне правил данных (разрешение конфликтов транзакций) выполняется проверка отсутствия зафиксированных другим пользователем изменений. Если данные были кем-то изменены во время редактирования, то пользователь получает предупреждение об этом. Ему может быть предложено просмотреть изменения, которые находятся на данный момент в базе данных, и в зависимости от этого сохранить свои изменения поверх или отказаться от них. Естественно, это не требует от пользователя повторного заполнения всей формы (что очень раздражает пользователей).

Трехуровневая архитектура

Приложение разделяется на три части: 1) управление интерфейсом пользователя; 2) выполнение правил обработки данных; 3) выполнение функций сохранения и выборки данных.

Данная архитектура позволяет четко разделить правила процессов и правила данных. Правила процессов реализуются исключительно на втором уровне. Этот уровень может представлять собой выделенный сервер приложений, который имеет право доступа к базе данных, а приложение пользователя обращается к базе данных только опосредовано. Одно из преимуществ такой архитектуры – использование нескольких СУБД в качестве хранилища данных. Все три части комплекса имеют фиксированные интерфейсы обмена данными, следовательно, имеет место изолированность уровня интерфейса пользователя от уровня базы данных за счет наличия ПО промежуточного слоя.

Современные СУБД позволяют реализовывать ПО промежуточного слоя как посредством мониторов транзакций (CICS, Enchina, Tuxedo и др.), так и на самом сервере баз данных в виде хранимых процедур и пакетов (частично или полностью). В этом случае код, реализующий интерфейс пользователя, не содержит вызовов предложений SQL, они «спрятаны» в код пакета или хранимой процедуры (здесь можно говорить об инкапсуляции, что некоторые авторы и делают). В таких случаях для решения пользовательского интерфейса применяют рекомендуемые, а не обя-

зательные проверки правил. Данные при этом блокируются (они вообще не связаны с хранилищем данных) до тех пор, пока пользователь не захочет зафиксировать свои изменения (кнопка OK), а собственно изменения передаются атомарной транзакцией. Это позволяет эффективно использовать мониторы транзакций. Подобные решения рекомендованы для OLTP.

Метрики генерации модулей

Одна из задач проектирования кода – оценить, сколько времени на это нужно и какие средства будут при этом использоваться.

В большинстве проектов оценку времени разработки производят дважды:

- на основе аналитической документации. Здесь привлекаются разработчики, но не на уровне: «За сколько сделаешь? – А за сколько надо?». Следует учесть не только средства разработки, но и аппаратное обеспечение. Это наиболее оптимистические оценки; хорошо, если они будут превышены только на 50%;
- после выполнения большей части проектирования схемы данных и модулей. Этот набор оценок предполагает более глубокую детализацию.

Если вы ни разу таких оценок не делали и вам непонятно, с чего начать, то разделите модули по группам: генерируется автоматически, простой, средней сложности, сложный, очень сложный.

Метрика – это таблица плановой трудоемкости (столько-то дней и столько-то человек требуется). В метрике учитываются как минимум три составляющие: проектирование модуля, генерация его кода, тестирование модуля (в которое входит и тестирование связей модулей). В метрику лучше включить больше условий – это станет своеобразной страховкой от отставания от графика. Следует учитывать как минимум следующие факторы:

- стабильность модели данных и степень ее изменения в течение разработки;
- стабильность модели функций и степень ее изменения в течение разработки;
- уровень квалификации персонала;
- пригодность выбранных средств разработки;
- использование автоматических генераторов кода (например, экранных форм и отчетов);
- соответствие среды требованиям средств разработки (станции разработчиков, серверы, сеть, операционные си-

стемы и т.п.).

Следует с очень большой осторожностью относиться к модулям, которые отмечены как сложные и для которых выбрано относительно низкофункциональное средство. Учтите также зависимости модулей и возможности накопления отставания от графика.

Мегамодули

Это весьма распространенная особенность интерактивных систем. Создаются сложнейшие экранные формы с десятками страниц, один DML-запрос инициирует пару десятков триггеров и т.д. Задачу уменьшения сложности модулей сложно решить, если используются средства ускоренной разработки приложений. Решите, что вам нужно в этом модуле, а что – нет. И вряд ли мегамодуль – это то, о чем мечтал пользователь. Он вряд ли обрадуется, если будет листать страницы формы и уже на пятой забудет, что было в начале.

Мегамодули появляются и при пакетной обработке – они строят пакетные задания и отправляют их на обработку. Тестировать такие модули очень сложно. Возможно, что более простым решением окажется построение управляющей структуры для построения пакета.

Мегамодули в отчетах выглядят жутковато. Это происходит, когда проектировщики пытаются в одном отчете соединить несколько похожих отчетов. Пользователю совершенно все равно, каким количеством модулей обслуживается его отчет: чем меньше лишней информации он будет содержать, тем лучше.

В оперативных приложениях главная причина появления мегамодуля состоит в том, что аналитики указывают множество ненужных ограничений данных – например требования наличия в базе информации о покупателе только в том случае, если от его имени выполнен хотя бы один заказ товара. При обработке заказа это требование порождает проверку: «новый это покупатель или уже зарегистрированный в системе, и если новый, то когда его регистрировать». Можно разрешить покупателям «существовать» независимо от заказов, а если вам хочется, чтобы покупатели непременно были жестко связаны с заказами, - введите супертип «потенциальный клиент», и если такой клиент делает хотя бы один заказ, то он становится покупателем. Таким образом, вызовы двух экранных форм станут уже независимыми от реализации транзакций в базе данных.

Макеты

При проектировании всегда возникает вопрос, стоит ли тра-

тить драгоценное время на создание макетов. Это элементы (большие или маленькие) реализации реальных задач; они служат для демонстрации потенциальных функциональных возможностей, для изучения мнения пользователей. Макет – это средства представления идей в визуальной форме, поскольку, как говорят, лучше один раз увидеть.

Сколько функций выбрать для макетирования и каких, зависит от количества времени на выполнение этих работ и количества людей, которых можно к этому привлечь. Макет – это своего рода витрина для пользователей. Это означает, что большинство функций реализуют правила интерфейса. Успешная демонстрация может обеспечить заключение контракта на разработку, но есть и другая сторона вопроса: проследите, чтобы пользователь не принял внешнюю оболочку за готовую программу и не надеялся получить готовую систему через пару недель.

Другое назначение макетов – проверить проектные решения. Для этого годятся выявленные на этапе анализа критические участки системы. Хорошими вариантами будут несколько сложных отчетов; часть OLTP, часть OLAP. Это позволит привлечь к процессу проектирования группы тестеров, для того чтобы они проверили производительность системы.

Описание

Для того чтобы разработчики могли правильно сгенерировать модуль, его техническая спецификация должна быть четкой и достаточно подробной.

Описание экранных форм и отчетов должно содержать:

- описание назначения формы (что делает);
- данные навигации (откуда вызвана, что может вызвать сама);
- формат вызова формы;
- список входных параметров и параметров по умолчанию;
- список выходных параметров и правила их обработки;
- описание обработки (события внутри модуля и их обработка);
- список ошибок, которые генерируются в процессе обработки формы и реакция на них;
- ограничения доступа к форме (каковы привилегии, разрешающие действия над формой и ее элементами, каковы привилегии, запрещающие эти действия).

Описание пакетных процессов должно содержать:

- описание функции, выполняемой пакетом;
- данные навигации (откуда вызван пакет, что может вы-

звать сам);
- формат вызова пакета;
- список входных параметров и параметров по умолчанию;
- список выходных параметров и правила их обработки;
- описание обработки (события внутри пакета и их обработка);
- список ошибок, которые генерируются в процессе обработки пакета и реакция на них;
- восстановление (обработка возникновения исключения в середине обработки пакета и реакция на него);
- вероятные блокировки (потенциальные конфликты и обработка ожидания);
- несовместимость с другими пакетными заданиями, отчетами и т.п. (эта информация для планировщика задач);
- регистрацию и аудит;
- ожидаемое состояние базы данных после выполнения пакета и проверку целостности данных;
- ограничения доступа (каковы привилегии, разрешающие вызов пакета, каковы привилегии, запрещающие этот вызов).

Обработка ошибок

Обработка ошибок – это одна из подсистем, которая часто портит жизнь проектировщикам. Пользователи требуют вразумительных сообщений об ошибках. Им не понравится, если при попытке удалить поставщика информационная система выдаст сообщение вида «SQL0532N a parent row cannot be deleted because the relationship CLIENT_ restricts deletion» вместо того, чтобы сообщить о невозможности удаления поставщика, если имеются факты поставки им товара на склад.

Рекомендуется сделать несколько уровней обработки ошибок. Первый уровень доступен в слое модулей, отвечающих непосредственно за вызовы SQL. Этот слой обрабатывает коды ошибок, передаваемые интерфейсом вызовов СУБД. Известно, что коды ошибок, детектирующие одно и то же нарушение ограничений данных, в разных СУБД разные. Поэтому, если ваше предложение имеет хотя бы вероятность работы с несколькими СУБД, вам придется интерпретировать коды возврата СУБД и построить матрицу соответствия кода возврата СУБД и кода ошибки, используемого в подсистеме обработки ошибок. Никакие модули, кроме интерпретатора кодов, не должны иметь доступа к кодам возврата СУБД. Вся обработка ошибок в информационной системе должна строиться на внутренних кодах возврата. Множество этих

кодов может расширяться, но значения кодов изменять нельзя. Требуется проанализировать, в каком формате СУБД возвращает код. Это может быть:

- код ошибки, код ошибки операционной системы;
- sqlca.sqlcode, sqlca.sqlstate;
- sqlca.sqlcode, sqlca.sqlstate, код ошибки операционной системы.

Внутренняя ошибка может состоять из трех компонентов:

- sqlcode – собственно код ошибки;
- sqlstate – уточняющий код состояния и классификации ошибки;
- oscode – сопровождающий код ошибки операционной системы.

Первые два компонента обеспечивают поддержку стандарта, последний позволяет использовать эту же подсистему обработки ошибок для работы всех подсистем. Следует отметить, что некоторые СУБД возвращают код ошибки операционной системы в случае сбоев создания файлов, чтения страниц и т.п. Таким образом, подобная структура может обеспечить достаточную универсальность обработки ошибок. Тип ошибки – ошибка СУБД или ошибка определенного компонента информационной системы – будет контролироваться с помощью кода sqlstate. Как правило, sqlcode – это interger, sqlstate – это char(8) (длина выравнена на 4, что следует делать, например, для RS/6000, SUN SPARC, ALPHA, SGI), oscode – это integer. Например, код -532 интерпретируется в (-906, ’23001’, 0).

Второй слой обработки ошибок – это контекстная интерпретация внутреннего кода возврата. С каждым интерфейсным модулем может быть сопоставлен некоторый набор параметров интерпретации кода возврата. Например, для модуля редактирования списка поставщиков код (-906, ’23001’, 0) соответствует строке сообщения «нельзя удалить поставщика, поскольку есть связанные с ним поставки товара». По контексту этого исключения можно по требованию пользователя показать список поставок данного поставщика.

Реализация

Трудно давать советы по реализации кода модулей, так как каждый разработчик имеет какие-то привычки и свой стиль разработки кода. При реализации проекта важно координировать группу (группы) разработчиков. Все разработчики должны подчиняться жестким правилам контроля исходных тестов. Группа разработчиков, получив технический проект, начинает писать код

модулей, и в этом случае основная задача состоит в том, чтобы уяснить спецификацию. Проектировщик указал, что необходимо сделать, а разработчик определяет способы выполнения.

На этапе разработки осуществляется тесное взаимодействие проектировщиков, разработчиков и групп тестеров. В случае интенсивной разработки тестер буквально «пристегивается» к разработчику, фактически являясь членом группы разработки.

Проектировщик на данном этапе выполняет функции «ходячего справочника», поскольку постоянно отвечает на вопросы разработчиков, касающиеся технической спецификации.

Чаще всего на этапе разработки меняются интерфейсы пользователя. Это обусловлено в том числе и тем, что модули периодически демонстрируются заказчику. Существенно могут меняться и запросы к данным.

Следует отметить, что для сборки всего проекта должно быть выделенное рабочее место. Именно эти модули передаются на тестирование. Взаимодействие тестера и разработчика без централизованной передачи частей проекта допустимо, но только в случае, если необходимо срочно проверить какую-то правку. Очень часто этап разработки и этап тестирования взаимосвязаны и идут параллельно. Синхронизирует действия тестеров и разработчиков система bug tracking.

При разработке должны быть организованы постоянно обновляемые хранилища готовых модулей проекта и библиотек, которые используются при сборке модулей. Желательно, чтобы процесс обновления хранилищ контролировал один человек. Одно из хранилищ должно быть предназначено для модулей, прошедших функциональное тестирование, а другое – для модулей, прошедших тестирование связей. Первое из них – это черновики. Второе – то, из чего уже можно собирать дистрибутив системы и демонстрировать его заказчику для проведения контрольных испытаний или сдачи каких-либо этапов работ.

Документация создается в течение всего процесса разработки. Как только модуль прошел тестирование связей, его можно описывать в документации. В случае если модули изменяются часто, к описанию приступают только тогда, когда модуль становится более или менее стабильным.

Обработка результатов проектирования

На этапе разработки, как правило, еще раз проверяется атомарность функций, а также отсутствие их дублирования.

Желательно, чтобы на этапе проектирования уже была построена матрица «функции-сущности». Это фактически формали-

зованное представление того, что фирма пытается сделать (функции) и какую информацию требуется обработать для достижения результата (сущности). Подобная матрица позволяет проверить следующие моменты:

- имеет ли каждая сущность конструктор – функцию, создающую экземпляры сущности (create);
- есть ли ссылки на данную сущность, то есть используется ли где-либо данная сущность (references);
- имеют ли место изменения данной сущности (update);
- имеет ли каждая сущность деструктор – функцию, которая удаляет экземпляры сущности (delete).

Часто роль деструктора выполняет комплект программ архивирования данных. Нередко в информационных системах информацию просто накапливают. Это допустимо лишь в том случае, если в течение всего периода накопления информации (а фактически в течение всей жизнедеятельности информационной системы) характеристики ее производительности удовлетворяют требованиям заказчика. На практике это чрезвычайно редкое стечение обстоятельств. Связано это в основном с ростом обрабатываемых объемов информации. Следует отметить, что надеяться в этом случае только на мощность СУБД или аппаратного обеспечения нельзя, так как подобные экстенсивные методы повышения производительности дают низкий расчетный прирост скорости. Фактически задача реагирования системы или отдельных ее частей на рост объема обрабатываемых данных является наиболее вероятной задачей тестирования. В таком случае группа тестирования создает модуль генерации (пусть даже абстрактных) данных, выбирается набор запросов, для которых скоростные характеристики критичны, далее производятся замеры и строится зависимость скорости выполнения от объема данных для каждого из запросов. Такое простое действие позволит избежать серьезных ошибок и в проектировании, и в реализации информационной системы.

Спецификация модулей должна быть выполнена еще на этапе проектирования, чему в реальных проектах зачастую просто не придают значения. И напрасно – ведь из-за непродуманной реализации модулей любые достоинства схемы базы данных могут быть утрачены. Так, пренебрегая спецификациями модулей, вы рискуете заложить в информационную систему:

- неконтролируемый рост объемов данных;
- потоки запросов с изначально высокой вероятностью конфликта или потоки запросов, которые будут выпол-

няться «вечно» (попытка выполнить поток, обнаружение конфликта и откат всех действий, новая попытка и т.д.) из-за конфликтующих с ними потоков;

- смешивание системных и интерфейсных модулей;
- дублирование модулей;
- ошибки в размещении бизнес-логики;
- отсутствие реализации или неполная реализация требуемых заказчиком функций системы.

Это далеко не полный список проблем, которые будут обнаружены или на этапе комплексного тестирования, или при вводе системы в эксплуатацию, а может быть, даже в процессе эксплуатации системы (когда начнут реально использоваться модули).

Кроме того, отсутствие спецификаций модулей не позволит точно оценить сложность каждого модуля и, как следствие, определить последовательность создания модулей и правильно распределить нагрузку персонала. Обычная ситуация в подобном случае - «кто-то кого-то ждет», при этом процесс создания информационной системы стоит на месте.

Системные модули

Часто приходится рассматривать большое количество обслуживающих или вспомогательных процессов, которые непосредственно не связаны со сформулированной бизнес-функцией. Как правило, это системные функции, имеющиеся в любой информационной системе, такие как:

- диспетчер очередей или планировщик заданий;
- диспетчер печати;
- средства доступа к данным и создания нерегламентированных запросов (часто это генераторы отчетов);
- управление каталогами и иными ресурсами файловой системы;
- автоматическое резервное копирование;
- автоматическое восстановление после сбоя системы;
- средства регламентирования доступа пользователей к системе (состоящие из средства создания пользователей и средства назначения им привилегий);
- средство настройки среды для пользователя информационной системы;
- средство изменения пользователем своих настроек (в том числе и пароля);
- средство управления приложениями;
- среда администратора информационной системы.

Часть этих функций должна выполнять операционная си-

стема, но если она будет работать в неоднородной среде, то нет гарантии, что пользователям придется по вкусу наличие различных интерфейсов в разных операционных системах. В идеале все приложения-клиенты должны работать в одной операционной системе, однако на практике разработчикам часто приходится сталкиваться с целым «зоопарком» различных рабочих станций у заказчика – итогом нескольких попыток автоматизировать бизнес. Цель разработчика – довести систему до максимально однородного состояния либо сделать похожими хотя бы рабочие места конечных пользователей.

Задача создания информационной системы в разнородной среде существенно повышает требования к разработчикам кода и к выбираемому средству разработки. Особенно это касается разработки системных модулей. Следует уделить внимание модулям, реализация кода которых зависит от операционной системы. Подобные модули должны быть выделены отдельно для каждой из операционных систем в группы, например Win98, WinNT и т.д. Модули каждой из групп должны иметь строгие интерфейсы обмена – данные, которые они передают и получают, строго определены, любое отклонение от спецификации наказуемо. Ни один из модулей вне этой группы не может использовать никаких других вызовов, кроме интерфейсов обмена. Таким образом модули, зависящие от операционнй системы, изолируются от других модулей.

Вообще говоря, практика изолирования системных модулей посредством строгой регламентации их интерфейсов обмена существенно минимизирует затраты по исправлению ошибок и поддержке системы. Кроме того, это облегчает и тестирование, а именно детектирование ошибок и их отладку. Другая сторона вопроса – требования к коду интерфейса обмена системных модулей резко повышаются. Это то, что отлаживается в первую очередь и должно работать очень четко.

Средства мониторинга информационной системы

Если информационная система велика, то следует рассмотреть задачу ее администрирования с одной рабочей станции. Необходимо заботиться не только о конечном пользователе информационной системы, но и о персонале, который будет ее обслуживать. Особое внимание следует уделить мониторингу критических участков информационной системы, поскольку сбой зачастую проще предотвратить, чем исправлять его последствия. Мониторинг относится к тем задачам, о необходимости решения которых заказчик, как правило, не задумывается и которые обычно

отсутствуют и в аналитическом исследовании, и даже при проектировании. Потребность в средствах мониторинга становится очевидной лишь на этапе ввода системы в эксплуатацию, причем потребность эта тем выше, чем сложнее система и чем больше в ней критических участков.

Разработчикам и проектировщикам следует проводить оценку сложности системы. Если принимается решение о написании комплексного средства администрирования и мониторинга, не предусмотренного техническим заданием, то в этом случае следует менять техническое задание, а не идти на поводу у заказчика. В сложной системе отслеживать критические процессы все равно придется. Внедрять подобные средства в уже готовую систему очень сложно, поскольку исходные данные мониторы часто получают от системных модулей достаточно низкого уровня. Без изменений схемы базы данных здесь тоже вряд ли можно будет обойтись, и нет никакой гарантии, что подобное изменение не ухудшит производительность системы.

Разработка мониторов – это довольно специфический класс задач: с одной стороны, они должны обрабатывать достаточный объем информации, с другой – не должны существенно влиять на работу других компонентов информационной системы. Это заставляет разработчиков с особой тщательностью подходить к проектированию мониторов и очень аккуратно писать код их модулей.

Интерфейсы

Интерфейсы конечного пользователя – это то, что заказчик критикует в наибольшей степени, в силу того что именно эти части информационной системы он может более или менее квалифицированно оценить – обычно только их он и видит. Это означает, что интерфейсы являются наиболее часто изменяемым элементом информационной системы именно на этапе реализации.

Часто изменяемый компонент (компоненты) информационной системы следует изолировать от редко изменяемых компонентов, чтобы одни изменения не влекли за собой другие. Один из приемов подобной изоляции – изоляция запросов к данным от интерфейса следующим образом:

- каждый из запросов кодируется идентификатором или «закрывается» определенной системной функцией;
- разработчик интерфейса не знает о запросе к данным ничего, кроме параметров атрибутов выборки – их типа и, возможно, количества строк в выборке;
- обработка ошибок в запросах данных представляет собой

отдельный модуль;
- обработка ошибок в интерпретации результата запроса также представляет собой отдельный модуль.

При обработке результатов запросов данных следует также особое внимание уделить вопросам соответствия типов включающего языка и СУБД, в том числе вопросам точности числовых типов, так как представление их у разных СУБД существенно различается. Кроме того, обратите внимание на запросы к данным, которые используют функции, зависящие от операционной системы, например функции работы с байтами и словами значения атрибута (например, на Intel и SUN SPARC эти функции будут работать по-разному). Типы данных могут быть приведены или явно в запросе функциями приведения cast и встроенными в СУБД функциями, или в функции прикладной программы. Не для всех СУБД неявное преобразование типов дает один и тот же результат, поэтому если информационая система использует данные из нескольких баз данных под управлением разных СУБД, то неявных преобразований типов лучше избегать.

Следует также установить достаточно жесткие правила для внешнего вида интерфейсов пользователя. Должно создаваться впечатление единого стиля для всех компонентов информационной системы.

Версии базы данных

Первую версию базы данных проекта в большинстве случаев создают достаточно быстро – это реализация полностью нормализованной структуры, которую получают на этапе анализа. Основным назначением этой базы данных является обеспечение макетирования, демонстрационных показов, некоторых экспериментов разработчиков и проектировщиков.

Скрипты создания базы данных и заполнения ее стартовыми данными – это тоже исходный код информационной системы, и на него распространяются правила контроля версий. Следует отметить, что поддерживать версии базы даных на уровне скриптов все же проще, чем на уровне средств выгрузки и загрузки данных, предоставляемых самой СУБД, так как в подавляющем большинстве случаев подобные средства не могут предоставить несколько простых, но необходимых функций:

- проконтролировать, какие объекты данных и данные имеют место в объектах загрузки A и B, и загрузить в базу данных только «разницу» A и B (произвести обновление версии);
- проконтролировать, не конфликтуют ли изменения, име-

ющие место в объектах выгрузки C и D, по сравнению с объектом выгрузки A (произвести слияние версий).

CASE-инструменты имеют средства контроля версий схемы базы данных, некоторые имеют настройки, позволяющие также контролировать и стартовые данные. Это дает возможность использовать указанные средства для обеспечения контроля версий базы данных.

Контроль версий исходного кода триггеров, хранимых процедур надежнее осуществлять путем использования той же системы контроля версий, что принята для хранения исходных текстов самого проекта.

Размещение логики обработки

Один из важных вопросов проектирования является способ размещения бизнес-логики обработки данных: размещать ее (и какую часть) либо на сервере в виде хранимых процедур, пакетов, триггеров, иных ограничений целостности непосредственно на сервере баз данных, либо в виде функций на клиенте (в составе ПО клиента). Местонахождение правил интерфейса и правил данных задано точно: первые всегда размещены на клиенте, вторые – на сервере. Правила бизнес-логики в современных СУБД могут быть размещены как на клиенте, так и на сервере. Рассмотрим один из примеров простейшего бизнес-правила:

- Значение в поле экранной формы вводится пользователем, а не выбирается из списка, но набор допустимых значений строго ограничен (например, два или три различных значения).

С одной стороны, пользователь требует немедленной реакции системы на ошибку ввода данных, с другой – недопустимы значения в поле базы данных, отличные от заданных (двух или трех). На самом деле в этой ситуации должны быть реализованы два правила. Правило данных в этом случае будет организовано в виде ограничения check, а правило интерфейса, запрещающее вводить значения, отличные от заданных, будет в точности повторять правило даных, но будет реализовано на уровне интерфейса пользователя. Казалось бы, реализация формы со списком в этом случае является идеальным решением, но большинство операторов предпочитают именно набор в форме, особенно если длина вводимого значения невелика. Формы с большим количеством списков достаточно трудны для обработки конечными пользователями. В случае набора значений в форме следует также позаботиться о приведении регистров строк символов (там, где регистр не существенен) к верхнему или нижнему регистру, на

уровне интерфейса прикладной программы.

Шаблоны

Использование шаблонов и библиотек для построения «похожих» модулей – достаточно распространенная практика. Что использовать в этом случае – объекты и классы или библиотеки – решает конкретная группа разработчиков. Диктовать способ разработки в большинстве случаев бессмысленно, потому что разработчик пишет код так, как умеет или как привык. Эти моменты обычно контролирует руководитель проекта.

В любом проекте запрещается копирование кода, поскольку это ведет к возникновению различных версий одного и того же кода в разных фрагментах прикладной программы и, как следствие, к сложно детектируемым и исправляемым ошибкам. Следует установить жесткое правило: используется вызов функции, а не его копия в коде; любое отклонение от данного правила наказуемо.

Тестирование

Как было сказано выше, группы тестирования могут привлекаться уже на ранних стадиях разработки проекта. Собственно комплексное тестирование действительно следует выделять в отдельный этап разработки. В зависимости от сложности проекта тестирование и исправление ошибок могут занимать треть, половину и больше времени разработки всего проекта.

Чем сложнее проект, тем больше будет потребность в автоматизации системы хранения ошибок – bug tracking. Подобная система обеспечивает следующие функции:

- хранение сообщения об ошибке (с обязательной информацией о том, к какому компоненту системы относится ошибка, кто ее нашел, как ее воспроизвести, кто отвечает за ее исправление и когда она должна быть исправлена);
- система уведомления о появлении новых ошибок, об изменении статуса известных в системе ошибок (как правило, это уведомления по электронной почте);
- отчеты об актуальных ошибках по компонентам системы, по интервалам времени, по группам разработчиков и разработчикам;
- информация об истории ошибки (в том числе отслеживание похожих ошибок, отслеживание повторного возникновения ошибки);
- правила доступа к ошибкам тех или иных категорий;
- интерфейс ограниченного доступа к системе bug tracking для конечного пользователя информационной системы,

который используется как интерфейс обмена информацией между пользователем и службой технической поддержки системы.

Подобные системы снимают множество организационных проблем, в частности вопросы автоматического уведомления об ошибках.

Собственно тесты систем можно разделить на несколько категорий:

- автономные тесты модулей – используются уже на этапе разработки компонентов системы и позволяют отслеживать ошибки отдельных компонентов;
- тесты связей компонентов системы – используются и на этапе разработки, и на этапе тестирования и позволяют отслеживать правильность взаимодействия и обмена информацией компонентов системы;
- системный тест – является основным критерием приемки системы. Как правило, это группа тестов, включающая в себя и автономные тесты, и тесты связей и модели. Данный тест должен воспроизводить работу всех компонентов и функций системы, его основная цель – внутренняя приемка системы и оценка ее качества;
- приемо-сдаточный тест – используется при сдаче системы заказчику. Здесь разработчики часто занижают требования к системе по сравнению с системным тестом, и в общем-то понятно, почему это оправданно;
- тесты производительности и нагрузки – входят в системный тест, но достойны отдельного упоминания, так как именно эта группа тестов является основной для оценки надежности системы.

В тесты каждой группы обязательно входят тесты моделирования отказов. Здесь проверяется реакция компонента, группы компонентов, системы в целом на отказы следующего типа:

- отказ отдельного компонента информационной системы;
- отказ группы компонентов информационной системы;
- отказ основных модулей информационной системы;
- отказ операционной системы;
- «жесткий» сбой (отказ питания, жестких дисков).

Эти тесты позволяют оценить качество подсистемы восстановления корректного состояния информационной системы и служат основным источником информации для разработки стратегий предотвращения негативных последствий сбоев при промышленной эксплуатации. Как правило, это тот класс тестов, которым

разработчики пренебрегают, а затем борются с последствиями сбоев на промышленной системе.

Еще одним важным моментом программы тестирования информационных систем является наличие генераторов тестовых данных. Они используются для проведения как тестов функциональности системы, так и тестов надежности системы, а также тестов производительности системы. Задача оценки характеристик зависимости производительности информационной системы от роста объемов обрабатываемой информации не может быть решена без генераторов данных.

Эксплуатация и сопровождение

Опытная эксплуатация перекрывает процесс тестирования. Как правило, система вводится в эксплуатацию не полностью, постепенно.

Ввод в эксплуатацию проходит по крайней мере три фазы:

1. первоначальная загрузка информации;
2. накопление информации;
3. выход на проектную мощность.

Первоначальная загрузка информации инициирует довольно узкий круг ошибок – в основном это проблемы рассогласования данных при загрузке и собственные ошибки загрузчиков, то есть то, что не было отслежено на тестовых данных. Подобные ошибки должны быть исправлены как можно быстрее. Не поленитесь поставить отладочную версию системы (если, конечно, вам позволят развернуть весь комплекс сопровождающего отладку информационной системы ПО на месте). Если отладку «на живых» данных производить невозможно, то придется моделировать ситуацию, причем быстро. Здесь требуются очень квалифицированные тестеры.

В период накопления информации проявится наибольшее количество ошибок, допущенных при создании информационной системы. Как правило, это ошибки, связанные с многопользовательским доступом. Часто на этапе тестирования таким ошибкам не уделяется должного внимания – видимо, из-за сложности моделирования и дороговизны средств автоматизации процесса тестирования информационной системы в условиях многопользовательского доступа . Некоторые ошибки исправить будет довольно сложно, так как они являются ошибками проектирования. Ни один самый хороший проект от них не застрахован. Это значит, что на всякий случай надо резервировать время на локализацию и исправление таких ошибок.

Вторая категория исправлений связана с тем, что пользова-

теля не устраивает интерфейс. Здесь не всегда нужно выполнять абсолютно все пожелания пользователя, иначе процесс ввода в эксплуатацию не кончится никогда.

В период накопления информации можно столкнуться со знаменитым «упала база». При самом плохом раскладе окажется, что СУБД не выдерживает потока информации. При хорошем – просто параметры конфигурации неверны. Первый случай опасен, так как повлиять на производителя СУБД довольно сложно, а заказчик очень не любит ссылок на службу технической поддержки СУБД. Решать проблему отказа СУБД придется не производителю, а вам – менять схему, снижать поток запросов, менять сами запросы; в общем – вариантов много. Хорошо, если время восстановления базы вписывается в запланированное.

Выход системы на проектную мощность при удачном стечении обстоятельств – это исправление ряда мелких ошибок, и изредка – ошибок серьезных.

Другие подходы к разработке приложений

Как правило, конечные пользователи и руководство полагают, что процесс проектирования не дал никаких результатов, поскольку отсутствуют готовые компоненты, которые можно было бы «пощупать». Зачастую заказчик настаивает на досрочном проведении этапа реализации проекта, для того чтобы как можно быстрее получить какой-то результат и продемонстрировать его. В таком случае существует большой соблазн выбрать ускоренную разработку приложений (УРП) или совместную разработку приложений (СРП). Подобные методы предусматривают разработку рабочего прототипа с последующей демонстрацией его пользователям. Пользователи отмечают, что им нравится, а что – нет. Проектировщик дорабатывает прототип с учетом сделанных замечаний, после чего снова демонстрирует то, что получилось. И так далее. Процесс повторяется до тех пор, пока пользователям не понравится то, что они видят, а прототип не станет рабочим приложением. Обычно устанавливается лимит времени и количество итераций, иначе пользователи будут совершенствовать прототип вечно. Теоретически это позволяет получить ту систему, которая требуется пользователям. На практике подобный подход к разработке приложений сопряжен с серьезными проблемами.

- Все внимание сконцентрировано на экранных формах, а то, что касается правил обработки данных и системных функций, остается за кадром. Есть соблазн начать работу с отчетов, в то время как отчет является не стартовым, а производным продуктом информационной системы.

- Пользователи полагают, что если вариант прототипа согласован, то модуль готов. На самом деле это может быть всего лишь картинка с набором «заглушек» для вызовов системных функций и взаимодействия с другими модулями.
- Модули проектируются изолированно друг от друга (наверное, большинство из вас сталкивались с бухгалтерскими программами, где каждый АРМ является автономным и функции часто дублируются). Следствием этого являются противоречия модулей, дублирование функций и данных, что может быть выявлено только при тестировании комплекса модулей.
- Функциональные возможности наращиваются параллельно в нескольких направлениях, значит, структура базы данных должна контролироваться жестко. При УРП схема базы данных превращается в свалку, где таблицы «лепятся» наскоро, в результате чего имеет место набор противоречивых и дублирующихся данных.
- Документация при использовании метода УРП, как правило, отсутствует, а вернее, о необходимости документировать систему забывают, поскольку создается иллюзия, что пользователь и без того понимает, что происходит. Когда же приложение начинает работать не так, как предполагает пользователь, возникает масса проблем.
- Обработка исключительных ситуаций для каждого модуля производится своя.
- Целостная система, как правило, не получается, скорее всего, это будет некий набор автоматизированных рабочих мест, наскоро связанных между собой.

Методы УРП и СРП можно использовать далеко не всегда, а лишь в том случае, если:

- объем проекта и требования бизнеса четко определены, не изменяются, а сам проект невелик;
- проект не зависит от других средств автоматизации бизнеса, количество внешних интерфейсов, с которыми придется иметь дело, ограниченно;
- система ориентирована на экранные формы, обработка данных и системные функции составляют незначительную часть, удобство экранных форм входит в пятерку важнейших факторов успеха проекта;
- пользователи имеют высокую квалификацию и априори положительно оценивают идею создания нового ПО.

Тем не менее методом УРП лучше разрабатывать небольшие и, желательно, автономные части проекта.

В настоящее время предпринята попытка представить еще один способ быстрого написания проекта – метод экстремального программирования. Ниже будут рассмотрены принципы данного подхода.

Этап планирования (planning game). На основании оценок, сделанных программистами, заказчик определяет функциональные возможности и срок реализации версий системы. Программисты реализуют только те функции, которые необходимы для возможностей, выбранных на данной итерации.

В результате такого решения «за кадром» остается развитие системы, вследствие чего при разработке возникает необходимость строить «заглушки» и переписывать код. Непонятно, почему срок реализации определяет заказчик, ведь на самом деле это прямая обязанность группы проектировщиков. Заказчик, вообще говоря, может лишь выразить свои пожелания по поводу сроков («хочу, чтобы к такому-то числу»), но определить срок может только проектировщик («выполнимо не меньше чем за такое-то время»).

Частая смена версий (small releases). Систему запускают в эксплуатацию уже через несколько месяцев после начала реализации, не дожидаясь окончательного разрешения всех поставленных проблем. Выпуск новых версий может происходить с периодичностью от ежедневного до ежемесячного.

Все хорошо, кроме одного: протестировать за такой срок более или менее сложный компонент невозможно. Заказчик фактически выступает в роли бета-тестера. В этом случае он может видеть, что разработчики трудятся и даже ошибки исправляют. Однако возникают резонные вопросы: стоит ли посвящать заказчика в рабочий процесс и нужно ли ставить эксперименты на рабочей системе? В дополнение к сказанному необходимо отметить, что подобный принцип вряд ли может быть реализован для частей проекта, которые требуют работы в режиме 24x7.

Метафора (metaphor). Общий вид системы определяется при помощи метафоры или набора метафор, над которыми совместно работают заказчик и программисты.

С одной стороны, этот постулат кажется неплохим, а с другой – имеет ли смысл посвящать заказчика во внутренние дела группы разработчиков? То, что касается общего вида (интерфейсы, отчеты и т.п.), действительно может находиться в компетенции заказчика, но когда речь идет об особенностях реализации

тех или иных компонентов, заказчик вряд ли может быть полезен из-за отсутствия у него необходимых знаний.

Простой проект (simple design). В каждый момент времени разрабатываемая система выполняет все тесты и поддерживает все взаимосвязи, определяемые программистом, не имеет дубликатов кода и содержит минимально возможное количество классов и методов. Это правило кратко можно выразить так: «Каждую мысль формулируй один и только один раз».

Эта мысль тоже хороша, но она не вполне согласуется с принципом быстрого написания кода. Может быть, стоит все-таки сначала подумать, как делать тот или иной модуль, группу модулей, и лишь потом заняться написанием кода?

Тесты (tests). Программисты постоянно пишут тесты для модулей (unit tests). Собранные вместе, эти тесты должны работать корректно. Для этапов в итерации заказчики пишут функциональные тесты (functional tests), которые также должны работать правильно. Однако на практике это не всегда достижимо. Чтобы принять правильное решение, необходимо понять, во сколько обойдется сдача системы с заранее известным дефектом, и сравнить это с ценой задержки на исправление дефекта.

При написании тестов самими программистами (особенно в условиях сверхурочных работ) эти тесты не полнофункциональны, и уж тем более не учитывают особенностей многопользовательской работы. На более продвинутые тесты у разработчиков обычно не хватает времени. Можно, конечно, построить систему разработки так, что всем будут заниматься одни и те же люди, но все-таки не стоит превращать проект в аналог телепередачи «Сам себе режиссер». К сказанному необходимо добавить, что тестирование системы вовсе не исчерпывается тестами компонентов (units); не менее важны тесты взаимодействия между ними, это же относится и к тестам надежности работы. И тем не менее метод экстремального программирования не предусматривает создания тестов данного класса. Это объясняется тем, что сами такие тесты могут представлять достаточно сложный код (особенно это касается тестов-имитаторов реальной работы системы). В данной технологии также никак не учитывается еще один важный класс тестов – тесты поведения системы при росте объемов обрабатываемой информации. При высокой скорости изменения версий выполнить такой тест технологически невозможно, поскольку его проведение требует стабильного и неизменного кода проекта, например, в течение недели. Подобные сроки, вообще говоря, не гарантируются из-за частой смены версий. В таком случае при-

дется или приостанавливать разработку компонентов, или на время проведения теста создавать параллельную версию проекта, которая будет сохраняться неизменной, тогда как вторая при этом будет изменяться. Потом нужно будет выполнять процесс слияния кода. Но в этом случае тест придется создавать снова, так как методы экстремального программирования просто не предусматривают разработку средств, позволяющих прогнозировать поведение системы при тех или иных изменениях.

Переработка системы (refactoring). Архитектура системы постоянно эволюционирует. Текущий проект трансформируется, при этом гарантируется правильное выполнение всех тестов.

Вот тут-то и начинается самое интересное. Экстремальное программирование исходит из того, что переделать всегда можно, причем без особых затрат. Однако практика свидетельствует об обратном.

Программирование в паре (pair programming). Весь код проекта пишут два человека, которые используют одну настольную систему.

Возникает вопрос: кто-нибудь видел двух совершенно одинаковых программистов, каждый из которых к тому же в конце рабочего дня успевал бы писать документацию для напарника? Можно ли найти таких программистов-близнецов, согласных во всем?

А главное, зачем нужна такая пара программистов? Причина, в общем-то, простая: не все выдерживают навязываемый при экстремальном программировании высокий темп работ, неизбежен отток персонала. Подобная пара может дать некую страховку – если уволится один, то, может быть, второй доведет дело до конца. Правда, оставшийся попадет в еще более жесткие временные рамки – ведь объем работ останется прежним же, а дублера уже не будет, по крайней мере какое-то время. Далее следует естественный процесс передачи информации новому дублеру, что опять-таки требует времени. И так без конца.

Непрерывная интеграция (continuous integration). Новый код интегрируется в существующую систему не позднее, чем через несколько часов. После этого система вновь собирается в единое целое и прогоняются все тесты. Если хотя бы один из них не выполняется корректно, внесенные изменения отменяются.

Этот постулат предоставляется по меньшей мере спорным, поскольку непонятно, кто будет исправлять ошибки, причем не только локальные, но и наведенные неправильным кодом. Ведь проведение комплексных тестов не предполагается на данном

этапе, кроме того, изменения остаются даже в том случае, когда ошибка детектирована. В то же самое время метод экстремального программирования не предусматривает наличия системы отслеживания ошибок.

Коллективное владение (collective ownership). Каждый программист имеет возможность в любое время усовершенствовать любую часть кода в системе, если сочтет это необходимым.

Вам это анархию не напоминает? Как в этом случае искать автора изменений? Встречал ли кто-либо при разработке большого проекта такого «на все руки доку» и сколько подобный «умелец» сумел бы продержаться на своем рабочем месте? Правильно, не слишком долго.

Заказчик с постоянным участием (on-site customer). Заказчик, который в период работы над системой находится в команде разработчиков.

Это, конечно, хорошо, но непонятна цель: то ли посвятить заказчика в суть дела, то ли сделать его соавтором? Вряд ли только у заказчика найдется столь высококвалифицированный специалист.

40-часовая неделя (40-hour weeks). Объем сверхурочных работ не может превышать по длительности одну рабочую неделю. Даже отдельные случаи сверхурочных работ, повторяющиеся слишком часто, являются сигналом серьезных проблем, которые требуют безотлагательного решения.

Как показывает практика применения экстремального программирования (несмотря на целый ряд положительных примеров, приводимых сторонниками данного метода), сверхурочные при таком подходе – это правило, а не исключение, и борьба с проблемами в данном случае – явление постоянное. Усиливается она в период замены текущей сырой версии продукта очередной, опять же сырой, версией. Заказчик, посвященный в процесс, испытывает все прелести проявления ошибок работы системы на себе. Как вы думаете, надолго ли хватит у заказчика терпения при таком положении дел? Ему ведь надо, чтобы система работала…

Открытое рабочее пространство (open workspace). Команда разработчиков располагается в большом помещении, окруженном комнатами меньшей площади. В центре рабочего пространства устанавливаются компьютеры, на которых работают пары программистов.

Причем все это, судя по предыдущим принципам, должно располагаться на территории заказчика, раз он весьма активно

привлекается к процессу разработки. Возникает вопрос: реально ли столь удачное стечение обстоятельств?

Не более чем правила (just rules). Члены коллектива, работающего по технологии экстремального программирования, обязуются выполнять изложенные правила. Однако это не более чем правила и команда может в любой момент поменять их, если ее члены достигли принципиального соглашения по поводу внесенных изменений.

Может быть, в конце концов и будет выработано одно полезное правило: «сначала подумай, потом сделай». В этом случае мы будем иметь схему, весьма похожую на «водопад». Почему-то сторонники экстремального программирования убеждены, что при использовании «водопада» и его клонов цикл разработки обязательно должен быть длинным. Непонятно, чем обусловлена такая уверенность. Ведь не запрещено дробить проект на этапы. Почему-то считается, что планирование обязательно будет одноразовым и неизменным, хотя на самом деле это не соответствует истине, в том числе и в случае «водопада».

В итоге мы получаем метод, потенциально обладающий высокой адаптируемостью к сильно изменяющимся требованиям к проекту, но в то же время не свободный от ряда серьезных недостатков. Последнее обстоятельство не позволяет рекомендовать данный метод к применению для проектов, требующих высокой или как минимум достаточной надежности работы.

5.2. Задание к лабораторной работе.

На основании исходных данных (ЛР 1,2,3) закончить разработку программного продукта.

К отчёту представить:

- документ в любом электронном формате содержащий разработанный программный продукт.

Отчёт о проделанной работе:

- собеседование по представленному документу.

5.3. Контрольные вопросы

1. Понятие жизненного цикла программного обеспечения.
2. Каскадная модель.
3. Поэтапная модель с промежуточным контролем.
4. Спиральная модель.
5. «Водопад» - схема разработки проекта.
6. Определение стратегии проекта.

7. Выбор средств разработки.
8. Отображение функций на модули.
9. Интерфейсы программ.
10. Интегрирование и наследование механизмов обмена данными.
11. Определение спецификаций модулей.