|  |  |  |
| --- | --- | --- |
| УЧЕБНОЕ ПОСОБИЕ | Ульяновский авиационный колледж | |
| Профессиональный цикл | |
| **ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**      **КУРС ЛЕКЦИЙ**    **для специальностей СПО базовой подготовки**  **09.02.03 Программирование в компьютерных системах**        Ульяновск  2016 | |
| ОДОБРЕНО  на заседании ЦМК  Программирования и ИТ  Протокол № \_\_\_\_  от «\_\_\_»\_\_\_\_\_\_\_\_\_20\_\_\_ г.    Председатель ЦМК:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А.Шарифуллина | | УТВЕРЖДАЮ  Зам. директора по учебно- методической работе:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.Н.Подкладкина  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_ 20 \_\_ г. |
|  | | |

**https://infourok.ru/kurs-lekciy-po-mdk-instrumentalnie-sredstva-razrabotki-po-dlya-specialnosti-programmirovanie-v-kompyuternih-sistemah-1412687.html**

**ЧАСТЬ 1. ИСТОРИЯ РАЗВИТИЯ ИСРПО**

Программные продукты можно классифицировать по различным признакам.

Рассмотрим классификацию, в которой основополагающим признаком является *сфера (область) использования программного обеспечения*.

Основными сферами использования ПО являются:

* аппаратная часть автономных компьютеров и сетей ЭВМ;
* функциональные задачи различных предметных областей;
* технология разработки программ.

Соответственно в этих областях выделим три класса программных продуктов:

* системное программное обеспечение;
* пакеты прикладных программ;
* инструментарий технологии программирования.

Рис. 1. Классы программных продуктов

Очевидно, что в рамках этой дисциплины нас интересует третье направление.

Развитие этого направления, связанно с переходом на промышленную технологию производства программ, стремлением к сокращению сроков, трудовых и материальных затрат на производство и эксплуатацию программ, обеспечению гарантированного уровня их качества.

В рамках этого направления сформировались следующие **группы программных продуктов:**

* 1. **средства для создания приложений**, включающие:
* локальные средства, обеспечивающие выполнение отдельных работ по созданию программ;
* интегрированные среды разработчиков программ, обеспечивающие выполнение комплекса взаимосвязанных работ по созданию программ;
  1. **СASE-технология (Computer-Aided System Engineering),** представляющая методы анализа, проектирования и создания программных систем и предназначенная дли автоматизации процессов *разработки и реализации* информационных систем.

**Средства для создания приложений** – это совокупность языков и систем программирования, а также различные программные комплексы для отладки и поддержки создаваемых программ.

**Локальные средства разработки программ.** Эти средства широко представлены на рынке программных продуктов и включают в себя:

* языки программирования
* системы программирования
* инструментальную среду пользователя.

**Язык программирования** - формализованный язык для описания алгоритма решения задачи на компьютере.

**Системы программирования***(programming system)*включают в себя:

- компилятор;

- интегрированную среду разработчика программ;

- отладчик;

- средства оптимизации кода программ;

- набор библиотек (возможно с исходными текстами программ);

- редактор связей;

- сервисные средства (утилиты) для работы с библиотеками, текстовыми и двоичными файлами;

- справочные системы;

- документатор исходного кода программы;

- систему поддержки и управления проектом программного комплекса.

**Средства поддержки проектов** – это новый класс программного обеспечения, предназначен для:

- отслеживания изменений, выполненных разработчиками программ;

- поддержки версий программы с автоматической разноской изменений;

- получения статистики о ходе работ проекта.

**Инструментальная среда пользователя** представляет собой **специальные средства**, встроенные в пакеты прикладных программ, такие, как:

- библиотека функций, процедур, объектов и методов обработки;

- макрокоманды;

- клавишные макросы;

- языковые макросы;

- программные модули-вставки;

- конструкторы экранных форм и отчетов;

- генераторы приложений;

- языки запросов высокого уровня;

- языки манипулирования данными;

- конструкторы меню и многое другое.

**Интегрированные среды разработки программ.** Дальнейшим развитием *локальных* средств разработки программ, которые объединяют набор средств для комплексного применения на всех технологических этапах создания программ, являются интегрированные программные среды разработчиков. Основное назначение инструментария данного вида - повышение производительности труда программистов, автоматизация создания кодов программ, обеспечивающих интерфейс пользователя графического типа, разработка приложений для архитектуры клиент-сервер, запросов и отчетов.

**Коротко**: это набор средств разработки программ для

* автоматизации создания кодов;
* повышения производительности труда программистов при создании графического интерфейса;
* повышения производительности труда программистов при разработке клиент-серверных приложений, запросов и отчетов.

**CASE-технология создания информационных систем**

Средства CASE-технологии - относительно новое, сформировавшееся на рубеже 80-х годов направление.

CASE-технология – это программный комплекс, автоматизирующий весь технологический процесс

* анализа,
* проектирования,
* разработки
* сопровождения сложных программных систем.

В истории развития ИСРПО обычно выделяют шесть периодов. Периоды различаются

* применяемой техникой
* методами разработки ПС.

В эти периоды, *в качестве инструментов разработки*, использовались следующие ПС:

**Период 1.** Ассемблеры, анализаторы.

**Период 2.** Компиляторы, интерпретаторы, трассировщики.

**Период 3**. Символические отладчики, пакеты программ.

**Период 4**. Системы анализа и управления исходными текстами.

**Период 5**. Первое поколение CASE.

Это CASE – средства, позволяющие выполнять *поддержку начальных работ процесса разработки ПС и систем*. Адресованы непосредственно системным аналитикам, проектировщикам, специалистам в предметной области. Поддерживают графические модели, экранные редакторы, словари данных. Не предназначены для поддержки полного ЖЦ ПС.

**Период 6.** Второе поколение CASE. Представляют собой набор инструментальных средств, каждое из которых предназначено для поддержки отдельных этапов процесса разработки или других процессов ЖЦ. В совокупности обычно поддерживают практически полный ЖЦ. Используют средства моделирования предметной области, графического представления требований, поддержки автоматической кодогенерации. Содержат средства контроля и управления разработкой, интеграции системной информации, оценки качества результатов разработки. Поддерживают моделирование и прототипирование системы, тестирование, верификацию, анализ сгенерированных программ, генерацию документации по проекту.

Наиболее востребованы CASE – средства на первых этапах ЖЦ, связанных с анализом требований и проектированием. CASE – средства позволяют использовать визуальные среды разработки, средства моделирования и быстрого прототипирования разрабатываемой системы. Это позволяет как можно раньше оценить, насколько будущая система устраивает заказчика и насколько она дружественна будущему пользователю.

Сравним усредненные оценки трудозатрат по основным этапам разработки ПС при различных подходах к процессу разработки. Номерам строк будут соответствовать: 1 – традиционная разработка с использованием классических технологий; 2 – разработка с использованием современных структурных методологий проектирования; 3 – разработка с использование CASE – средств.

Из таблицы видно, что при традиционной разработке ПС основные усилия направлены на кодирование и тестирование, а при использовании CASE – технологий – на анализ и проектирование, поскольку CASE предполагают автоматическую кодогенерацию, автоматизированное тестирование и автоматический контроль проекта. Сопровождение кодов ПС заменяется сопровождением спецификаций проектирования. В результате данных факторов цена ошибок, существовавших в проекте при разработке и сопровождении, существенно снижается.

Литература:

1. Бахтизин В.В. Технология разработки программного обеспечения: учеб. пособие / В.В.Бахтизин, Л.А.Глухова. – Минск: БГУИР, 2010. – 267 с. [с.242 - 244]

**ЧАСТЬ 2. БАЗОВЫЕ ПРИНЦИПЫ ПОСТРОЕНИЯ CASE – СРЕДСТВ**

Большинство CASE – средств основано на парадигме

**«метод – нотация – средство»**

**Парадигма** – это система *изменяющихся форм* некоторого понятия.

**Метод** – это систематическая *процедура или техника* генерации описания компонент ПС.

**Нотация** – это система обозначений, предназначенная для описания структуры системы, элементов данных, этапов обработки; может включать графы, диаграммы, таблицы, схемы алгоритмов, формальные и естественные языки.

**Средства** – это инструментарий для поддержки методов, помогающий пользователям при создании и редактировании графического проекта в интерактивном режиме, способствующий организации проекта в виде иерархии уровней абстракции, выполняющий проверки соответствия компонентов.

К CASE – средствам может быть отнесено любое программное средство, обеспечивающее автоматическую помощь при разработке ПС, их сопровождении или управлении проектом, базирующееся на следующих основополагающих принципах:

1. **Графическая ориентация**. В CASE – средствах используется мощная графика для описания и документирования систем и для улучшения интерфейса с пользователем.
2. **Интеграция**. CASE – средство обеспечивает легкость передачи данных между своими компонентами и другими средствами, входящими в состав линейки CASE – средств. Это позволяет поддерживать совокупность процессов ЖЦ.
3. **Локализация всей проектной информации в репозитории**. Исполнителям проекта доступны соответствующие разделы репозитория в соответствии с их уровнем доступа. Это обеспечивает поддержку принципа коллективной работы.

Кроме того, в основе концептуального построения CASE – средств лежат следующие положения:

* Человеческий фактор, его учет позволяет привести процессы ЖЦ к легкой, удобной и экономичной форме.
* Использование базовых программных средств, применяющихся в других приложениях (СУБД, компиляторов различных языков программирования, отладчиков и др.).
* Автоматизированная или автоматическая кодогенерация. При автоматизированной кодогенерации выполняется частичная генерация кодов программного средства, остальные участки программируются в ручную. При автоматической кодогенерации выполняется полная генерация кодов. Возможны различные виды генерации (проектной документации, баз данных, кодов, автоматическая сборка модулей)
* Ограничение сложности. Такое ограничение позволяет поддерживать сложность компонентов разрабатываемого программного средства или системы на уровне, доступном для понимания, использования и модификации.
* Доступность для различных категорий пользователей, в том числе заказчиков, специалистов в предметной области, системных аналитиков, проектировщиков, программистов, тестировщиков, инженеров по качеству, менеджеров проекта. CASE – средства содержат инструменты различного функционального назначения, поддерживающие различные этапы основных, вспомогательных и организационных процессов ЖЦ.
* Рентабельность, обеспечивает быструю окупаемость денежных средств, вложенных в CASE – средства, за счет сокращения сроков и стоимости проектов.
* Сопровождаемость. CASE – средства обладают способностью адаптации к изменяющимся требованиям и целям проекта.

**ЧАСТЬ 3.**  **ОСНОВНЫЕ ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ CASE - СРЕДСТВ**

В состав CASE – средств входят четыре основных компонента.

1. **Репозиторий**. Средства централизованного хранения всей информации о проекте (репозиторий). Он предназначен для хранения информации о разрабатываемом программном средстве в течении всего ЖЦ.
2. **Средства ввода**. Служат для ввода данных в репозиторий, организации взаимодействия участников проекта с CASE – средством. Должны поддерживать различные методологии анализа, проектирования, тестирования, контроля. Предназначены для использования в течении ЖЦ различными категориями участников проекта.
3. **Средства анализа и разработки**. Предназначены для анализа различных видов графических и текстовых описаний и их преобразования в процессе разработки.
4. **Средства вывода**. Служат для кодогенерации, создания различного вида документов, управления проектом.

Все компоненты CASE – средств в совокупности обладают следующими функциональными возможностями:

* Поддержка графических моделей;
* Контроль ошибок;
* Поддержка репозитория;
* Поддержка основных, вспомогательных и организационных процессов ЖЦ.

В CASE – средствах разрабатываемые ПС представляются схематически. На разных уровнях проектирования могут использоваться различные виды и нотации графического представления ПС. Обычно применяются диаграммы различных типов. Разработка диаграмм осуществляется с помощью специальных графических редакторов, основными функциями которых являются создание и редактирование иерархически связанных диаграмм, их объектов и связей между объектами, а также автоматический контроль ошибок.

В CASE - средствах, как правило, реализуются следующие типы контроля.

1. Контроль синтаксиса диаграмм и типов их элементов.
2. Контроль полноты и корректности диаграмм.
3. Контроль декомпозиции функций.
4. Сквозной контроль диаграмм.

*Основными функциями репозитория являются* хранение, обновление, анализ, визуализация всей информации по проекту и организация доступа к ней. Репозиторий хранит более 100 типов объектов.

Каждый информационный объект, хранящийся в репозитории, описывается совокупностью своих свойств, например, идентификатор, тип, текстовое описание, компоненты, область значений, свиязи с другими объектами, времена создания и последнего обновления, автор и т.п.

Репозиторий является базой для автоматической генерации документации по проекту. Основными типами отчетов являются:

* **Отчеты по содержимому** – включают информацию по потокам данных и их компонентов; списки функциональных блоков диаграмм и их входные и выходные потоки; списки всех информационных объектов и их атрибутов; историю изменений объектов; описания модулей и интерфейсов между ними; планы тестирования модулей и т.п.
* **Отчеты по перекрестным ссылкам** – содержат информацию по связям всех вызывающих и вызываемых модулей; спсики объектов репозитория, к которым имеет доступ конкретный исполнитель проекта; информацию по связям между диаграммами и конкретными данными; маршруты движения данных от входа к выходу;
* **Отчеты по результатам анализа** – включают данные по взаимной корректности диаграмм, списки неопределенных информационных объектов, списки неполных диаграмм, данные по результатам анализа струткуры проекта и т.п.;
* **Отчеты по декомпозиции объектов** – включают совокупности объектов, входящих в каждый объект, а также объекты, в состав которых входит каждый объект.

Основой поддержки процесса разработки являются следующие свойства современных CASE – средств.

1. Покрытие всего жизненного цикла ПС.
2. Поддержка прототипирования. Это касается в первую очередь моделей, поддерживающих инкрементную и эволюционную стратегии разработки. Прототипирование применяется на ранних этапах ЖЦ и позволяет уточнять требования к системе, а также прогнозировать поведение разрабатываемого продукта.
3. Поддержка современных методологий разработки ПС. Современные CASE – средства поддерживают, как правило, различные методологии, предназначенные для использования на различных этапах процесса разработки. При этом выполняется графическая поддержка построения диаграмм различных типов, контроль корректности использования шагов проектирования и подготовка документации.
4. Автоматическая кодогенерация.

**ЧАСТЬ 4. КЛАССИФИКАЦИЯ CASE - СРЕДСТВ**

Все CASE - средства подразделяют на типы, категории и уровни.

**Классификация по типам**отражает функциональное назначение CASE – средств в ЖЦ ПС.

1. **Анализ и проектирование**. Средства этого типа используются для поддержки начальных этапов процесса разработки: анализа предметной области, разработки требований к системе, проектирования системной архитектуры, технического проектирования программной архитектуры, технического проектирования программных средств. Средства данного типа поддерживают известные методологии анализа и проектирования. На выходе генерируются спецификации системы, ее компонентов и интерфейсов, архитектура системы, архитектура программного средства, технический проект, включая алгоритмы и структуры данных. Примерами являются: AllFusion Process Modeler (BPwin), CASE.Аналититк, Design/IDEF, Telelogic DOORS, Telelogic Modeler, Telelogic TAU, Telelogic Rhapsody, Telelogic Statemate.
2. **Проектирование баз данных и файлов**. Средства этого типа обеспечивают логическое моделирование данных, автоматическое преобразование моделей данных в третью нормальную форму, автоматическую генерацию схем баз данных и описаний форматов файлов на уровне программного кода. Примерами являются: AllFusion Process Modeler (BPwin), CA Erwin Data Model Validator, S-Designor, Silverrun, Designer2000, Telelogic TAU, Telelogic Rhapsody.
3. **Программирование и тестирование**. Средства этого типа поддерживают программирование и тестирование. Данные средства автоматически выполняют кодогенерацию на основе спецификаций или моделей. Содержат графические редакторы, средства поддержки работы с репозиторием, генераторы и анализаторы кодов, генераторы тестов, анализаторы покрытия тестами, отладчики. Примерамиявляются: TAU/Developer, TUA/Tester, Logiscope Audit, Logiscope RuleChecker, Logiscope TestChecker, Logiscope Reviewer, Rhapsody Developer.
4. **Сопровождение и реинженерия**. Общей целью средсвт этого типа является поддержка корректировки, изменений, преобразований, реинженерия существующей системы, поддержка документации по проекту. К данным средствам относятся средства документирования, анализаторы программ, средства управления изменениями и конфигурацией ПС и систем, средства реструктурирования и реинженении, средства обеспечения мобильности. Примерами являются: Telelogic DocExpress, Telelogic Synegry, Telelogic Change, AllFusion Change Management Suite.
5. **Окружение**. К средствам данного типа относятся средства поддержки интеграции CASE – средств и данных. Примерами являются: Telelogic Rhapsody Gateway, Telelogic Rhapsody Interface Pack, AllFusion Data Profiler, AllFusion Model Manager, AllFusion Model Navigator.
6. **Управление проектом**. К средствам данного типа относятся средства поддержки процесса управления ЖЦ. Их функциями являются планирование, контроль, руководство, организация взаимодействия. Примерами являются: Telelogic Focal Point, Telelogic dashboard, AllFusion ProcessManagement Suite, Advisor.

**Классификация по категориям** отражает уровень интегрированности CASE – средств по выполняемым функциям.

1. **Категория Tool**. Tool – рабочий инструмент. Включает средства самого низкого уровня интегрированности. В данную категорию входят инструментальные средства, решающие небольшую автономную задачу при разработке ПС. Обычно средства данной категории являются компонентами CASE – средств белее высокого уровня интегрированности.
2. **Категория ToolKit**. ToolKit – набор инструментов, пакет разработчика. Включает средства среднего уровня интегрированности. Средства данной категории используют репозиторий для всей информации о проекте и ориентированы обычно на поддержку одного этапа или одной работы процесса разработки или на поддержку одного из вспомогательных или организационных процессов. Средства данной категории представляют собой интегрированную совокупность инструментальных средств, имеющих как правило общую функциональную ориентацию.
3. **Категория Workbench**. Workbench – рабочее место. Средства данной категории обладают самой высокой степенью интеграции. Они представляют собой интегрированную совокупность инструментальных средств, поддерживающих практически весь процесс разработки и ряд вспомогательных и организационных процессов.

**Классификация по уровням** связана с областью действия CASE – средств.

1. **Верхние CASE – средства**. Средства данного уровня называют средствами компьютерного планирования. Их основной целью является помощь руководителям организации, предприятия и конкретных проектов в определении политики организации и создании планов проектов. CASE – средства данного уровня позволяют строить модель предметной области, проводить анализ различных сценариев (существующего, наихудших, наилучших), накапливать информацию для принятия оптимальных решений. Таким образом, применительно к ЖЦ ПС данные средства поддерживают процесс заказа и первую работу процесса разработки (подготовка процесса разработки). Графические средства данного уровня используются как формализованный язык общения между заказчиком и разработчиком требований. Примерами являются: Telelogic System Architect, Telelogic Focal Point, Telelogic Dashboard, AllFusion Modeling Suite.
2. **Средние CASE – средства**. Средства данного уровня поддерживают начальные этапы процесса разработки. Обычно данные средства обладают возможностью накопления и хранения информации по проекту. Это позволяет использовать накопленные данные как в текущем, так и в других проектах. Часто эти средства поддерживают прототипирование и автоматическое документирование.
3. **Нижние CASE – средства**. Средства данного уровня поддерживают вторую половину работ процесса разработки. Содержат графические средства, исключающие необходимость разработки спецификаций для программных модулей. Спецификации представляются моделями и непосредственно преобразуются в программные коды. Автоматически генерируется до 90% кода. Как правило, они поддерживают прототипирование, тестирование, управление конфигурацией, генерацию документации, облегчают модификацию и сопровождение ПС.

Литература:

1. Бахтизин В.В. Технология разработки программного обеспечения: учеб. пособие / В.В.Бахтизин, Л.А.Глухова. – Минск: БГУИР, 2010. – 267 с. [с.24 - 254]

**Часть 5. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА УПРАВЛЕНИЯ ПРОЕКТОМ**

**5.1. ОБЗОР ВОЗМОЖНОСТЕЙ ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВ УПРАВЛЕНИЯ ПРОЕКТОМ**

Управление проектом объединяет множество инструментальных средств, методов и технологий, связанных с планированием проекта и управлением интегрированным процессом.

Управление проектом – нечто вроде распределения и управления бюджетом, временем и людьми. Давно проверенный практикой принцип гласит, что если вы не сможете спланировать что-то, вы не сможете это и сделать. Планирование требует некоторых начальных знаний того, что требуется для разработки системы в этой организации, с этими людьми, с этими ресурсами, в этой организационной культуре и т.п. Организация должна оценить, что происходило с предыдущими проектами, определить из предыдущих проектов метрики (систему показателей).

Современные инструментальные средства управления проектами объединяют обычное управление проектом со стратегическим планированием, бизнес – моделированием, портфельным управлением, управлением документацией, технологическим процессом и т.д. Лучшие представители управления большими проектами используют web- технологии. Которые позволяют осуществлять динамическое, интерактивное и синхронное управление большими коллективами и процессами.

Одним из главных инструментов управления планированием проектом являются сетевые графики. Два традиционных вида *графиков операций* – метод критического пути и диаграмма Ганта.

Инструментальные средства управления проектами могут формировать график операции, как только будет обеспечена необходимая проектная информация. Требуемая проектная информация включает определение операций, их продолжительность, ресурсы, предшествующие события и события которые являются следствием операций.

Инструментальные средства управления проектами регулируют и перестраивают планы всякий раз, когда происходят изменения в операциях или ресурсах. Это позволяет менеджерам понять, может ли дополнительный ресурс дать лучший результат в проекте.

**5.2. УПРАВЛЕНИЕ ПРОЕКТОМ С ПОМОЩЬЮ РОГРАММЫ  MS  PROJECT**

Несмотря на пользовательский интерфейс, схожий с приложениями Microsoft Office, Microsoft Project является специализированной программой, предназначенной исключительно для управления проектами.

Семейство продуктов Microsoft Project состоит из трех приложений MS Project Standard, MS Project Professional, MS Project Server. MS Project Standard и MS Project Server переведены на русский язык.

Стандартная и профессиональная версия программы предназначены для создания плана проекта, который затем можно опубликовать на сервере MS Project Server для организации коллективной работы над проектом.

С помощью сервера MS Project Server члены проектной команды получают от руководителя задачи, сообщают о ходе их выполнения. Руководитель проекта в соответствии с данными, поступающими от сотрудников, отслеживает ход выполнения работ и анализирует загрузку сотрудников.

В качестве СУБД MS Project Server использует MS SQL Server версии 7 и выше.

Для совместной работы над документами и задачами предназначен пакет Microsoft SharePoint Team Services, входящий в состав MS Project Server.

Основные термины

***Задача****-*активность, направленная на достижение определенного результата.

***Ресурс***назначается на задачу и может быть ***материальным***(материалы, оборудование и сырье) и *рабочим*(работники и оборудование, оплачиваемое по часам).

Выделение ресурса на задачу называется ***назначением.***На одну задачу можно назначить неограниченное количество ресурсов.

Задача характеризуется ***длительностью, объемом трудозатрат {работ)*и *стоимостью***необходимыми для ее выполнения.

***Длительность проекта***складывается из промежутков времени от начала самой ранней задачи до окончания самой поздней с учетом зависимостей между задачами.

Планирование проектов

В деятельности любой организации различают программы, операции и проекты. Программа - это деятельность по управлению проектами и операциями внутри организации. Операции - это повторяющиеся действия, направленные на поддержание деловой активности. Проект же предпринимается для достижения определенного результата в условиях установленных сроков и средств.

Управление проектами заключается в составлении плана и отслеживании выполнения работ по нему. Чем лучше составлен план проекта, тем легче выполнять проектные работы и удачно завершить проект.

Проекты могут быть любого масштаба, в них может быть задействовано несколько человек, а может и несколько тысяч. Проекты могут иметь разной длительности от нескольких дней до нескольких лет. Проекты могут осуществляться в любой области деятельности.

Для каждого проекта четко определены начало и конец. Конец проекта наступает с достижением поставленных целей или когда становится ясно, что цели не могут быть достигнуты и проект обрывается.

После окончания проекта проектная команда распадается, и ее участники переходят в другие проекты.

План проекта составляется, чтобы определить, с помощью каких работ будут достигнуты результаты проекта, какие люди и оборудование нужны для выполнения этих работ, и в течение какого времени люди и оборудование будут заняты работой по проекту. План проекта содержит три основных элемента: задачи, ресурсы и назначения.

***Задачей***называется работа, осуществляемая в рамках проекта для достижения определенного результата.

*Фаза*проекта состоит из одной или нескольких задач, в результате выполнения которых достигается один или несколько результатов проекта. Таким образом, результат фазы суммирует результаты входящих в нее задач. Совокупность фаз проекта называется его ***жизненным циклом.***Фазы могут состоять как из задач, так и из других фаз.

В каждом проекте для достижения общей цели необходимо достигнуть нескольких промежуточных целей. Задачи, в результате выполнения которых достигаются промежуточные цели, называются завершающими задачами. В MS Project они называются ***вехами****.*Вехи могут присутствовать в проекте для облегчения отслеживания плана проекта.

Длительность задачи - это период рабочего времени, который необходим для ее выполнения. Длительность может не соответствовать трудозатратам работников, которые выполняют эту задачу.

Задачи в плане проекта взаимосвязаны. Зависимости между задачами обозначаются *связями*и используются, чтобы задать логику, определяющую последовательность работ в плане проекта.

***Ресурсы***в MS Project делятся на трудовые и материальные и обозначают сотрудников и оборудование, необходимые для выполнения проектных задач. При составлении списка ресурсов часто используется ролевое планирование, то есть для выполнения определенной задачи назначается *роль,*а конкретный сотрудник выбирается на эту роль на этапе реализации проекта.

Для каждого ресурса имеется важное свойство - его ***стоимость****.*В MS Project есть два типа стоимости ресурсов: ***повременная****ставка*и ***стоимость за использование****.*Повременная ставка выражается в стоимости использования ресурса в единицу времени. Обычно повременная ставка используется для учета трудовых ресурсов. Затраты на использование начисляются всякий раз, когда в задаче используется ресурс.

Назначение - это связь определенной задачи и ресурсов, необходимых для ее выполнения. Для выполнения задачи могут быть назначено несколько трудовых и материальных ресурсов.

Три составляющих проекта - время, стоимость, объем работ образуют проектный треугольник. При изменении одной из составляющих меняются две другие. Все три составляющие тесно связаны между собой. Например, если уменьшается время, то возрастет стоимость или уменьшится объем.

При изменении любой из трех составляющих надо убедиться, что это не приведет к уменьшению запланированного уровня качества проекта.

Планирование проекта в MS Project

Планирование начинается с определения проекта, то есть описания его ключевых характеристик. Затем составляется список задач и фаз и список необходимых для их выполнения ресурсов. Вносится дополнительная информация о задачах и ресурсах. Осуществляются назначения, после чего проект оптимизируется.

Определение проекта состоит в задании его ключевых характеристик. Эти характеристики задаются при создании нового файла проекта.

Планирование стоимости в MS Project

Общая стоимость проекта складывается из фиксированной стоимости ресурсов и задач и стоимости назначений, которая определяется ставками ресурса, трудозатратами и стоимостью использования ресурса.

Для каждого ресурса можно определить его стоимость использования в проекте: почасовую ставку или стоимость за использование. Стоимость назначения определяется стоимостью ресурса, умноженной на длительность назначения (по часовой ставке), либо фиксированной стоимостью ресурса.

Стоимость ресурса определяется на вкладке «Затраты» диалогового окна «Сведения о ресурсе».

Стоимость назначений определяется автоматически путем умножения ставки ресурса на трудозатраты и прибавлением к результату затрат на использование ресурса. Изменить стоимость назначения можно, указав другую тарификационную таблицу. Это mcokhq сделать на вкладке «Общие» диалогового окна «Сведения о назначении», где из списка можно выбрать таблицу А, В, С, D, Е.

Если в проекте часто применяются тарификационные таблицы, то можно в представлении добавить колонку «Таблица норм затрат» для удобства.

Стоимость задач складывается из суммарной стоимости назначений и фиксированных затрат. Фиксированные затраты - это затраты, не связанные с использованием проектных ресурсов. Например, стоимость авиабилетов.

Для ввода фиксированных затрат используется поле «Фиксированные затраты» в таблице «Затраты» в любом представлении для работы с задачами.

Планируя стоимость проекта, необходимо предусмотреть не только его бюджет, но и определить, как этот бюджет будет расходоваться на протяжении проекта. Расходование бюджета зависит от порядка оплаты работ. Оплачивать работу можно по-разному: может использоваться предоплата, оплата по факту завершения или оплата по мере выполнения работ. Причем в одном проекте могут сочетаться несколько способов оплаты.

Способ оплаты можно указать и для ресурсов, и для фиксированных затрат на задачу. На вкладке «Затраты» в диалоговом окне «Сведения о ресурсах» способ оплаты выбирается в раскрывающемся списке «Начисление затрат».

Определить порядок оплаты фиксированных затрат можно в колонке «Начисление фиксированных затрат» таблицы «Затраты» в любом представлении для работы с задачами.

По умолчанию способ начисления фиксированных затрат выбирается согласно параметра «Начисление фиксированных затрат по умолчанию» на вкладке «Расчет» диалогового окна «Параметры».

Анализ и оптимизация плана работ.

В ходе анализа плана проекта нужно оценить, насколько установленные длительности задач реалистичны. MSProject имеет средства для вероятностной оценки проекта.

Метод PERT (Program, Evaluation and Review Technique) использует три сценария: пессимистичный (максимальные длительности задач), оптимистичный (минимальные длительности задач) и ожидаемый. В соответствии с удельным весом каждого варианта программа рассчитывает средневзвешенную длительность каждой задачи.

Для анализа проекта по методу PERT надо вывести панель инструментов «Анализ по методу PERT».

Для анализа надо ввести для каждой задачи пессимистичную, оптимистичную ожидаемую длительности. Для этого надо нажать на кнопку «Лист ввода PERT».

Затем надо ввести весовые коэффициенты. Для этого на панели инструментов «Анализ по методу PERT» надо нажать кнопку «Задание весовых коэффициентов для метода PERT».

Сумма весовых коэффициентов всегда равна 6.

После определения весовых коэффициентов можно преступить к расчету для этого надо нажать кнопку «Вычисления по методу PERT». Перед этим следует сохранить файл проекта под другим именем, чтобы можно было вернуться к предыдущим данным.

MS Project выведет окно с предупреждением, что значения полей «Длительность», «Начало» и «Окончание» будут пересчитаны.

Анализ и оптимизация стоимости проекта

При анализе стоимости проекта обычно оценивается его бюджет (суммарные затраты на проект) и соотношение составляющих бюджета. Если общая стоимость проекта превышает ожидания, то бюджет оптимизируется.

Чтобы оценить общую стоимость проекта, достаточно перейти в таблицу «Затраты» в любом представлении со списком затрат и посмотреть данные в столбце «Общие затраты» у суммарной задачи проекта.

Также требуется проанализировать пропорциональное соотношение затрат внутри бюджета. При этом рассматриваются:

* распределение затрат по фазам проекта;
* распределение затрат по типам работ;
* соотношение между затратами на сверхурочные трудозатраты и обычные;
* распределение затрат на ресурсы разных типов.

Для этого можно воспользоваться формулами и настраиваемыми полями.

После проведения анализа может потребоваться провести оптимизацию проекта: уменьшить или увеличить затраты на задачи или ресурсы определенного типа.

Затраты определяются ставками ресурсов, трудозатратами и фиксированными затратами на задачи.

Для уменьшения затрат можно привлечь более дешевые ресурсы или использовать тарификационные таблицы с более низкими ставками. Привлечение более дешевых ресурсов может вызвать снижение качества проекта и увеличение сроков проекта.

Если у проекта оказывается дополнительный бюджет, то можно повысить качество проекта, привлекая более квалифицированные ресурсы и используя более дорогие материалы и оборудование.

Анализ рисков

Анализ опасностей, которые могут возникнуть при выполнении проекта, - один из самых сложных и ответственных этапов подготовки проекта. От качества проведения анализа во многом зависит успешный результат проекта.

Анализ рисков состоит из нескольких этапов:

• определение возможных рисков;

* определить стратегию действий при возникновении подобной ситуации.

Риски могут быть многих типов:

* риски в расписании проекта;
* ресурсные риски;
* бюджетные риски;
* социальные риски;
* политические риски;
* природные.

На обычный проект чаще всего могут оказывать влияние первые три типа рисков.

Для оценки рисков в расписании проекта следует выделить задачи со следующими параметрами: задачи с предварительными длительностями, задачи с большой длительностью и в которых задействовано большое количество ресурсов, задачи с большим числом зависимостей, задачи с внешними зависимостями.

Для оценки ресурсных рисков следует выделить следующие ресурсы: работников с недостаточной квалификацией, ресурсы, чрезмерно загруженные, ресурсы с уникальными навыками, которые некем заменить.

Для оценки бюджетных рисков требуется определить насколько увеличение объема работы по проекту приведет к увеличению затрат на него и имеются ли бюджетные резервы для компенсации.

Для смягчения рисков требуется разработать стратегию их сдерживания и уменьшения влияния. Для этого надо заложить в проект временной и бюджетный запас, а также предусмотреть меры по дублированию уникальных ресурсов.

Отслеживание проекта в MS Project.

Диаграмма Ганта

Диаграммы являются графическим средством отображения содержащейся в проектном файле информации. В MS Project диаграммы являются не только средством просмотра проектной информации. С помощью диаграмм можно вводить и редактировать данные.

Диаграмма Ганта - это один из наиболее популярных способов *графического представления плана проекта*, применяемый во многих программах управления проектами.

Она представляет собой график, на котором по горизонтали расположена шкала времени, а по вертикали -список задач. Длина отрезков, обозначающих задачи пропорциональна длительности задач. Задачи соединяются между собой линиями связей.

Рядом с отрезками может отображаться дополнительная информация, определяемая настройками программы.

В MS Project входят несколько представлений, основанных на диаграмме Ганта: «Диаграмма Ганта», «Подробная диаграмма Ганта», «Диаграмма Ганта с выравниванием», «Диаграмма Ганта с отслеживанием» и т.д.

Форматирование диаграммы Ганта

Для изменения отображения информации на диаграмме Ганта используется форматирование. С помощью средств форматирования можно изменять форму и цвет составляющих диаграмму фигур, определять информацию, отображаемую рядом с фигурами, отображать дополнительную графическую информацию, форматировать шкалу времени.

Настройка формы и цвета отрезков определяется в диалоговом окне форматирования отрезков. Для открытия окна можно дважды щелкнуть левой кнопкой мыши на отрезке или щелкнуть правой кнопкой мыши и выбрать из выпадающего меню пункт «Форматировать отрезок...».

Диалоговое окно «Шкала времени» имеет четыре закладки. Закладки «Верхний уровень», «Средний уровень», «Нижний уровень» настраивают три уровня временной шкалы. Единицы измерения времени более высокого уровня не могут быть меньше единиц измерения более низкого.

Количество отображаемых уровней временной шкалы задается с помощью списка «Отображать».

Параметры отображения уровня: «Единицы» - определяет единицы измерения; «Интервал» - число единиц в одном делении шкалы уровня; «Надписи» - определяет формат даты.

Закладка «Нерабочее время» задает отображение на диаграмме нерабочих дней. (Воскресных и праздничных).

Для выбора периода времени, отображенного на временной шкале, используется диалоговое окно «Масштаб», которое вызывается из контекстного меню, выбором одноименной команды.

В диалоговом окне можно выбрать период времени, для отображения задач, запланированных на этот период.

Для быстрого изменения масштаба диаграммы Ганта можно воспользоваться кнопками «Увеличить» и «Уменьшить», расположенными на панели инструментов «Стандартная».

Для форматирования вспомогательных линий диаграммы используется диалоговое окно «Сетка». Это окно можно вызвать с помощью команды «Формат» -> «Сетка...» или из контекстного меню.

Для настройки дополнительных параметров диаграммы используется диалоговое окно «Макет». В этом диалоговом окне задаются дополнительные параметры отображения диаграммы: вид линий связи между задачами, формат даты, высоту отрезков и т.д.

Для быстрого форматирования диаграммы Ганта можно использовать мастер диаграмм Ганта. Вызвать мастер можно с помощью команды «Формат» -> «Мастер диаграмм Ганта».

Диалоговое окно «Шкала времени» имеет четыре закладки. Закладки «Верхний уровень», «Средний уровень», «Нижний уровень» настраивают три уровня временной шкалы. Единицы измерения времени более высокого уровня не могут быть меньше единиц измерения более низкого.

Количество отображаемых уровней временной шкалы задается с помощью списка «Отображать».

Параметры отображения уровня: «Единицы» - определяет единицы измерения; «Интервал» - число единиц в одном делении шкалы уровня; «Надписи» - определяет формат даты.

Закладка «Нерабочее время» задает отображение на диаграмме нерабочих дней. (Воскресных и праздничных).

Для выбора периода времени, отображенного на временной шкале, используется диалоговое окно «Масштаб», которое вызывается из контекстного меню, выбором одноименной команды.

В диалоговом окне можно выбрать период времени, для отображения задач, запланированных на этот период.

Для быстрого изменения масштаба диаграммы Ганта можно воспользоваться кнопками «Увеличить» и «Уменьшить», расположенными на панели инструментов «Стандартная».

Для форматирования вспомогательных линий диаграммы используется диалоговое окно «Сетка». Это окно можно вызвать с помощью команды «Формат» -> «Сетка...» или из контекстного меню.

Для настройки дополнительных параметров диаграммы используется диалоговое окно «Макет». В этом диалоговом окне задаются дополнительные параметры отображения диаграммы: вид линий связи между задачами, формат даты, высоту отрезков и т.д.

Для быстрого форматирования диаграммы Ганта можно использовать мастер диаграмм Ганта. Вызвать мастер можно с помощью команды «Формат» -> «Мастер диаграмм Ганта».

Литература:

1. Орлов С.А. Технологии разработки программного обеспечения: Учебник для вузов. / А.С.Орлов, Б.Я.Цилькер, 4-е изд. Стандарт третьего поколения. – СПб.: Питер, 2012. – 608 с. [с.75-98]

**ЧАСТЬ 6. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ ПРЕДМЕТНОЙ ОБЛАСТИ**

**6.1. ПРОЕКТИРОВАНИЕ В СРЕДЕ BPWIN**

В структурном подходе к анализу и проектированию используются в основном две группы средств, описывающих функциональную структуру системы и отношения между данными. Каждой группе средств соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых являются:

• DFD (Data Flow Diagrams) – диаграммы потоков данных;

• SADT (Structured Analysis and Design Technique – метод структурного анализа и проектирования) – модели и соответствующие функциональные диаграммы;

• ERD (Entity-Relationship Diagrams) – диаграммы «сущность-связь»

Диаграммы потоков данных и диаграммы «сущность-связь» наиболее часто используемые в CASE-средствах виды моделей.

Конкретный вид перечисленных диаграмм и интерпретация их конструкций зависят от стадии жизненного цикла ПО, на которой они применяются.

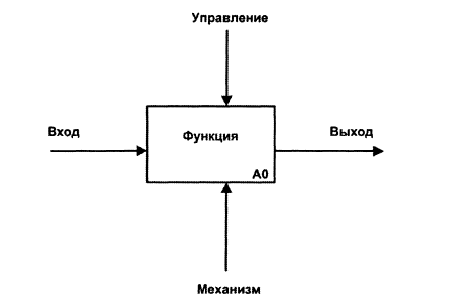
На стадии формирования требований к ПО SADT-модели и DFD используются, как правило, для моделирования бизнес-процессов (использование SADT-моделей ограничивается только данной стадией, поскольку они не предназначены для проектирования ПО). С помощью ERD выполняется описание данных на концептуальном уровне, не зависимом от средств реализации базы данных (СУБД).

На стадии анализа и проектирования ПО DFD используются для описания структуры проектируемой системы, при этом они могут уточняться, расширяться и дополняться новыми конструкциями. Аналогично ERD уточняются и дополняются новыми конструкциями, описывающими представление данных на логическом уровне, пригодном для последующей генерации схемы базы данных. Вышеперечисленные модели могут дополняться диаграммами, отражающими системную архитектуру ПО, структурные схемы программ, иерархию экранных форм и меню и др. Состав диаграмм в каждом конкретном случае зависит от сложности системы и необходимой полноты ее описания.

Метод SADT^ разработан Дугласом Россом (SoftTech, Inc.) в 1969 г. для моделирования искусственных систем средней сложности.

Данный метод успешно использовался в военных, промышленных и коммерческих организациях США для решения широкого круга задач, таких, как долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, разработка ПО для оборонных систем, управление финансами и материально-техническим снабжением и др. Метод SADT поддерживается Министерством обороны США, которое было инициатором разработки семейства стандартов IDEF (Icam DEFinition), являющегося основной частью программы ICAM (интефированная компьютеризация производства), проводимой по инициативе ВВС США. Метод SADT реализован в одном из стандартов этого семейства — IDEFO, который был утвержден в качестве федерального стандарта США в 1993 г., его подробные спецификации можно найти на сайте http://www.idef.com.

Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель

SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этого метода основываются на следующих концепциях: • графическое представление блочного моделирования. Графика блоков и дуг SADT-диафаммы отображает функцию в виде блока, а интерфейсы входа/выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описывается посредством интерфейсных дуг, выражающих «офаничения», которые, в свою очередь, определяют, когда и каким образом функции выполняются и управляются;

Построение SADT-модели

Построение SADT-модели заключается в выполнении следующих действий:

• сбор информации об объекте, определение его границ;

• определение цели и точки зрения модели;

• построение, обобщение и декомпозиция диаграмм;

• критическая оценка, рецензирование и комментирование.

Построение диаграмм начинается с представления всей системы в виде простейшего компонента - одного блока и дуг, изображающих интерфейсы с функциями вне системы. Поскольку единственный блок отражает систему как единое целое, имя, указанное в блоке, является общим. Это верно и для интерфейсных дуг — они также соответствуют полному набору внешних интерфейсов системы в целом.

Затем блок, который представляет систему в качестве единого модуля, детализируется на другой диаграмме с помощью нескольких блоков, соединенных интерфейсными дугами. Эти блоки определяют основные подфункции исходной функции. Данная декомпозиция выявляет полный набор подфункций, каждая из которых показана как блок, границы которого определены интерфейсными дугами. Каждая из этих подфункций может быть декомпозирована подобным образом в целях большей детализации.

Во всех случаях каждая подфункция может содержать только те элементы, которые входят в исходную функцию. Кроме того, модель не может опустить какие-либо элементы, т.е., как уже отмечалось, родительский блок и его интерфейсы обеспечивают контекст. К нему нельзя ничего добавить и из него не может быть ничего удалено.

Модель SADT представляет собой серию диаграмм с сопроводительной документацией, разбивающих сложный объект на составные части, которые изображены в виде блоков. Детали каждого из основных блоков показаны в виде блоков на других диаграммах.

Каждая детальная диаграмма является декомпозицией блока из диаграммы предыдущего уровня. На каждом шаге декомпозиции диаграмма предыдущего уровня называется родительской для более детальной диаграммы.

Синтаксис диаграмм определяется следующими правилами:

• диаграммы содержат блоки и дуги;

• блоки представляют функции;

• блоки имеют доминирование (выражающееся в их ступенчатом расположении, причем доминирующий блок располагается в верхнем левом углу диаграммы);

• дуги изображают наборы объектов, передаваемых между блоками;

• дуги изображают взаимосвязи между блоками:

* выход-управление;
* выход-вход;
* обратная связь по управлению;
* обратная связь по входу;
* выход-механизм.

Дуги, входящие в блок и выходящие из него на диаграмме верхнего уровня, являются точно теми же самыми, что и дуги, входящие в диаграмму нижнего уровня и выходящие из нее, потому что блок и диаграмма изображают одну и ту же часть системы.

Некоторые дуги присоединены к блокам диаграммы обоими концами, у других же один конец остается неприсоединенным.

Неприсоединенные дуги соответствуют входам, управлениям и выходам родительского блока. Источник или получатель этих пограничных дуг может быть обнаружен только на родительской диаграмме. Неприсоединенные концы должны соответствовать дугам на исходной диаграмме. Все граничные дуги должны продолжаться на родительской диаграмме, чтобы она была полной и непротиворечивой.

На SADT-диаграммах не указаны явно ни последовательность, ни время. Обратные связи, итерации, продолжающиеся процессы и перекрывающиеся (по времени) функции могут быть изображены с помощью дуг. Обратные связи могут выступать в виде комментариев, замечаний, исправлений и т.д.

Как было отмечено, механизмы (дуги с нижней стороны) показывают средства, с помощью которых осуществляется выполнение функций. Механизм может быть человеком, компьютером или любым другим устройством, которое помогает выполнять данную функцию.

Каждый блок на диаграмме имеет свой номер. Блок любой диаграммы может быть далее описан диаграммой нижнего уровня, которая, в свою очередь, может быть далее детализирована с помощью необходимого числа диаграмм. Таким образом, формируется иерархия диаграмм.

Для того чтобы указать положение любой диаграммы или блока в иерархии, используются номера диаграмм.

Типы связей между функциями

Одним из важных моментов при моделировании с помощью метода SADT является точная согласованность типов связей между функциями. Различают по крайней мере связи семи типов

(в порядке возрастания их относительной значимости):

• случайная;

• логическая;

• временная;

• процедурная;

• коммуникационная;

• последовательная;

• функциональная.

Случайная связь показывает, что конкретная связь между функциями незначительна или полностью отсутствует. Это относится к ситуации, когда имена данных на SADT-дугах в одной диаграмме имеют слабую связь друг с другом.

Логическая связь — данные и функции собираются вместе благодаря тому, что они попадают в общий класс или набор элементов, но необходимых функциональных отношений между ними не обнаруживается.

Временная связь — представляет функции, связанные во времени, когда данные используются одновременно или функции включаются параллельно, а не последовательно.

Процедурная связь — функции сгруппированы вместе благодаря тому, что они выполняются в течение одной и той же части цикла или процесса.

Коммуникационная связь — функции группируются благодаря тому, что они используют одни и те же входные данные и/или производят одни и те же выходные данные.

Последовательная связь — выход одной функции служит входными данными для следующей функции. Связь между элементами на диаграмме является более тесной, чем в рассмотренных выше случаях, поскольку моделируются причинно-следственные зависимости

Функциональная связь — все элементы функции влияют на выполнение одной и только одной функции. Диаграмма, являющаяся чисто функциональной, не содержит чужеродных элементов, относящихся к последовательному или более слабому типу связи.

Одним из способов определения функционально-связанных диаграмм является рассмотрение двух блоков, связанных через управляющие дуги.

Литература:

1. Вендеров А.М. Проектирование программного обеспечения экономических информационных систем: Учебник. - 2-е изд., перераб. и доп. — М.: Финансы и статистика, 2006. — 544 с. [с.108-132 ].

**ЧАСТЬ 7. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПРОЕКТИРОВАНИЯ И АНАЛИЗА ТРЕБОВАНИЙ К ПРОГРАММНОМУ ОБЕСПЕЧЕНИЮ**

**7.1. ПРОЕКТИРОВАНИЕ НА ЯЗЫКЕ UML**

Язык UML – это язык визуального моделирования, позволяющий разрабатывать концептуальные, логические и физические модели сложных систем.

Он предназначен для визуализации, анализа, спецификации, проектирования и документирования предметных областей, сложных систем вообще и ПС в частности.

Язык UML основан на следующих принципах объектно-ориентированного анализа и проектирования:

* *принцип абстрагирования*– предписывает включать в модель только те аспекты предметной области, которые имеют непосредственное отношение к выполнению проектируемой системой своих функций; абстрагирование сводится к формированию абстракций, определяющих основные характеристики внешнего представления объектов;
* *принцип инкапсуляции*– предписывает разделять элементы абстракции на секции с различной видимостью, что позволяет отделить интерфейс абстракции от его реализации; обычно скрываются структура объектов и реализация их методов;
* *принцип модульности*– определяет возможность декомпозиции проектируемой системы на совокупность сильно связанных и слабо сцепленных модулей; определение модулей выполняется при физической разработке системы, определение классов и объектов – при логической разработке;
* *принцип иерархии*– означает формирование иерархической структуры абстракций; принцип предписывает выполнять иерархическое построение моделей сложных систем на различных уровнях детализации;
* *принцип многомодельности*– обозначает, что при моделировании предметной области необходимо разрабатывать различные модели проектируемой сложной системы, отражающие различные аспекты ее поведения или структуры.

Для моделирования различных аспектов предметной области или проектируемой системы в языке UML предусмотрены следующие виды диаграмм:

1. диаграмма вариантов использования (Use Case Diagram);
2. диаграмма классов (Class Diagram);
3. диаграммы поведения (Behavior Diagram), в том числе:
4. диаграмма состояний (Statechart Diagram);
5. диаграмма деятельности (Activity Diagram);
6. диаграммы взаимодействия (Interaction Diagram), в том числе:
7. диаграмма последовательности (Sequence Diagram);
8. диаграмма кооперации (Collaboration Diagram);
9. диаграммы реализации (Implementation Diagram), в том числе:
10. диаграмма компонентов (Component Diagram);
11. диаграмма развертывания (Deployment Diagram).

Модели языка UML подразделяются на два вида:

* *структурные модели (статические модели)*описывают структуру сущностей предметной области или компонентов моделируемой системы, включая их классы, атрибуты, связи, интерфейсы; к данному виду моделей относятся диаграммы вариантов использования, классов, компонентов, развертывания;
* *модели поведения (динамические модели)*описывают функционирование сущностей предметной области или компонентов системы во времени, включая их методы, взаимодействия между ними, изменение состояний отдельных сущностей, компонентов и системы в целом; к данному виду моделей относятся диаграмма состояний, диаграмма деятельности, диаграмма последовательности, диаграмма кооперации.

Все модели языка UML подразделяются на три уровня:

* + ***концептуальные модели***представляют собой верхний, наиболее общий и абстрактный уровень описания моделируемой системы; к данному уровню относится диаграмма вариантов использования; с уровня концептуальной модели должно начинаться моделирование предметной области или проектируемой системы (программного средства);
  + ***логические модели***представляют собой второй уровень описания моделируемой системы; элементы моделей данного уровня не имеют физического воплощения и отражают логические аспекты структуры и поведения реальной предметной области или системы; логические модели должны строиться после концептуальных моделей; к данному уровню относятся диаграммы классов, состояний, деятельности, последовательности, кооперации;
  + ***физические модели***представляют собой нижний уровень описания моделируемой системы; элементы моделей данного уровня представляют собой конкретные материальные сущности физической системы; данные модели рекомендуется строить в последнюю очередь; к данному уровню относятся диаграммы компонентов и развертывания.

Процесс объектно-ориентированного анализа и проектирования, базирующийся на построении различных типов диаграмм UML, получил название рационального унифицированного процесса RUP (Rational Unified Process).

Основы данного процесса разработаны одним из авторов языка UML Джекобсоном.

Диаграмма вариантов использования описывает функциональное назначение моделируемой предметной области или системы.

Диаграмма классов является основной для создания кода приложения. Она описывает внутреннюю структуру программного средства, наследование и взаимное положение классов.

Диаграмма состояний описывает возможные последовательности состояний и переходов, выполняемых в ответ на некоторые события. Данная диаграмма характеризует поведение элементов модели в течение их жизненного цикла.

Диаграмма деятельности моделирует алгоритмическую и логическую реализации выполнения операций в системе и является аналогом схем алгоритмов, предназначенным для использования в объектно-ориентированных приложениях.

Диаграмма последовательности отображает синхронные процессы, описывающие взаимодействие объектов модели во времени. Время в данной модели присутствует в явном виде.

Диаграмма кооперации описывает структурные связи между взаимодействующими объектами модели.

Диаграмма компонентов описывает физическое представление проектируемой системы и позволяет определить ее архитектуру в терминах модулей, исходных и исполняемых кодов, файлов.

Диаграмма развертывания (диаграмма размещения) отображает общую конфигурацию и топологию распределенной системы. Данная диаграмма представляет распределение программных компонентов по отдельным узлам системы и маршруты передачи информации между ними.

Диаграмма вариантов использования является первой из диаграмм, разрабатываемых при моделировании предметной области, системы или программного средства. Она является базой при разработке спецификации функциональных требований и имеет основополагающее значение с точки зрения полноты и корректности дальнейшего моделирования проектируемой системы (программного средства).

**7.2. ДИАГРАММА ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ**

Диаграмма вариантов использования (Use Case Diagram) определяет функциональное назначение моделируемой системы или предметной области.

Данная диаграмма отображает множество актеров, взаимодействующих с проектируемой системой (программным средством) с помощью вариантов использования. Таким образом, основными элементами диаграммы вариантов использования являются актер и вариант использования.

Актер – это внешняя по отношению к моделируемой системе сущность, взаимодействующая с системой для решения некоторых задач. В качестве актера может использоваться человек, другая система, устройство или программное средство. Графическое изображение актера представлено на рис. а. Имя актера основано на использовании существительного.

Вариант использования определяет некоторый набор действий (операций), которые должны быть выполнены моделируемой системой или программным средством при взаимодействии с актером. Графическое изображение варианта использования представлено на рис. б. Название варианта использования базируется на неопределенной форме глагола.

В качестве примера рассмотрим предметную область автоматизации выполнения студентами лабораторных работ.

Очевидно, что внешними сущностями, взаимодействующими с моделируемой системой, являются актеры «Преподаватель», «Студент» и «Лаборант».

На рис. ниже приведен пример диаграммы вариантов использования системы автоматизации выполнения лабораторных работ для актеров «Преподаватель» и «Лаборант». Второй рисунок содержит тот же пример для актера «Студент».

Из рисунка следует, что при взаимодействии с актером «Преподаватель» система должна обеспечить возможность выполнения следующих основных функций:

* + выбор темы работы;
  + выдача индивидуального задания;
  + контроль хода выполнения индивидуального задания;
  + проверка результатов выполнения индивидуального задания;
  + проверка отчета;
  + прием лабораторной работы.

При взаимодействии с актером «Лаборант» система должна обеспечить возможность выполнения следующих основных функций:

* включение системы;
* проверка системы;
* выключение системы.

При взаимодействии с актером «Студент» система должна позволять выполнять следующие функции:

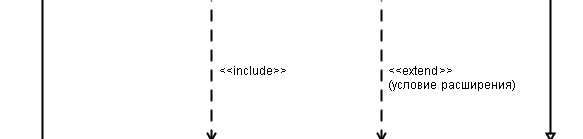
* изучение теории;
* ответы на контрольные вопросы по изученному материалу;
* выполнение индивидуального задания;
* оформление отчета по лабораторной работе;
* сдача лабораторной работы преподавателю.

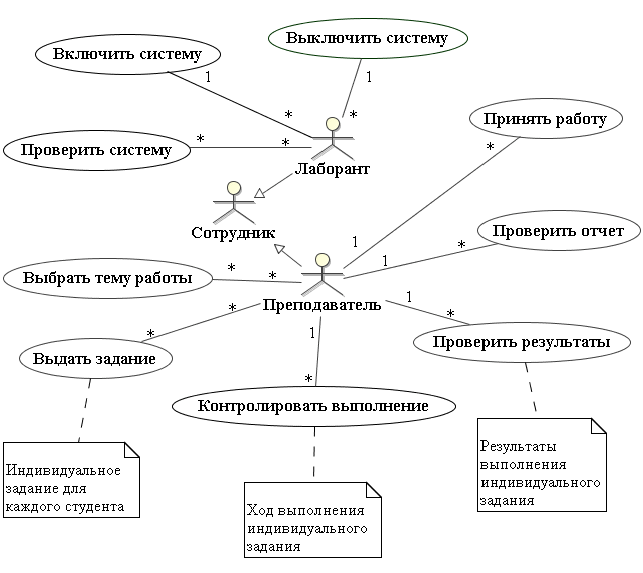
Помимо актеров и вариантов использования диаграмма вариантов использования может содержать примечания – элементы, служащие для размещения на диаграмме поясняющей текстовой информации. Графическое изображение примечания представлено на рис. в. Примечание может относиться к любому элементу диаграммы и соединяется с данным элементом штриховой линией.

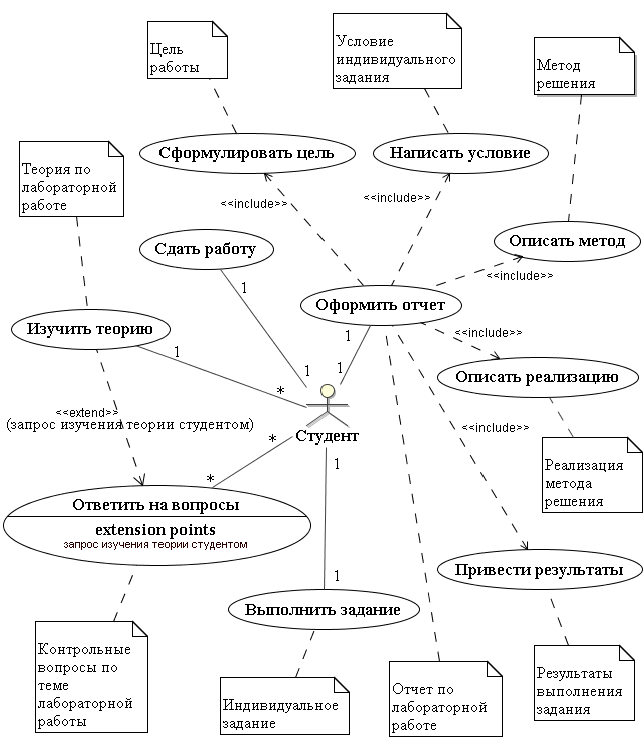
На диаграммах вариантов использования определены следующие виды отношений между отдельными элементами:

* отношение ассоциации;
* отношение включения;
* отношение расширения;
* отношение обобщения.

Графическое представление различных видов отношений приведено на рис.







Отношение ассоциации (association relationship) определено для всех видов диаграмм языка UML. На диаграммах вариантов использования данное отношение связывает актеров с вариантами использования и определяет конкретную роль актера при взаимодействии с вариантом использования. Графически отношение ассоциации изображается сплошной линией между актером и вариантом использования.

Например, между актерами «Преподаватель» и «Лаборант» и соответствующими вариантами использования существуют отношения ассоциации.

Отношения ассоциации характеризуются мощностью ассоциации. Мощность (кратность, multiplicity) ассоциации определяет количество экземпляров обеих сущностей, которое может участвовать в данной ассоциации. Графически значение мощности отмечается возле линии отношения ассоциации на стороне соответствующей сущности.

В диаграммах вариантов использования определено три типа мощности ассоциации: один-к-одному; один-ко-многим; многие-ко-многим. Возможна организация условной и биусловной ассоциации, при которой отдельные экземпляры одной или обеих сущностей могут не участвовать в ассоциации. Например, могут быть организованы мощности ассоциации ноль-или-один-к-нулю-или-одному, ноль-или-один-к-нулю-или-многим; ноль-или-много-к-нулю-или-многим.

Множественность ассоциации обозначается \* на стороне соответствующей сущности. Условность ассоциации обозначается введением нуля в обозначение мощности. Например, запись 0..1 обозначает мощность ноль-или-один, 0..\* мощность ноль-или-много. Если мощность ассоциативной связи не указана, то по умолчанию она принимается равной один-к-одному. Например, между актером «Преподаватель» и вариантом использования «Выбрать тему работы» существует ассоциативная связь мощностью многие-ко-многим, поскольку один преподаватель может выбрать разные темы работы, при этом одну и ту же работу могут выбрать разные преподаватели. Между актером «Преподаватель» и вариантом использования «Проверить отчет» существует связь мощностью один-ко-многим, так как один преподаватель может принимать много отчетов, но конкретный отчет принимается только одним преподавателем. На между актером «Студент» и вариантом использования «Оформить отчет» существует связь один-к-одному, что отражает тот факт, что один студент оформляет один отчет по лабораторной работе и каждый отчет оформляется одним студентом.

Отношение включения (отношение целое-часть, include relationship) может иметь место между двумя вариантами использования. Данное отношение определяет, что последовательность действий одного варианта использования (включаемого) включается в качестве составной части в последовательность действий другого варианта использования (базового). Графически отношение включения изображается штриховой стрелкой между вариантами использования, помеченной служебным словом «include». Стрелка направлена от базового варианта использования к включаемому варианту (базовый вариант «включает» действия включаемого варианта).

Например, базовый вариант использования «Оформить отчет» связан отношениями включения с вариантами использования «Сформулировать цель», «Написать условие», «Описать метод», «Описать реализацию» и «Привести результаты». По сути данные варианты использования определяют, что должно быть включено в содержание отчета по лабораторной работе.

Отношение расширения (extend relationship) может существовать между двумя вариантами использования. Данное отношение определяет, что некоторый вариант использования является расширением для другого варианта использования (базового). Это означает, что последовательность действий базового варианта использования может быть дополнена последовательностью действий варианта-расширения. Графически отношение расширения изображается штриховой стрелкой между вариантами использования, помеченной служебным словом «extend». Стрелка направлена от варианта использования, являющегося расширением, к базовому варианту (вариантрасширение «Расширяет» базовый вариант). Например, базовый вариант использования «Ответить на вопросы» связан отношением расширения с вариантом использования «Изучитьтеорию». Это означает, что при ответах студентом на контрольные вопросы может появиться необходимость в изучении теории. Таким образом, вариант использования «Изучить теорию», с одной стороны, связан отношением ассоциации с актером «Студент», то есть является независимым вариантом (изучение теории является этапом выполнения лабораторной работы). С другой стороны, данный вариант является расширением для варианта использования «Ответить на вопросы». Это отражает тот факт, что, во-первых, при ответах на контрольные вопросы студенту может не хватить полученных при изучении теории знаний и потребуется повторное обращение к изучению теории, и, во-вторых, то, что студент, обладающий достаточными знаниями, может без предварительного изучения теории отвечать на контрольные вопросы и только при необходимости обращаться к ее изучению.

Отношение расширения включает в себя условие, при котором вариантрасширение подключается к базовому варианту, и ссылки на точки расширения (extension points). Ссылки на точки расширения указывают на те места в базовом варианте использования, куда должно быть подключено расширение, если условие выполняется. На рисунке условием расширения варианта использования «Ответить на вопросы» является наличие запроса студента на изучение теории. Точками расширения базового варианта являются те точки, в которых при ответах на вопросы возник такой запрос.

Отношение обобщения (generalization relationship) может существовать как между актерами, так и между вариантами использования. Данное отношение определяет, что некоторая сущность А является специализацией сущности В. В этом случае сущность А называется потомком сущности В (дочерней сущностью), а сущность В – предком сущности А (родительской сущностью). При этом дочерние варианты использования наследуют свойства и поведение вариантов-предков и могут наделяться новыми свойствами и поведением. Актеры-потомки наследуют все роли актеров-предков. Графически отношение обобщения изображается сплошной линией с треугольной стрелкой. Стрелка направлена от сущности-потомка к сущности-предку. Например, актеры «Преподаватель» и «Лаборант» являются специализациями актера «Сотрудник», что определяется наличием соответствующих отношений обобщения между ними.

Следует отметить, что диаграмма вариантов использования представляет функции моделируемой системы или программного средства, но не отражает последовательности выполнения этих функций и связей между ними (в отличие, например, от методологии функционального моделирования IDEF0). Таким образом, для всестороннего анализа предметной области необходима разработка других диаграмм языка UML.

Литература:

1. Бахтизин В.В. Технология разработки программного обеспечения: учеб. пособие / В.В.Бахтизин, Л.А.Глухова. – Минск: БГУИР, 2010. – 267 с. [с.230 - 240].

**Часть 8. СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**8.1. ДИАГРАММА КЛАССОВ**

Детальное проектирование - это вторая ступень проектирования, которая следует за созданием архитектуры. В ходе этой деятельности ориентируются на максимальную подготовку к кодированию программной системы. Программисты должны получить детальные проектные решения, которые обеспечат их полной информацией для создания программного кода.

Напомним взаимосвязи между элементами всей предшествующей цепочки разработки: формирование требований - анализ требований - архитектурное проектирование.

При формировании требований создаются элементы Use Case, документирующие пожелания заказчика, эти пожелания детализируются и формализуются на этапе анализа, превращаясь в диаграммы последовательности. На этапе анализа уже приходится опираться на архитектурные черты будущей системы. Можно сказать, что параллельно с анализом начинается архитектурное проектирование.

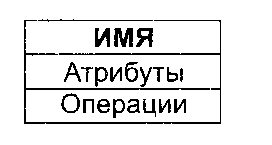
Архитектура и детальные требования питают фазу детализации проектирования. Архитектурный скелет обрастает деталями классами, способными реализовать сценарии, описанные диаграммами последовательности. 3авершается детальное проектирование в момент получения полного плана для этапа программирования.

Итак, основным строительным блоком детального проектирования являются классы. Создание структуры классов в языке UML поддерживается диаграммами классов.

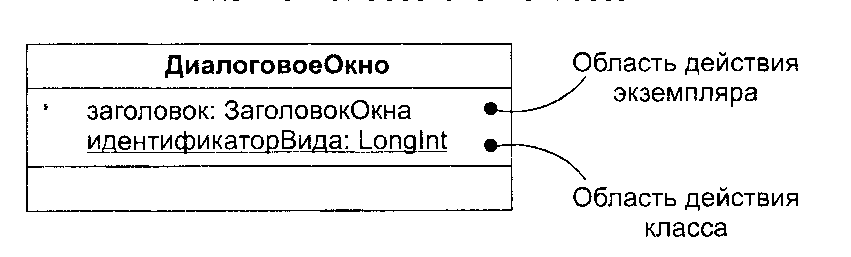
Вершины диаграмм классов нагружены классами, а дуrи (ребра) отношениями между ними. Диаграммы используются:

* в ходе анализа для указания ролей и обязанностей сущностей, которые обеспечивают поведение системы;
* в ходе проектирования для фиксации структуры классов, которые детализируют системную архитектуру.

Вершина в диаграмме классов это класс. Обозначение класса показано на рисунке.



Имя класса указывается всегда, атрибуты и операции выборочно. Предусмотрено задание области действия атрибута (операции). Если атрибут (операция) подчеркивается тo областью действия является класс, в противном случае областью действия является экземпляр класса.

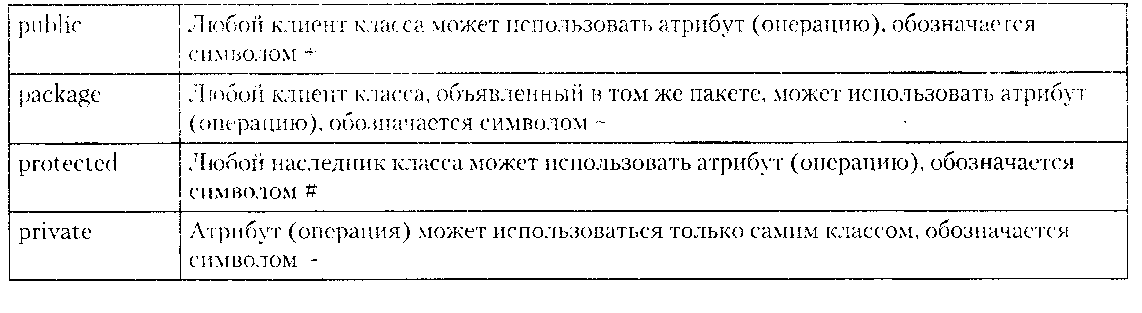


Если областью действия атрибута является класс, то все eгo экземпляры (объекты) используют общее значение этого атрибута, в противном случае у каждого экземпляра свое значение атрибута.

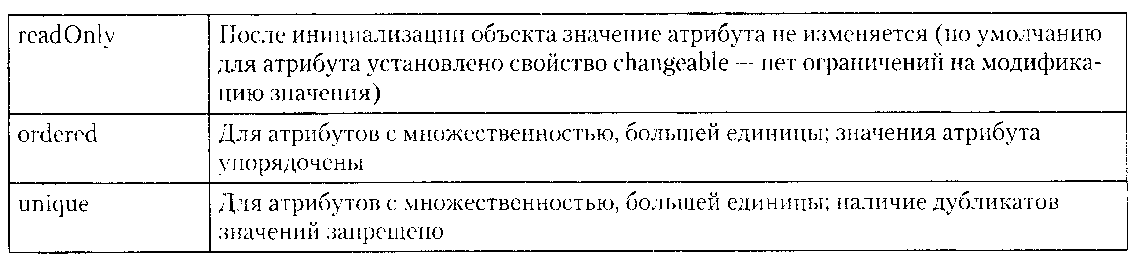
Общий синтаксис представления атрибута имеет вид:

Видимость Имя : Тип [множественность] = НачальЗначение {Свойства}

В языке UML определены четыре уровня видимости.



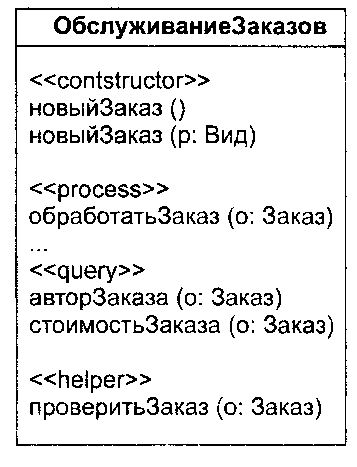
Свойств у атрибутов достаточно мнoгo, приведем лишь некоторые.



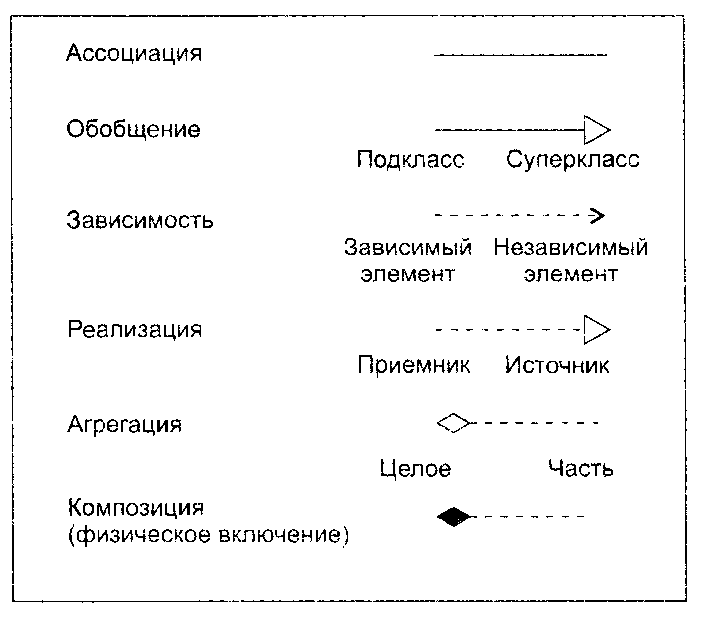
Общий синтаксис представления операции имеет вид:

Видимость Имя (Список параметров) : ВозвращаемыйТип {Свойства}

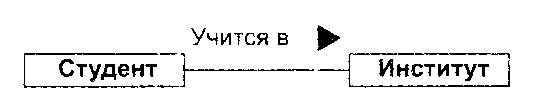
Известно, что пиктограмма класса включает три секции. Пустота секции не означает, что у класса отсутствуют атрибуты или операции, просто в данный момент они не показываются. Можно явно определить наличие у класса большего количества операций или атрибутов. Для этого в конце показанного списка проставляются три точки. Как показано на рисунке, в длинных списках атрибутов и операций разрешается группировка, каждая группа начинается со своего стереотипа.



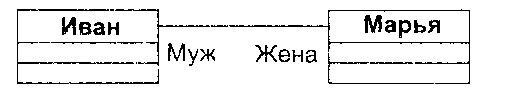
Отношения, используемые в диаграммах классов, наказаны на рисунке.



Ассоциации отображают структурные отношения между экземплярами классов, то есть соединения между объектами. Каждая ассоциация может иметь метку - имя, которое описывает природу отношения. Как показано на рисунке, имени можно придать направление достаточно поставить треугольник направления, который указывает направление. заданное для чтения имени.



Когда класс участвует в ассоциации, он играет в этом отношении определенную роль. Как показано на рисунке роль ассоциации определяет, какими представляется класс на одном полюсе ассоциации для класса на противоположном полюсе ассоциации.

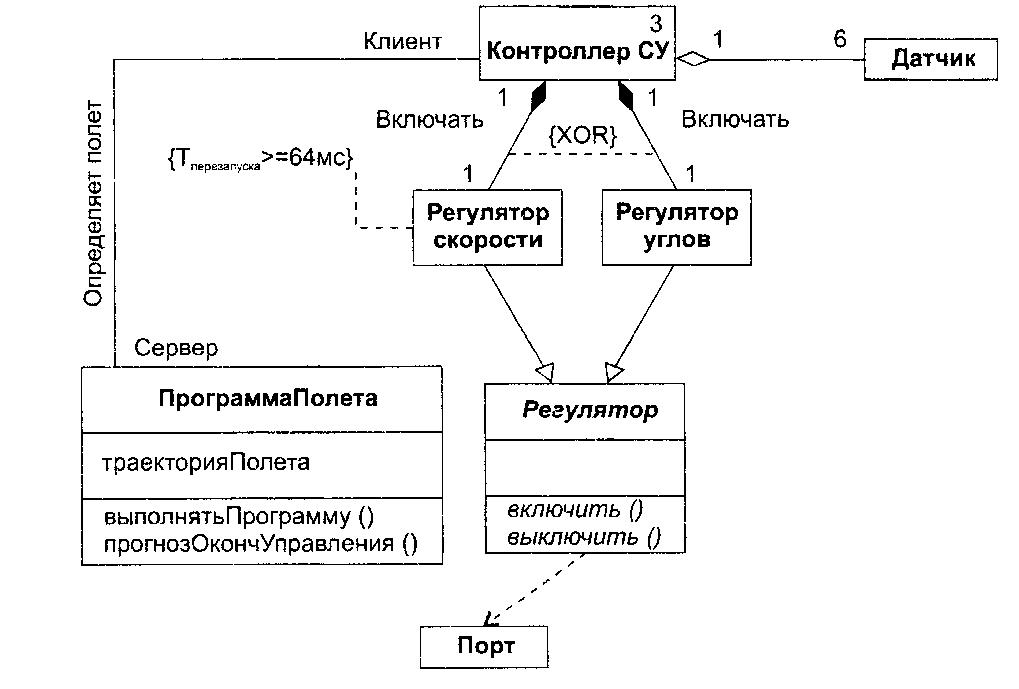


Часто важно знать, как много объектов может соединяться через экземпляр acсоциации. Это количество называется мощностью роли в ассоциации, записывается в виде выражения, задающего диапазон величин или одну величину.

Запись мощности на одном полюсе ассоциации определяет количество объектов соединяемых с каждым объектом на противоположном полюсе ассоциации.

Некоторые классы в иерархии классов могут быть абстрактными. Абстрактным называют класс, который не может иметь экземпляров. имена абстрактных классов записываются курсивом.

В качестве примера на рисунке показана диаграмма классов системы управления полетом летательного аппарата.



Литература:

1. Орлов С.А. Технологии разработки программного обеспечения: Учебник для вузов. / А.С.Орлов, Б.Я.Цилькер, 4-е изд. Стандарт третьего поколения. – СПб.: Питер, 2012. – 608 с. [с.230-240]

**Часть 9. СРЕДСТВА ПРОЕКТИРОВАНИЯ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ**

**9.1. ПРОЕКТИРОВАНИЕ ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ**

Оценка удобства интерфейса, и решений, предлагаемых для повышения удобства интерфейса, могут проводиться по следующим правилам.

Правило доступности. Система должна быть настолько понятной. Чтобы пользователь, никогда раньше не видевший ее, но хорошо разбирающийся в предметной области. Мог без всякого обучения ее использовать. Это правило служит некоторым идеалом, к которому можно стремиться, поскольку на практике достичь такой степени доступности почти ни когда не удается.

Правило эффективности. Система не должна препятствовать эффективной работе опытных пользователей, работающих с ней долгое время.

Правило непрерывного развития. Система должна способствовать непрерывному росту знаний, умений и навыков пользователя и приспосабливаться к его меняющемуся опыту. Нарушение непрерывности при переходе от одного набора возможностей к другому также приносит неудобства, поскольку пользователь вынужден разбираться с добавленными возможностями в новом контексте.

Правило соблюдения контекста. Система должна быть согласована с контекстом, в котором ей предстоит работать. В контекст могут входить специфика и объемы входных и выходных данных, тип и цели организации, уровень пользователей, зашумленность помещений и прочее.

Перечисленные правила определяют общие требования, которым должен удовлетворять интерфейс.

Принцип структуризации. ПИ должен быть целесообразно структурирован. Близкие по смыслу, родственные его части должны быть связаны видимым образом, а независимые – разделены; похожие элементы должны выглядеть похоже, а непохожие - различаться.

Принцип простоты. Наиболее распространенные операции должны выполняться максимально просто. При этом должны быть видимые ссылки на более сложные процедуры.

Принцип видимости. Все функции и данные, необходимые для решения определенной задачи. Должны быть видны, когда пользователь пытается ее решить.

Принцип обратной связи. Пользователь должен получать сообщения о действиях системы и о важных событиях внутри нее. Сообщения должны быть информативными, краткими, однозначными и написанными на языке, понятном пользователю.

Принцип толерантности. Интерфейс должен быть гибким и терпимым к ошибкам пользователя. Ущерб от ошибок должен снижаться за счет возможности отмены и повтора действий и за счет разумной интерпретации любых разумных действий пользователя и введенных им данных. По возможности следует избегать обязывающего взаимодействия (модальных диалогов), основанного на ограничении свободы пользователя.

Принцип повторного использования. Следует стараться многократно использовать внутренние и внешние компоненты, обеспечивая тем самым унифицированность интерфейса и сходство между его похожими элементами.

Человекомашинный интерфейс обеспечивает связь между пользователем и компьютером, он позволяет достигать поставленных целей, успешно находить решение поставленной задачи. Взаимодействие – обмен действиями и реакциями на эти действия между компьютером и пользователем. Существует ряд стилей взаимодействия, которые подразделяются на два основных вида.

1. Использование интерфейса команд – ввод команд текстовыми средствами.
2. Непосредственное манипулирование.

Количество информации, отображаемой на экране, называется экранной плотностью. Исследования показали, что чем меньше экранная плотность, тем отображаемая информация наиболее доступна и понятна для пользователя, и наоборот, если экранная плотность большая, это может вызвать затруднения в усвоении информации ее ясном понимании. Опытные пользователи могут предпочитать интерфейсы с большой экранной плотностью. Информация на экране может быть сгруппирована и упорядочена в значимые части. Это может быть достигнуто использованием кадров (фреймов), методов типа цветового кодирования, рамок, негативного изображения и других методов привлечения внимания.

Выделение элементов интерфейса яркостью – это размещение более ярких элементов на фоне более темных. Однако это может вызвать обратный эффект – перегрузку интерфейса.

Существует несколько приемов выделения яркостью:

* Движение (мигание или изменение позиции);
* Яркость – не очень действенный метод, т.к. люди имеют небольшое число уровней различения яркости;
* Цвет;
* Форма (символ, шрифт, форма символа;
* Использование различных шрифтов в различных формах;
* Размер – наиболее эффективно делать различие в 1,5 раза;
* Оттенение (различная текстура) объектов;
* Окружение (подчеркивание, рамки, инвертированное изображене).

Цвет – мощный визуальный инструмент, и применять его надо очень осторожно, чтобы не вызвать у пользователя дискомфорт от ошибочных цветовых комбинаций.

При разработке эргономичного интерфейса надо учитывать:

* Ограничить число цветов до 4 на экране и до 7 – для последовательных экранов;
* Для неактивных элементов использовать бледные цвета;
* Использовать цвет – как общепринятую кодировку: красный – запрет и т.п.
* Учитывать предметное представление о цвете: для картографа: зеленый – лес, желтый – пустыня, синий – вода; для химика: красный – горячо, синий – холодно и т.д.
* Для привлечения внимания наиболее эффективны белый, желтый и красный цвета.
* Для упорядочения данных используются цвета радуги.
* Для разделения данных используются цвета из разных частей спектра: кранный – зеленый; синий – желтый; любой цвет – белый.
* Для группировки данных, объединения и подобия нужно использовать соседние цвета спектра: оранжево – желтые, синие - фиолетовые и т.п.

При создании текстовых диалогов необходимо учитывать:

* Текст в нижнем регистре читается приблизительно на 13% быстрее, чем текст, полностью напечатанный в верхнем регистре.
* Выровненный по правому краю текст читать труднее, чем равномерно распределенный текст с невыровненным правым полем.
* Оптимальный интервал между строками равен или немного больше, чем высота символов.

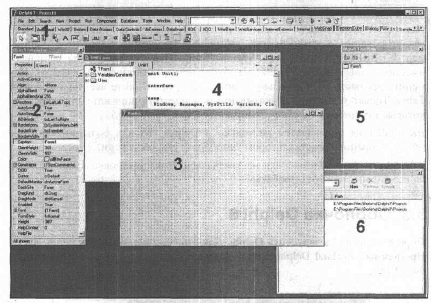
Литература:

1. Рудаков А.В. Технология разработки программных продуктов. Практикум: учеб.посоие для студ.учреждений сред.проф.образхования / А.В.Рудаков, Г.Н.Федорова. – 4-е изд., стер. – М.: Издательский центр «Академия», 2014. – 192с. [с.85-97].

**Часть 10. ВИЗУАЛЬНЫЕ СРЕДСТВА РАЗРАБОТКИ ПРИЛОЖЕНИЙ**

**10.1.** **СРЕДА РАЗРАБОТКИ DELPHI**

После запуска откроется окно, показанное рисунке.

В главном окне будет сформировано пять основных окон.

1. Главное окно программы. В нем находится основное меню, панели инструментов и палитра компонентов.
2. Инспектор объектов. Он предназначен для управления объектами и состоит из двух вкладок.

• Properties — свойства. На этой вкладке будут перечислены свойства выделенного объекта. Имя и тип выделенного объекта отображаются в выпадающем списке, вверху окна.

• Events — события. Здесь можно создавать и изменять реакцию объекта на различные события, которые возникают в компьютере.

1. Форма. Это уже готовая визуальная форма будущей программы. На ней мы будем размещать компоненты пользовательского интерфейса программы.
2. Редактор кода. В этом окне вы будете писать программу на языке Delphi.
3. Дерево компонентов. С его помощью легко находить компоненты, потому что они расположены в виде дерева. Если у вас какой-то компонент будет полностью перекрывать другой, то вы можете выделить верхний компонент, а потом, в дереве компонентов, легко найти тот, который находится снизу.

В Delphi достаточно много настроек и все их знать не обязательно. Мы познакомимся только с теми, которые действительно могут на что-то повлиять или что-то улучшить.

Для изменения основных настроек нужно выбрать из меню Tools (Инструменты) пункт Environment Options (Настройки окружения).

На вкладке Preferences (Предпочтения) вы можете задать ряд опций.

Autosave options — опции автосохранения. Здесь имеются два пункта.

• Editor files — сохранение редактируемых файлов (модулей). Если вы поставите галочку рядом с этим пунктом, то модули будут сохраняться автоматически.

• Project desktop — сохранение рабочей среды. Если вы поставите галочку напротив этого пункта, то состояние всех окон будет сохраняться автоматически.

Compiling and running — настройки процесса компилирования и запуска готовой программы. Здесь доступны несколько параметров.

• Show compiler progress — во время компиляции показывать окно состояния. В этом окне отображается информация о процессе компиляции и его результате. Окно очень полезно. Единственный его недостаток это то, что компиляция проходит немного дольше. При маленьких проектах это незаметно, но с большими программами задержка может быть ощутимой из-за затрат процессорного времени при выводе информации на экран.

• Warn on package rebuild — предупреждать, когда необходима перекомпиляция пакета.

• Minimize on run — минимизировать оболочку, когда запущена программа. Параметр действует, когда вы запускаете программу из Delphi. Если вы запустите скомпилированную программу из проводника, то Delphi не будет минимизирован.

• Hide designers on run — прятать окна объектного инспектора и визуальной формы при запуске программы. По умолчанию этот параметр выставлен, но я советую вам его отключить, чтобы вы могли выполнять программу и тут же корректировать какие-то ее элементы визуально. Если этот параметр включен, то все окна визуального редактирования пользовательского интерфейса во время выполнения программы будут спрятаны.

Как только вы попросите Delphi скомпилировать программу, перед вами появится окно в котором довольно хорошо отображается состояние компиляции. Интересовать здесь нас должны три значения.

Hints — сообщения. Это простые сообщения, которые указывают на места, где можно улучшить код. Например, вы объявили переменную, но не пользовались ей. В этом случае появится соответствующее сообщение. Это, конечно же, не ошибка и программа все же будет скомпилирована. Но благодаря этим сообщениям можно увидеть, где была объявлена

лишняя переменная или, возможно, просто что-то было забыто.

Warning — предупреждения. На них нужно обращать более пристальное внимание. Например, вы объявили переменную, затем попытались ее использовать, не присвоив начальное значение. В этом случае появится предупреждение. Это опять же не ошибка и программа будет скомпилирована, но Delphi предупреждает вас о возможной ошибке. Такие предупреждения нужно проверять, потому что вы действительно могли забыть что-то сделать и это уже может привести к фатальной ошибке выполнения программы.

Errors — это уже самые настоящие ошибки. Они указывают на те места, где была допущена грубая ошибка, из-за чего программа не может быть скомпилирована. Даже если вы откажитесь от показа окна состояния компиляции, вы все равно увидите все сообщения, ошибки и предупреждения. С этим окном просто удобнее проводить отладку программы.

**10.2. СРЕДА РАЗРАБОТКИ MS VISUAL STUDIO**

По сути своей платформа .NET является новой структурой создания программного продукта, которая предоставляет прикладной программный интерфейс (API) к своим службам, а также

API-интерфейсы классических операционных систем Windows (особенно семейства Windows 2000). Это достигается за счет объединения ранее разрозненных технологий, созданных Microsoft в конце 90-х.

Среди них можно назвать службы компонентов СОМ+, структуру разработки веб-приложений ASP, поддержку XML и объектно-ориентированного дизайна, поддержку новых протоколов веб-служб, таких как SOAP, WSDL и UDDI, а также нацеленность на Интернет. Все это интегрировано в единую архитектуру DNA.

Корпорация Microsoft утверждает, что до 80% средств, направленных на исследования и разработки, тратится на платформу .NET и связанные с ней технологии. Результаты такой политики на сегодняшний день выглядят впечатляюще. Например, область охвата платформы .NET просто огромна. Платформа состоит из четырех групп программных продуктов.

• Набор языков, куда входят С# и Visual Basic .NET; набор инструментальных средств разработки, в том числе Visual Studio .NET; обширная библиотека классов для построения веб-служб и приложений, работающих в Windows и в Интернете; а также среда выполнения программ CLR (Common Language Runtime, общеязыковая среда выполнения), в которой выполняются объекты, построенные на этой платформе.

• Набор серверов .NET Enterprise Servers, ранее известных под именами SQL Server 2000, Exchange 2000, BizTalk 2000 и др., которые предоставляют специализированные функциональные возможности для обращения к реляционным базам данных, использования

электронной почты, оказания коммерческих услуг ≪бизнес-бизнес≫ (В2В) и т. д.

•Богатый выбор коммерческих веб-служб, называемых .Net My Services. За умеренную плату разработчики могут пользоваться этими службами при построении приложений, требующих идентификации личности пользователя и других данных.

•Новые некомпьютерные устройства, поддерживающие средства .NET, от сотовых телефонов до игровых приставок.

Microsoft .NET поддерживает не только языковую независимость, но и языковую интеграцию. Это означает, что разработчик может наследовать от классов, обрабатывать исключения и использовать преимущества полиморфизма при одновременной работе с несколькими языками. Платформа .NET Framework предоставляет такую возможность с помощью спецификации Common Type System (CTS, общая система типов), которой должны следовать все компоненты .NET. Например, в .NET любая сущность является объектом какого-нибудь класса, производного от корневого класса System.Object. Спецификация CTS поддерживает такие общие понятия, как классы, делегаты (с поддержкой обратных вызовов), ссылочные и размерные типы.

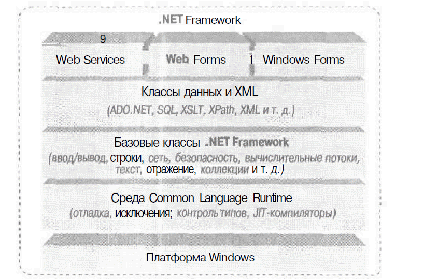
Кроме того, .NET включает спецификацию Common Language Specification (CLS, общая языковая спецификация), которая устанавливает: основные правила языковой интеграции. Спецификация CLS определяет минимальные требования, предъявляемые к языку платформы .NET. Компиляторы, удовлетворяющие этой спецификации, создают объекты, способные взаимодействовать друг с другом. Любой язык, соответствующий требованиям CLS, может использовать все возможности библиотеки FCL (Framework Class Library, библиотека классов платформы).

Платформа .NET Framework является надстройкой над операционной системой, в качестве которой может выступать любая версия Windows, и состоит из ряда компонентов. На сегодняшний день платформа .NET Framework включает в себя:

• Четыре официальных языка: С#, VB.NET, Managed C++ (управляемый C++) и JScript .NET.

• Объектно-ориентированную среду CLR, совместно используемую этими языками для создания приложений под Windows и для Интернета.

• Ряд связанных между собой библиотек классов под общим именем FCL.

Архитектурные компоненты платформы .NET Framework представлены на рисунке.

Самым важным компонентом платформы .NET Framework является CLR, предоставляющая среду, в которой выполняются программы. Она включает в себя виртуальную машину, во многих отношениях аналогичную виртуальной машине Java. На верхнем уровне среда активизирует объекты, производит проверку безопасности, размещает объекты в памяти, выполняет их, а также запускает сборщик мусора. (CTS также является частью CLR.)

На рисунке над уровнем CLR находится набор базовых классов платформы, а над ним расположены слой классов данных и XML, а также слой классов для создания веб-служб (Web Services), веб- и Windows-приложений (Web Forms и Windows Forms). Собранные воедино, эти классы известны под общим именем FCL (Framework Class Library).

Это одна из самых больших библиотек классов в истории программирования. Она предоставляет объектно-ориентированный API-интерфейс ко всем функциональным возможностям, инкапсулированным платформой .NET. Имея в своем составе более 4000 классов, библиотека FCL способствует быстрой разработке настольных, клиент-серверных и Других приложений и веб-служб.

Набор базовых классов платформы - нижний уровень FCL - аналогичен набору классов Java. Эти классы предоставляют элементарную поддержку ввода-вывода, манипуляций со строками, управления безопасностью, сетевой связи, управления вычислительными потоками, манипуляций с текстом, работы с отражениями и коллекциями и т. д.

Над этим уровнем находится уровень классов, которые расширяют базовые классы с целью обеспечения управления данными и XML. Классы данных позволяют реализовать управление информацией, хранящейся в серверных базах данных. В число этих классов входят классы SQL (Structured Query Language, язык структурированных запросов),

дающие программисту возможность обращаться к долговременным хранилищам данных через стандартный интерфейс SQL. Кроме того, набор классов, называемый ADO.NET, позволяет оперировать постоянными данными. Платформа .NET Framework поддерживает также целый ряд классов, позволяющих манипулировать XML-данными и

выполнять поиск и преобразования XML.

Базовые классы, классы данных и XML расширяются классами, предназначенными для построения приложений на основе трех различных технологий: Web Services (веб-службы), Web Forms (веб-формы) и Windows Forms (Windows-формы). Веб-службы включают в себя ряд классов, поддерживающих разработку облегченных распределяемых компонентов, которые могут работать даже с брандмауэрами и программами трансляции сетевых адресов (NAT). Поскольку веб-службы применяют в качестве базовых протоколов связи стандартные протоколы HTTP и SOAP, эти компоненты поддерживают в киберпростран-

стве подход ≪plug-and-play≫.

Инструментальные средства Web Forms и Windows Forms позволяют применять технику Rapid Application Development (RAD, быстрая разработка приложений) для построения веб- и Windows-приложений, Эта техника сводится к перетаскиванию элементов управления с панели инструментов на форму, двойному щелчку по элементу и написанию кода, обрабатывающего события, связанные с этим элементом.

Литература:

1. Либерти Дж. Программирование на C#. Создание .NET приложений. Изд. 2-е. М.:Символ-плюс. – 20г. 684с. [c. 21-26]
2. Фленов М.Е. Библия Delhpi. – БХВ-Петербург, 2004г. – 880с. [с. 45-55]

**Часть 11. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ТЕСТИРОВАНИЯ ПРИЛОЖЕНИЙ**

**11.1.** **ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПОДДЕРЖКИ ПРОЦЕССА ТЕСТИРОВАНИЯ**

К программным продуктам, рекомендованным к использованию в тестировании на разных этапах относятся:

1. **Rational Quantify**. Профилирование производительности;
2. **Rational Purify**. Отслеживание ошибок с распределением памяти;
3. **Rational PureCoverage**. Отслеживание области покрытия кода;
4. **Rational TestManager**. Планирование тестирования;
5. **Rational Robot**. Выполнение функциональных тестов.

Как при приемочном тестировании, так и при регрессионном разработчик должен предоставить не только работающий код, удовлетворяющий начальным требованиям, но и делающий это максимально безукоризненно с точки зрения производительности и устойчивости. В дополнение, разработчик должен максимально тщательно проверить исполнение всех строчек написанного им кода, во избежание дальнейших несуразностей, ведь одна из часто встречаемых ошибок — это копирование блоков программного кода в разные части программы, при том, что не все ветви первоначального блока были обработаны и протестированы.

Подобные первоначальные ошибки сводят на «нет» дальнейшее функциональное тестирование, так как низкокачественный код не позволяет тестерам быстро писать скрипт, реализующий тестирование новой формы, так как она полна странностей.

Следующие недостатки (относительные) программного кода могут потенциально привести приложение к состоянию зависания (входа в вечный цикл) или к состоянию нестабильности, проявляемой время от времени (в зависимости от внешних факторов):

* Использование рекурсии. Всем известна практическая ценность рекурсии для большого числа задач, особенно математических, но рекурсия является и источником проблем, поскольку не все компиляторы и не во всех случаях способны правильно отрабатывать рекурсию. Посему очень часто в требованиях на разработку вводится требование на отсутствие рекурсивных функций. Quantify ведет статистический анализ по вызовам функций и она выдает имена всех рекурсивных функций.
* Степень вложенности процедур (функций). В программе можно создать любое число функций (классов), вложить их друг в друга, а потом удивляться почему та или иная функция работает с не той производительностью. Если в требованиях на разработку ограничить полет фантазии разработчиков, то получится более быстрое приложение. Статистику по вложенности и скорости даст Quantify.
* Соглашения о именах. Также оговаривается в требованиях на разработку. Необходимо ввести единый стиль наименования функций, во избежание появления дублей и непонятных имен, что тормозит развитие проекта. Quantify также реализует поиски рассогласовании в именах.
* Использование указателей. Язык С и С++ трудно представить без указателей, даже невозможно. Но, являясь гибким и мощным механизмом обращения к памяти, они же являются минами замедленного действия, проявляющими себя в самые неподходящие моменты. Если проект не может существовать без указателей, то все ошибки, связанные с их неправильной работой позволит отловить Purify.
* Не целевое использование переменных и идентификаторов. Сюда попадают и ошибки связанные с операциями чтения и записи файлов до их открытия или после закрытия. Это и присвоение в переменных до их инициализации. Это и наличие лишних переменных. Эти неточности также могут привести к нестабильности приложения. Purify обрабатывает данную категорию ошибок.
* Не целевое использование блоков данных. Под эту категорию попадают ошибки связанные с распределением памяти, например, невозвращение блоков памяти после их использования. С точки зрения функциональности подобная ошибка не совсем ошибка, так как целостность приложения не нарушается и к сбоям не ведет. Побочный эффект — это замусоривание системы ненужными данными и быстрое «истекание» системных ресурсов. Данный вид отлавливается Purify.
* Присутствие участков кода не исполнявшихся в течении определенного времени. Также потенциальная ошибка, так как не выполненный код может содержать ошибки. PureCoverage отлавливает данный вид ошибок.
* Для осуществления всех функций по тестированию программных модулей, все три продукта используют специальную технологию Object Code Insertion, при которой бинарный файл, находящийся под тестированием, насыщается отладочной бинарной информацией, обеспечивающей сбор информации о ходе тестирования.
* Отметим общие черты для всех трех программных продуктов. Для сбора и обработки информации программам тестирования нужны два файла: исполняемый модуль и его исходный текст. Исполняемый модуль насыщается отладочным кодом, а наличие исходного текста позволяет разработчику легко переключаться между схематическим отображением и кодом тестируемого приложения.
* На код накладываются дополнительные особые ограничения, а именно: тестируемый код должен быть получен при помощи компиляции с опцией «Debug», то есть должен содержать отладочную информацию. В противном случае Quantify, Purify и PureCoverage не смогут правильно отображать (вообще не смогут отображать) имена функций внутренних модулей тестируемого приложения. Исключения могут составлять только вызовы внешних модулей из dll-библиотек, поскольку метод компоновки динамических библиотек позволяет узнавать имена функций.
* Отсюда можно сделать простой вывод: тестировать приложения можно даже не имея отладочной информации в модуле и при отсутствии исходных текстов, но в этом случае разработчик получает статистику исключительно по внешним вызовам.

Программные продукты могут работать в трех режимах:

1. Независимом графическом. В этом случае каждое средство запускается индивидуально, а тестирование осуществляется из него в графическом режиме;
2. Независимом командном. Данный режим характеризуется управлением ходом насыщения тестируемого модуля отладочной информации из командной строки;
3. Интегрированном. Этот режим позволяет разработчикам не выходя из привычной среды разработки (Visual Studio .NET) прозрачно вызывать инструменты **Quantify, Purify** и **PureCoverage**.

Из дополнительных возможностей всех инструментов хочется отметить наличие специального набора файлов автоматизации, разрешающих разработчикам еще на этапе разработки внедрять С-образные вызовы функций сбора информации по тестированию в свои приложения, получая при этом максимальный контроль над ними.

Средства анализа, строенные в **Quantify**, **Purify** и **PureCoverage** позволяют удовлетворить детально контролировать все нюансы в исполнении тестируемого приложения. Здесь и сравнение запусков, и создание слепков в ходе тестирования и экспорт в Excel для построения точных графиков множественных запусков.

Поддерживаются следующие языки программирования:

* Visual C\C++\C# в exe-модулях, dll-библиотеках, ActiveX-компонентах и COM-объектах;
* Поддерживаются проекты на Visual Basic и Java Applets (с любой виртуальной машиной);
* Дополнительно можно тестировать дополнительные модули к MS Word и Ms Excel.

Quantify, вставляя отладочный код в бинарный текст тестируемого модуля, замеряет временные интервалы, которые прошли между предыдущим и текущими запусками. Полученная информация отображается в нескольких видах: табличном, графическом, комбинированном.

Статистическая информация от Quantify позволит узнать какие dll библиотеки участвовали в работе приложения, узнать список всех вызванных функций с их именами, формальными параметрами вызова и с статистическим анализатором, показывающим сколько каждая функция исполнялась.

Гибкая система фильтров Quantify позволяет, не загромождая экран лишними выводами (например, системными вызовами), получать необходимую информацию либо только о внутренних, программных вызовах либо только о внешних, либо комбинируя оба подхода.

Вооружившись полученной статистикой, разработчик без труда выявит узкие места в производительности тестируемого приложения и устранит их в кратчайшие сроки.

Начать описание возможностей продукта Rational Purify хочется перефразированием одного очень известного изречения: «с точностью до миллиБАЙТА». Данное сравнение не случайно, ведь именно этот продукт направлен на разрешение всех проблем, связанных с утечками памяти. Ни для кого не секрет, что многие программные продукты ведут себя «не слишком скромно», замыкая на себя во время работы все системные ресурсы без большой на то необходимости. Подобная ситуация может возникнуть вследствие нежелания программистов доводить созданный код «до ума», но чаще подобное происходит не из-за лени, а из-за невнимательности. Это понятно — современные темпы разработки ПО в условиях жесточайшего прессинга со стороны конкурентов не позволяют уделять слишком много времени оптимизации кода, ведь для этого необходимы и высокая квалификация, и наличие достаточного количества ресурсов проектного времени. Как мне видится, имея в своем распоряжении надежный инструмент, который бы сам в процессе работы над проектом указывал на все черные дыры в использовании памяти, разработчики начали бы его повсеместное внедрение, повысив надежность создаваемого ПО. Ведь и здесь не для кого не секрет, что в большинстве сложных проектов первоочередная задача, стоящая перед разработчиками заключается в замещении стандартного оператора «new» в С++, так как он не совсем адекватно себя ведет при распределении памяти.

* Вот на создании\написании собственных велосипедов и позволит сэкономить Purify.
* Общие возможности по управлению Purify схожи с Quantify, за исключением специфики самого продукта. Здесь также можно тестировать многопоточные приложения, также можно делать «слепки» памяти во время тестирования приложения.
* Особенности использования данного приложения касаются спецификой отлавливаемых ошибок и способом выдачи информации.
* Информация выдается в виде списке, с наименованием найденной ошибки или предупреждения. При разворачивании списка с конкретной ошибкой выводится дополнительный набор данных, характеризующих ошибку.

Основное назначение продукта PureCoverage — выявление участков кода, пропущенного при тестировании приложения — проверка области охвата кода.

Очевидно, что при тестировании разработчику или тестировщику не удастся проверить работоспособность абсолютно всех функций. Также невозможно за один проход тестирования исполнить приложение с учетом всех условных ветвлений.

По требованиям на разработку программного кода, программист должен предоставить для функционального тестирования стабильно работающее приложение или модуль, без утечек памяти и полностью протестированный.

Понятие «полностью протестированный» определяет руководство компании в числовом, процентном значении. То есть, при оформлении требований указано, что область охвата кода 70%. Соответственно, по достижении данной цифры дальнейшие проверки кода можно считать нецелесообразными. Конечно, вопрос области охвата, очень сложный и неоднозначный. Единственным утешением может служить то, что 100% области охвата в крупных проектах не бывает.

Из трех рассматриваемых инструментов тестирования PureCoverage можно считать наиболее простым, так как информация им предоставляемая — это просмотр исходного текста приложения, где указано сколько раз исполнилась та или иная строка в приложении.

Литература:

1. Орлов С.А., Цилькер Б.Я. Технологии разработки программного обеспечения: Учебник для вузов. /4-е изд. Стандарт третьего поколения. – СПб.: Питер, 2012. – 608 с. [с. 488-500]

**Часть 12. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ДОКУМЕНТИРОВАНИЯ ПРИЛОЖЕНИЙ**

**12.1. ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА ПОДДЕРЖКИ ПРОЦЕССА ДОКУМЕНТИРОВАНИЯ**

Документация является органической, составной частью программного продукта для ЭВМ, и требуются значительные ресурсы для ее создания и применения. Тексты и объектный код программ для ЭВМ могут стать программным продуктом только в совокупности с комплексом документов, полностью соответствующих их содержанию и достаточных для его освоения, применения и изменения. Для этого документы должны быть корректными, строго адекватными текстам программ и содержанию баз данных — систематически, структурированно и понятно изложены, для возможности их успешного освоения и использования достаточно квалифицированными специалистами различных рангов и назначения. Качество и полнота отображения в документах процессов и продуктов в жизненном цикле программных средств должны полностью определять достоверность информации для взаимодействия заказчиков, пользователей и разработчиков, а тем самым корректность функций и достигаемое качество программных продуктов и соответствующих систем. Посредством документов (электронных или бумажных) специалисты взаимодействуют с программными средствами и данными в реализующих их вычислительных машинах, а также между собой.

Существует большая разница между тем, чтобы просто написать и запрограммировать некоторую функцию для индивидуального использования ее разработчиком, и тем, чтобы изготовить ее как качественный программный продукт, отчуждаемый от разработчиков, поставляемый заказчику и пользователям. Создание программного продукта требует значительных организационных усилий, ибо ее документация — это сложный живой организм, подверженный постоянным изменениям, которые могут вноситься многими специалистами. Управление документацией должно непрерывно поддерживать ее полноту, корректность и согласованность с программным продуктом. Необходимо обеспечивать возможность достоверного, формально точного общения всех участников проекта ПС между собой, с создаваемым продуктом и с документами для гарантии поступательного развития, совершенствования и применения комплекса программ. Адекватность документации требованиям, состоянию текстов и объектных кодов программ должна инспектироваться и удостоверяться (подписываться) ответственными руководителями и заказчиками проекта.

Ошибки и дефекты документов не менее опасны для применения ПС, чем ошибки в структуре, интерфейсах, файлах текстов программ и в содержании данных. Поэтому к разработке, полноте, корректности и качеству документации необходимо столь же тщательное отношение, как к разработке и изменениям текстов программ и данных.

Реализация документов ПС в значительной степени определяет достигаемое качество сложных программных продуктов, трудоемкость и длительность их создания. Для этого должны формироваться и использоваться регламентированная стратегия, стандарты, распределение ресурсов и планы создания, изменения и применения документов на программы и данные сложных систем. В общем случае должны быть выделены руководители и коллектив специалистов, которые будут планировать, описывать, утверждать, выпускать, распространять и сопровождать комплекты документов. Они должны стимулировать разработчиков ПС осуществлять непрерывное, полноценное документирование процессов и результатов своей деятельности, а также контролировать полноту и качество исходных, результирующих и отчетных документов ЖЦ ПС. Официальная, описанная и утвержденная стратегия документирования должна устанавливать дисциплину, необходимую для эффективного создания высококачественных документов на продукты и процессы в жизненном цикле ПС.

Методы и средства документирования каждой процедуры в стандартах обычно не раскрываются и адресуются к специальным нормативным документам различного уровня. Быстро оснащающиеся различными методами и инструментальными средствами этапы системного анализа, моделирования и проектирования ПС различных классов и назначения затрудняют стандартизацию этих процессов, достаточную для полной формализации структуры и содержания документов на программы и данные на уровне международных стандартов. Поэтому для этих этапов создаются нормативные документы — шаблоны на уровне стандартов де-факто, использующие, адаптирующие и дополняющие компоненты стандартов де-юре в разумной степени. Такие нормативные документы содержат выделенные фрагменты стандартов ЖЦ ПС и других стандартов, регламентирующих шаблоны программных документов на различных этапах проекта. В результате создаются и применяются проблемно-ориентированные совокупности методических руководств, отражающие наиболее современные методы, формы и фрагменты документов для документальной поддержки этапов и процессов жизненного цикла ПС, определенного класса или функционального назначения.

При создании и применении сложных ПС и обеспечении их жизненного цикла документами целесообразно применять выборку из совокупности стандартов, представленных в Приложении 1, детализирующих частные процессы, работы и документы. В результате на начальном этапе проектирования следует выделять и формировать целесообразный комплект шаблонов документов, обеспечивающих регламентирование всех этапов, процессов и документов при создании определенных проблемно-ориентированных проектов ПС. Для создания и реализации положений этих документов должны быть выбраны инструментальные средства, совместно образующие взаимосвязанный комплекс технологической поддержки и автоматизации ЖЦ и не противоречащие предварительно скомпонованному набору нормативных документов.

Процессы документирования программ и данных входят в весь жизненный цикл сложных систем и ПС. Поэтому организация и реализация работ по созданию документов должны распределяться между специалистами, ведущими непосредственное и преимущественное создание проектов комплексов программ, и специалистами, осуществляющими в основном разработку, контроль и издание документов. При создании особо сложных систем целесообразно выделение специального коллектива, обеспечивающего организацию и реализацию основных системных работ по документообороту ПС. Совокупные затраты на документирование крупных программных продуктов могут достигать 20—30% от общей трудоемКОСТИ проекта и необходимого числа (десятки) специалистов в жизненном цикле проекта ПС. В более простых случаях организация работ может быть упрощена, затраты на документирование снижаются приблизительно до 10%, однако всегда целесообразно выделять специалистов, непосредственно ответственных за создание, адекватность и контроль полноценного комплекта документов на программный продукт. Состав и общий объем документов широко варьируются в зависимости от класса и характеристик объекта разработки, а также в зависимости от используемой технологии.

Наиболее сложному случаю разработки критических ПС реального времени высокого качества соответствует самая широкая номенклатура документов. Такой перечень документов может быть использован как базовый для формирования на его основе состава и шаблонов документов в остальных более простых проектах.

Создание и применение ПС сложных систем сопровождается документированием этих объектов и процессов их жизненного цикла для обеспечения возможности освоения и развития функций программных средств и баз данных на любых этапах проекта ПС, а также для обеспечения интерфейса между разработчиками и с пользователями. По своему назначению и ориентации на определенные задачи и группы пользователей документацию ПС можно разделить на:

— технологическую документацию процессов разработки и обеспечения всего жизненного цикла, включающую подробные технические описания и подготавливаемую для специалистов, ведущих проектирование, разработку и сопровождение комплексов программ, обеспечивающую возможность отчуждения, детального освоения, развития и корректировки ими программ и данных на всем жизненном цикле ПС;

— эксплуатационную документацию программного продукта — объекта и результатов разработки, создаваемую для конечных пользователей ПС и позволяющую им осваивать и квалифицированно применять эти средства для решения конкретных функциональных задач систем.

Технологическая документация непосредственно и в наибольшей степени должна отражать процессы жизненного цикла комплексов программ и данных и требования к этим документам. Стандарты и нормативные документы, входящие в жизненный цикл проекта ПС, должны регламентировать структуру, состав этапов, работ и документов ЖЦ ПС. Они

должны: формализовать выполнение и документирование конкретных работ при проектировании, разработке и сопровождении ПС; обеспечивать адаптацию документов к характеристикам среды разработки, внешней и операционной системы; регламентировать процессы обеспечения качества ПС и его компонентов, методы и средства их достижения, реальные значения достигнутых показателей качества. Для контроля возможных изменений целесообразно предусматривать и согласовывать с заказчиком специальный документ, регламентирующий правила применения и корректировки номенклатуры, а также состава и содержания документации, поддерживающей ЖЦ ПС.

Эксплуатационная документация должна обеспечивать отчуждаемость программного продукта от первичных поставщиков — разработчиков и возможность освоения и эффективного применения комплексов программ достаточно квалифицированными специалистами — пользователями.

Эксплуатационные документы должны исключать возможность некорректного использования ПС за пределами условий эксплуатации, при которых документами гарантируются требуемые показатели качества функционирования ПС. Основная ее задача состоит в фиксировании, полноценном использовании и обобщении результатов функционирования объектов и процессов всего жизненного цикла ПС и системы.

Базой эффективного управления проектом ПС и его документированием должен быть План, в котором задачи исполнителей частных работ согласованы с выделяемыми для них ресурсами, а также между собой по результатам и срокам их достижения. План проекта должен отражать рациональное сочетание целей, стратегий действий, конкретных процедур, доступных ресурсов и других компонентов, необходимых для достижения основной цели с заданным качеством. Планирование реализации проектов и их документирования должно обеспечивать компромисс между характеристиками создаваемой системы и ресурсами, необходимыми на ее разработку и применение. Реализация плана зависит от результатов выполнения частных работ и документов и может требовать оперативной корректировки плана. Контроль обеспечивает исходные данные для координации элементов данной организации в соответствии с планом конкретной задачи. Для этого необходимо следить за ходом проекта и документирования на всем протяжении жизненного цикла и сравнивать запланированные и фактические результаты работ и документы. Контроль является органической функцией управления и должен иметь ряд средств регулирования поведения отдельных специалистов и коллектива разработчиков документов в целом.

При подготовке этих планов целесообразно по возможности разделять их цели и функции. План управления разработкой следует ориентировать на организацию специалистов, непосредственно создающих компоненты и ПС в целом, на эффективное распределение и использование ими ресурсов и средств автоматизации. В Плане управления документированием и обеспечением качества ПС внимание специалистов должно акцентироваться на анализе достигнутых результатов разработки, методах и средствах достижения заданных заказчиком характеристик ПС. При планировании и разработке комплекс документации должен проверяться и аттестовываться на полноту в условиях ограниченных ресурсов, на корректность, адекватность и непротиворечивость отдельных документов.

Различия в организационных службах и процедурах, методах и стратегиях приобретения ПС, масштабах и сложности проектов, требованиях систем и методах их разработки влияют на способы разработки, применения и сопровождения документов. Во многих предприятиях используются собственные нормативные шаблоны документов на процессы и продукты ЖЦ ПС, адаптированные к конкретным условиям разработки и характеристикам создаваемых ПС. В них выделяются состав и шаблоны документов, наиболее важные для проекта и качества создаваемого ПС, а также для конкретных заказчиков и пользователей. Соответственно определяются технологические и эксплуатационные документы, для которых регламентируются структура и основное содержание шаблонов документов.

Одна из важнейших задач документирования состоит в том, чтобы увязать четкими экономическими категориями взаимодействие разных специалистов в типовой производственной цепочке: заказчик — разработчик —изготовитель — пользователь документации. Для этого объект потребления — программный продукт, его документация и все процессы взаимодействия в цепочке должны быть связаны системой экономических и технических характеристик, в той или иной степени использующих основные экономические показатели — реальные затраты ресурсов: финансов, труда и времени специалистов на конечный программный продукт и документы. Сложность документирования, количество и полнота содержания комплекса документов в первую очередь зависят от масштаба —размера проекта ПС, что целесообразно оценивать в начале его ЖЦ. Для решения этой задачи необходимо детально учитывать требуемые ресурсы современных процессов создания, документирования и использования программ различных классов и назначения — встроенных, коммерческих, административных, учебных, уникальных.

Особое внимание в последнее время уделяется совершенствованию и детализации документов, обеспечивающих высокое качество создаваемых ПС, а также возможности их эффективного итерационного развития длительное время в многочисленных версиях. Соответственно должны изменяться документы, отражающие состояние процессов и компонентов проектов. Для этого организация процессов документирования должна обеспечивать гибкое и точное изменение документов — сопровождение и конфигурационное управление версиями и редакциями каждого документа.

Эти процессы и поддерживающие их средства автоматизации должны быть адекватными тем, которые применяются при сопровождении непосредственных объектов разработки — комплекса программ и данных.

Они должны быть поддержаны организацией контроля, регистрации и утверждения версии каждого документа, в той степени и на том уровне, которые необходимы в данном проекте для определенного документа.

Для хранения, тиражирования и распространения документов, сложных ПС высокого качества следует выделять группу специалистов, ответственных за контроль, обеспечение и гарантированное сохранение документации.

Для критических, важных систем документация на программы и данные должна храниться и дублироваться на различных типах носителей и эпизодически выводиться на бумажные носители. При определении схемы обеспечения сохранности документации разного содержания, следует учитывать ее важность, трудоемкость хранения и возможность аварийного восстановления. Кроме того, должна быть организована служба нормативного контроля, ответственная за соблюдение стандартов, нормативных и руководящих документов при подготовке документации всеми специалистами, участвующими в крупном проекте. Эта служба обязана обеспечить унификацию и высокое качество содержания, структуры и оформления шаблонов документов.

Для обеспечения достоверных данных об объектах и процессах управления документами ПС необходима автоматизированная база данных —информационная система обеспечения и хранения документов проекта. Такая информационная система документооборота представляет собой комплекс формальных и неформальных каналов поэтапного обмена информацией и документами между участниками проекта.

Степень формализации документооборота в коллективе специалистов может варьироваться от утверждаемых руководителями планов, технических заданий и документов на компоненты и версии ПС до личных бесед между разработчиками.

Литература:

1. Липаев В.В. Программная инженерия. Методологические основы. М.: ТЕИС. – 2006г.- 609 с. [с. 551-558]