Содержание

Аннотация

Введение

Глава 1. Теоретические Основы.....3

1.1 Цель и задача данного проекта.....4

1.2 Процесс создание и разработки программ.....4

1.3 Системный анализ программных систем.....5

1.4 Системный подход как исследование.....6

Глава 2. Процесс разработки программных систем.....8

2.1 Жизненный цикл ПО.....8

2.2 Этапы проектирование программных систем.....9

2.3 Управление программных проектами.....11

2.4 CASE - технологии.....12

2.5 Методологии разработки ПО.....13

Глава 3. Моделирование Программных систем.....14

3.1 Архитектура программной системы.....15

3.2 Структурное моделирование процессов.....18

3.3 Структурное моделирование информации.....19

Глава 4. Архитектуры распределенных программных систем.....20

4.1 Характеристики современных распределение систем.....22

4.2 Проблемы проектирование распределенных систем.....23

4.3 Основные типы архитектур PAC.....23

4.4 Клиент-серверные архитектуры.....23

4.5 Технологии проектирование PAC.....25

Глава 5. Объектное моделирование и программирование...26

5.1 Объективный подход и объектная декомпозиция.....27

5.2 Язык объектного моделирование UML.....28

5.3 Объектно-ориентированное программирование.....29

Глава 6. Обеспечение качества программных продуктов..31

6.1 Проблемы надежности и качества программных систем....32

6.2 Тестирование и рефакторинг программного кода....33

6.3 Качество программного обеспечение по ISO 9126.....34

Глава 7. Документирование программных систем...35

7.1 Документация в АСОИУ...35

7.2 Требования к программной документации....36

7.3 Стандартизация программной документации.....37

Заключение.....40

Список литературы

Приложение

Аннотация

Дисциплина “Проектирование архитектуры программных систем” является одной из ключевых дисциплин из цикла курсов подготовки современного специалиста в области информационных технологий, дает полную картину всего цикла промышленной разработки программного обеспечения с использованием современных методологий, технологий и инструментария разработки. В рамках данного курса рассматриваются все этапы разработки и закладываются основы теоретического и практического подходов к самому процессу разработки, как к четкой последовательности фаз, приводящей к гарантированному результату в заданных временных рамках и с заданным качеством

The discipline "Designing the architecture of software systems" is one of the key disciplines from the cycle of training courses for a modern specialist in the field of information technology, it gives a complete picture of the entire cycle of industrial software development using modern methodologies, technologies and development tools. Within the framework of this course, all stages of development are considered and the foundations are laid for theoretical and practical approaches to the development process itself, as a clear sequence of phases leading to a guaranteed result within a given time frame and with a given quality

Введение

Разработка программных систем является трудоемким, плохо формализуемым и одновременно творческим процессом. В условиях развитого рынка программного обеспечения разработчики вынуждены решать две во многом противоречивые задачи: с одной стороны, это создание программного комплекса в заданные сроки в соответствии с формальными требованиями заказчика и, с другой стороны, творческий процесс исследовательского поиска по его оптимальной реализации. Для достижения эффективных результатов при решении этих задач инженерия программного обеспечения требует применения специализированных методик управления всеми стадиями жизненного цикла программной системы: анализа, проектирования, реализации и эксплуатации разработанного программного продукта. Поскольку производство программных продуктов является одной из самых динамично развивающихся областей в сфере информационных технологий, что демонстрируется существенным ежегодным приростом объемов рынка реализации, необходимо развивать технологии, превращающие программирование из творческого поиска в инженерную науку.

1.1 Цель и задача данного проекта

Целями освоения дисциплины "Проектирование и архитектуры программных систем" являются получение теоретических знаний о принципах, технологии, методах и средствах проектирования архитектуры программных систем, а также приобретение практических навыков в выполнении действий по различным фазам создания программных продуктов

«Проектирование и архитектура программных систем» относится к вариативной части учебного плана и обеспечивает комплексную подготовку обучающихся по различным разделам теории и практики решения задач массового обслуживания, поиска оптимальных системотехнических и управленческих решений. Дисциплина связана с предшествующими ей дисциплинами: «Информатика», «Структуры и алгоритмы обработки данных», «Основы инженерного проектирования», а также с последующими дисциплинами «Тестирование программного обеспечения», «Конструирование программного обеспечения», «Проектированипе человеко-машинного интерфейса».

1.2 Процесс создание и разработки программ

Архитектурное проектирование Архитектурным проектированием называют первый этап процесса проектирования, на котором определяются подсистемы, а также структура управления и взаимодействия подсистем. Целью архитектурного проектирования является описание архитектуры программного обеспечения. Модель системной архитектуры часто является отправной точкой для создания спецификации различных частей системы. В процессе архитектурного проектирования разрабатывается базовая структура системы, определяются основные компоненты системы и взаимодействия между ними. Архитектурное проектирование Полезные определения: Подсистема — это система удовлетворяет "классическому" определению "система"), операции (методы) которой не зависят от сервисов, предоставляемых другими подсистемами. Подсистемы состоят из модулей и имеют определенные интерфейсы, с помощью которых взаимодействуют с другими подсистемами. Модуль — это обычно компонент системы, который предоставляет один или несколько сервисов для других модулей. Модуль может использовать сервисы, поддерживаемые другими модулями. Как правило, модуль никогда не рассматривается как независимая система. Модули обычно состоят из ряда других, более простых компонентов.

1.3 Системный анализ программных систем

Системный анализ означает синтез идей и принципов теории систем и кибернетики с возможностями современной вычислительной техники и используется для изучения и моделирования объектов сложной природы (систем). В основе системного анализа лежат такие основополагающие принципы, как:

1. структурированность
2. взаимосвязанность и согласованность проблем
3. целеполагание и разрешимость
4. допустимость, рациональность и оптимальность
5. ориентация на качественный результат
6. согласованность целей, средств и результатов
7. стабильность и изменчивость.

Сутью структурного принципа является сохранение качественных характеристик системы при ее разделении на части или объединении частей.

При этом используются два основных метода исследования: − декомпозиция – метод анализа и получения оценок проблемы на основе изучения свойств ее частей; − агрегирование – метод исследования на основе объединения подзадачи в единую задачу.

При последовательной декомпозиции совокупность составных частей образует так называемое иерархическое дерево, при этом процесс разбиения является итеративным. Выделение элементов одного уровня такой структуры следует проводить на основании следующих принципов: − существенности, выделять следует элементы, существенные для цели анализа данного уровня; − однородности, следует включать элементы, имеющие одинаковую важность по отношению к цели анализа данного уровня; − независимости, элементы анализируемого уровня должны быть взаимно независимы. Главной целью системного анализа является помощь в понимании и решении имеющейся проблемы, которая возникает при проектировании или управлении, путем ее перевода в задачу принятия решения. Представить систему в удобном для исследования виде и выступить в качестве инструмента концептуального проектирования позволяют модели системного анализа. Системный анализ программной системы использует три основных типа моделей: информационную, структурную и поведенческую, которые в процессе проектирования последовательно преобразуются в проектные решения по разработке базы данных, программной архитектуры и процедурных алгоритмов.

1.4 Системный подход как исследование

Системный подход представляет собой универсальное направление методологии исследования, основанного на рассмотрении объекта исследования как целостного множества взаимосвязанных элементов, т. е. как системы. При системном подходе основное внимание уделяется не анализу элементов как таковых, а изучению структуры объекта и взаимодействия составляющих систему элементов [4, 5]. Основными моментами, определяющими системность любого исследования, являются следующие:

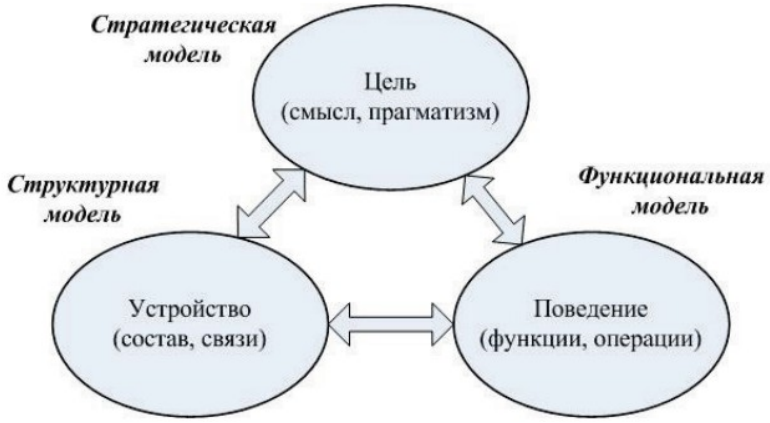
1. Исследование феномена целостности и установление состава элементов целого.
2. Изучение закономерностей соединения элементов в систему и определение структуры объекта, что является ядром системного подхода.
3. Установление функций системы в тесной связи с анализом ее структуры, т. е. структурно-функциональный анализ системы
4. Изучение и анализ генезиса системы, сфер ее влияния, связей с другими системами.
5. Системный подход как комплексная методология процесса исследования и анализа систем характеризуется следующими основными признаками.
6. Исследуемый объект оценивается как единое целое независимо от рассматриваемой точки зрения.
7. Решение проблем отдельных элементов подчиняется решению общих для всей системы проблем.
8. Познание объекта не ограничивается только механизмом функционирования, а распространяется и на установление внутренних закономерностей развития объекта.
9. Элементы системы, которые имеют второстепенное значение в одних условиях, при изменении обстоятельств могут оказаться существенными.

Главными принципами системного подхода являются:

1. Единство – система рассматривается как единое целое, являясь совокупностью составных частей.
2. Целостность – элементы системы могут быть разной направленности по функциональности и характеру поведения, но они одновременно должны быть совместимы между собой.
3. Динамичность – способность системы к изменению состояния под воздействием направленных или случайных факторов.

4. Взаимозависимость системы и среды – система проявляет свои свойства в процессе взаимодействия со средой.
5. Иерархичность – структурное упорядочивание частей, когда каждый элемент системы рассматривается как подсистема, а сама система – как элемент более сложной системы.
6. Организованность – формальное использование правил взаимодействия элементов для приведения в порядок составных частей и связей.
7. Множественность состояний – определение различных моделей, каждая из которых описывает определенное состояние системы.
8. Декомпозиция – возможность разделения объекта на составные части исходя из подцелей, возникающих из общей цели всей системы.
9. Сущность системного подхода определяется следующей совокупностью взаимосвязанных видов представления объекта.
10. Элементного, показывающего из каких элементов состоит система при ее отображении и исследовании.
11. Структурного, раскрывающего внутреннюю организацию системы, характер связей и способы взаимодействия компонентов.
12. Функционального, отвечающего на вопрос: «Какие функции выполняет сама система и образующие ее компоненты?»
13. Коммуникационного, определяющего взаимосвязь данной системы с другими как по горизонтали (сотрудничество), так и по вертикали (соподчиненность).
14. Интегративного, устанавливающего общие механизмы сохранения, совершенствования и развития системы.
15. Исторического, объясняющего то, каким образом возникла система, какие этапы проходила в своем развитии и каковы перспективы ее дальнейшего совершенствования.

Наибольшая эффективность применения системного подхода к исследованию и анализу появляется при комплексном его использовании при решениях как стратегических задач объекта, так задач определения структуры и поведения системы. Это позволяет рассматривать системный подход как триаду системных моделей, которая обеспечивает нужный эффект при реализации и эксплуатации программной системы.



В триаде системного подхода учитывается, что:

1. Любая система создана для решения определенных задач и имеет предназначение, задаваемое ее целями.
2. Для реализации этого предназначения она должна уметь выполнять определенные функции, владеть некими аспектами деятельности.

Общие цели и задачи достигаются в структурном взаимодействии функциональных элементов, между которыми установлены связи. Системный подход является еще и жизненной философией, которая помогает успешно решать проблемы повседневной жизни путем нахождения нестандартных решений, придерживаться «золотой середины» и избегать крайностей. Совершенствование системного мышления является непростым процессом, требующим интеллектуальных способностей

Глава 2. Процесс разработки программных систем

2.1 Жизненный цикл ПО

Жизненный цикл ПО представляет собой непрерывный и упорядоченный набор видов деятельности, осуществляемый и управляемый в рамках проекта по разработке и эксплуатации ПО. Он начинается с момента появления идеи (замысла) создания ПО, принятия решения о необходимости его создания и завершается процессом полного изъятия из эксплуатации. В жизненном цикле программного обеспечения обычно выделяют следующие этапы:

системный анализ и обоснование необходимости разработки;

1. Формирование требований.
2. Проектирование.
3. Программирование.
4. Тестирование и отладка.

5. Ввод программы в действие.
6. Эксплуатация и сопровождение.
7. Завершение эксплуатации.

Процессы жизненного цикла ПО в настоящее время регламентируются стандартом ISO 12207–1999 и показаны на (рис 2). Они объединены в три группы: основные, вспомогательные и организационные.

Основной процесс разработки ПО в данной модели жизненного цикла представляет собой анализ, проектирование и реализацию программной системы. Он включает работы по созданию ПО в соответствии с заданными требованиями, оформление проектной и эксплуатационной документации, подготовку материалов, необходимых для проведения тестирования и проверки работоспособности и качества программного продукта, организации обучения персонала.

2.2 Этапы проектирование программных систем

При планировании процесса разработки программной системы важное значение имеет выбор стратегии разработки при реализации проектных работ. Существуют три основных стратегии создания ПО:

1. Однократный проход (водопадная стратегия), в ходе которого выполняется линейная последовательность этапов конструирования
2. Инкрементная стратегия, заключающаяся в изначальном установлении всех пользовательских и системных требований, которые в дальнейшем реализуются в виде последовательности версий
3. Эволюционная стратегия, при которой программная система также строится в виде последовательности версий, но в начале процесса

определены не все требования, они уточняются в результате разработки версий.

Классический цикл проектирования реализуется водопадной (каскадной) моделью, представляющей последовательность этапов, при том, что переход на следующий этап происходит после завершения работ на текущем. В оригинальной каскадной модели Ройса эти этапы расположены в таком порядке:

− определение требований.

− проектирование.

− конструирование (реализация, кодирование).

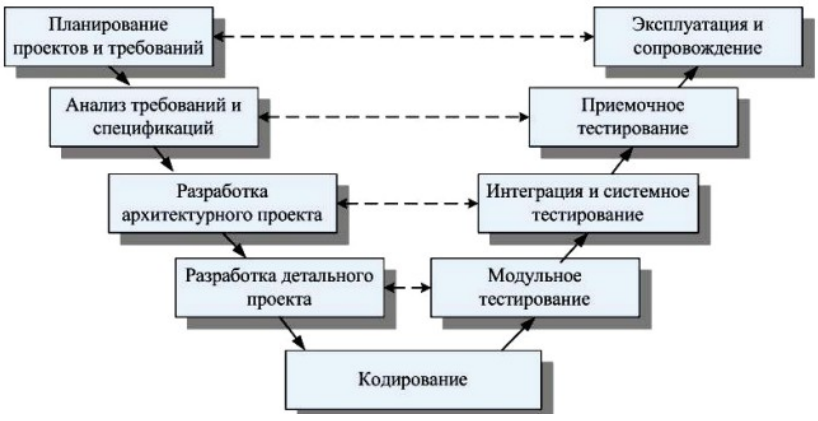
− воплощение.

− тестирование и отладка (в том числе верификация).

− инсталляция,

− поддержка.

Каскадная модель в чистом виде почти не используется из-за недостаточной гибкости процесса проектирования, однако она определяет логическую последовательность этапов проектирования, применяемую как в 22 итерационных, так и эволюционных моделях. В частности, широкое применение получила V-образная модель, которая была разработана как разновидность каскадной модели. Здесь каждая последующая фаза также начинается по завершению получения результативных данных предыдущей фазы. Однако в данной модели используется комплексный подход к определению фаз процесса проектирования ПО за счет выделения взаимосвязей, существующих между аналитическими фазами и фазами проектирования, предшествующими кодированию, и последующими фазами верификации и тестирования. Схематически данная модель представлена на рис. 3.



2.3 Управление программных проектами

Процесс проектирования программной системы является, как правило, достаточно сложным и противоречивым.

К основным источникам возникающих сложностей относятся:

− наличие и доступность высококвалифицированных специалистов на рынке труда

−стабильность используемой технологической платформы, стабильность и функциональность инструментов разработки

−эффективность используемых методов разработки, включая методы моделирования, проектирования, тестирования и управления версиями

−наличие и доступность специалистов, обладающих экспертизой в прикладной области

−используемая методология и ее соответствие данному проекту; сроки и финансирование проекта

−множество других организационных и технических переменных.

Данные сложности приводят к проблемам при управлении программными проектами, которые могут определяться следующими факторами.

1. Многие процессы разработки неуправляемы, их исходные данные и желаемый результат неизвестны или определены очень нечетко.
2. Процесс достижения желаемого результата не поддается формализации, наиболее это характерно для разработки архитектуры и исчерпывающего тестирования продукта.
3. Идентифицированные процессы разработки сопровождаются неизвестным количеством неидентифицированных.
4. Требования к продукту часто меняются в течение жизненного цикла проекта, что требует сложной процедуры изменения и согласования требований.
5. Попытки предложить формальную, детализованную методологию разработки ПО оказываются безуспешны, потому что сам процесс разработки не поддается детализации и формализации.
6. Слепое следование методологиям, предполагающим управляемость и предсказуемость процессов разработки, приводит к непредсказуемым результатам проекта.

Любой проект разработки ПО имеет свою организационную модель, которая определяет распределение обязанностей, ответственности и полномочий среди исполнителей проекта, а также отношений отчетности. Чем меньше проект, тем больше ролей приходится совмещать одному исполнителю. Всех исполнителей

типового проекта разработки ПО можно условно разделить на пять категорий в соответствии с их ролями:

1. анализ – сбор информации, исследование, составление и сопровождение требований к продукту
2. управление – организация и управление производственными процессами
3. производство – проектирование и разработка ПО
4. тестирование – проверка и контрольные испытания ПО
5. обеспечение – производство дополнительных продуктов и услуг

Для того чтобы обеспечить успешность программного проекта, необходимо реализовать его выполнение в соответствии со спецификациями, в срок и в пределах бюджета.

2.4 CASE - технологии

Многочисленные факторы, определяющие сложности в реализации программных проектов, привели к появлению программно-технологических средств автоматизации процесса проектирования ПО – CASE-средств. При этом реализуется специальная CASE-технология создания и сопровождения программных систем, которая представляет собой методологию автоматизированного проектирования ПО, а также инструментальные средства реализации. Это позволяет в наглядной форме представлять модель предметной области, анализировать ее на всех этапах разработки и сопровождения системы, разрабатывать программные компоненты в соответствии с информационными запросами пользователей. CASE технология включает в себя методы визуализации программных решений, с помощью которых на основе графической нотации строятся диаграммы, поддерживаемые инструментальной средой.

CASE-средства повышают производительность труда программистов и улучшают качество программного обеспечения. Это достигается тем, что они:

−Обеспечивают автоматизированный контроль совместимости спецификаций проекта

−Уменьшают время создания прототипа системы и ускоряют процесс проектирования и разработки

−Автоматизируют формирование проектной документации для всех этапов жизненного цикла

−Частично генерируют коды программ для различных сред разработки; − поддерживают технологии повторного использования компонентов

−Обеспечивают возможность восстановления проектной документации по имеющимся исходным кодам.

Автоматизация процессов разработки ПО предполагает широкое использование компонентного подхода к проектированию, т. е. реализацию его построения из отдельных компонентов, физически обособленных существующих частей, которые взаимодействуют между собой с использованием стандартизованных двоичных интерфейсов. Отличием объектов – компонентов ПО от обычных программных модулей является то, что объекты-компоненты можно собрать в динамически вызываемые библиотеки или исполняемые файлы и потом распространять их в двоичном виде без исходных текстов и применять в любом языке программирования, поддерживающем соответствующую технологию.

2.5 Методологии разработки ПО

В последнее время вопросу выбора методологии разработки программного обеспечения уделяется особое внимание, так как без правильной методологии даже небольшие проекты не могут быть успешными.

На разных уровнях и по разным критериям выделяют пересекающиеся модели: − водопадная (каскадная) модель, реализующая основное направление нисходящего структурного программирования

− макетирование

− спиральная (итерационная) модель разработки ПО

− объектно-ориентированное программирование

− гибкие (agile) технологии: экстремальное программирование (XP), Scrum, TDD, FDD

− методология моделирования RUP (Rational Unified Process); − методологии компонентного подхода (COM, CORBA, RAD).

Глава 3. Моделирование Программных систем

В практике проектирования программных систем широко применяется визуальное моделирование, которое осуществляется с использованием средств схематического или иного описания, анализа, проектирования и документирования решений по

структурной организации ПО. Модели используются для того, чтобы разработчик и специалист по эксплуатации мог понять структуру и поведение программной системы, управление проектом осуществлялось максимально легко и прозрачно, с уменьшением возможных рисков. Кроме того, моделирование дает возможность документировать принимаемые проектные решения и служит основой взаимодействия между участниками проекта, что способствует созданию качественного ПО. Документация по программной архитектуре представлена набором графических средств и нотаций представления моделей системы и их описанием. В описании указывается, из каких подсистем состоит система, из каких модулей формируется каждая подсистема. Графические схемы позволяют оценить структуру системы с разных сторон. Модель системной архитектуры является основой для создания спецификации системных компонентов, причем при ее проектировании разрабатывается базовая структура, т. е. определяются основные элементы системы и взаимодействия между ними. Процесс моделирования может быть разделен на несколько этапов: опрос экспертов, создание диаграмм и моделей, распространение документации, оценка адекватности моделей и принятие их для дальнейшего использования. Принятие решений по системной организации программной системы представляет собой многогранный, плохо формализованный и сложный процесс, в ходе которого широко используются системные исследования и анализ с использованием моделирования различных аспектов проектирования. При этом может использоваться широкий спектр методологий системного анализа и моделирования. К наиболее продуктивным из них следует отнести методологию структурного анализа и моделирования SADT. Она интегрирует процессы моделирования и управления конфигурациями проекта с использованием дополнительных языковых средств логической систематизации структуры и поведения, а также эффективной визуализации программных систем. 28 Основные правила формирования и визуализации структурных моделей базируются на методологии IDEF, которая предназначена для решения задач моделирования, отображения и анализа деятельностей разнообразных сложных систем в различных разрезах. Глубина и объем обследования процессов определяются самим разработчиком, что позволяет оптимизировать создаваемую модель, не перегружая ее излишними данными. В настоящее время данная методология содержит 15 стандартов (от IDEF 0 до IDEF 14), используемых для создания различных видов представления сложных систем при проведении структурного анализа. В целом SADT представляет собой совокупность методов, правил и процедур, используемых для построения структурной модели объекта автоматизации. Модели SADT отображают функциональную, процессную или информационную структуры объекта, включая производимые элементами системы действия и связи между этими действиями. Основными концепциями методологии структурного анализа и проектирования являются:

− графическое представление структурных блоков модели, при этом графика диаграммы представляет функцию в виде прямоугольного блока, а интерфейсы входа/выхода изображаются дугами, входящими в блок и выходящими из него

− строгость и точность, предполагающие выполнение формальных условий SADT. При создании SADT-диаграммы нужно руководствоваться следующими правилами: − ограничение количества блоков на каждом уровне декомпозиции (правило 3–6 блоков).

− связность диаграмм с помощью нумерации блоков; − уникальность наименований (отсутствие повторяющихся названий)

− формальность синтаксических правил для графики блоков и дуг; − разделение входов и управлений для определения роли данных

− исключение влияния организационной структуры на функциональную модель.

Эта методология предназначена для моделирования широкого круга систем, а также определения требований и функций, которые затем являются основой для проектирования. Для уже существующих систем методология используется с целью структурного анализа функций, выполняемых системой, и указания механизмов, посредством которых они осуществляются.

3.1 Архитектура программной системы

Под архитектурой программной системы понимается совокупность основополагающих решений по структурной организации программной системы. Она включает в себя:

− выбор структурных элементов и их интерфейсов, с помощью которых обеспечивается их взаимодействие и совместное функционирование

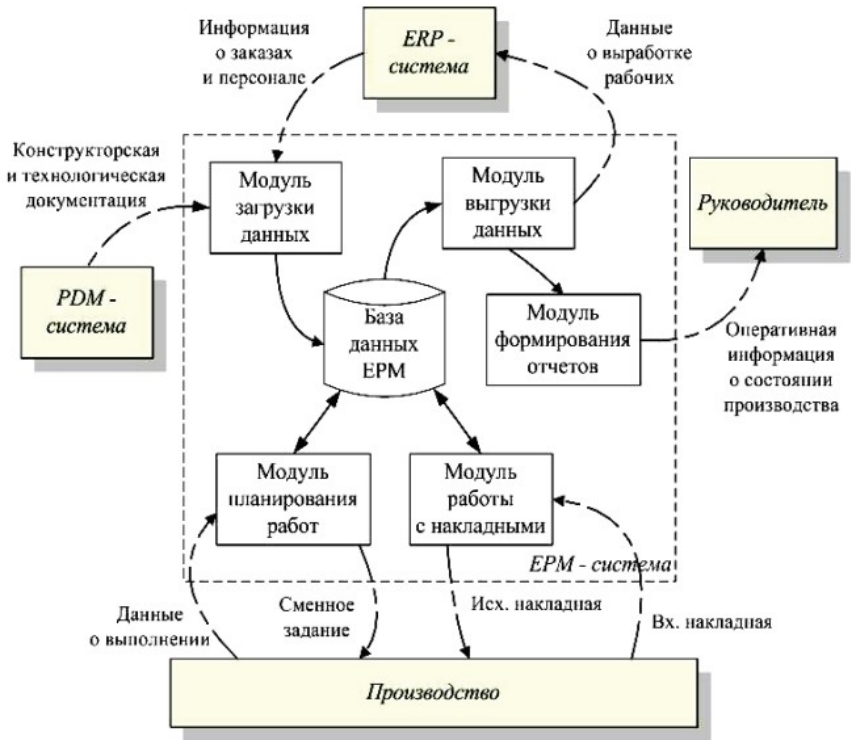
− соединение выбранных элементов структуры и поведения во всё более крупные системы

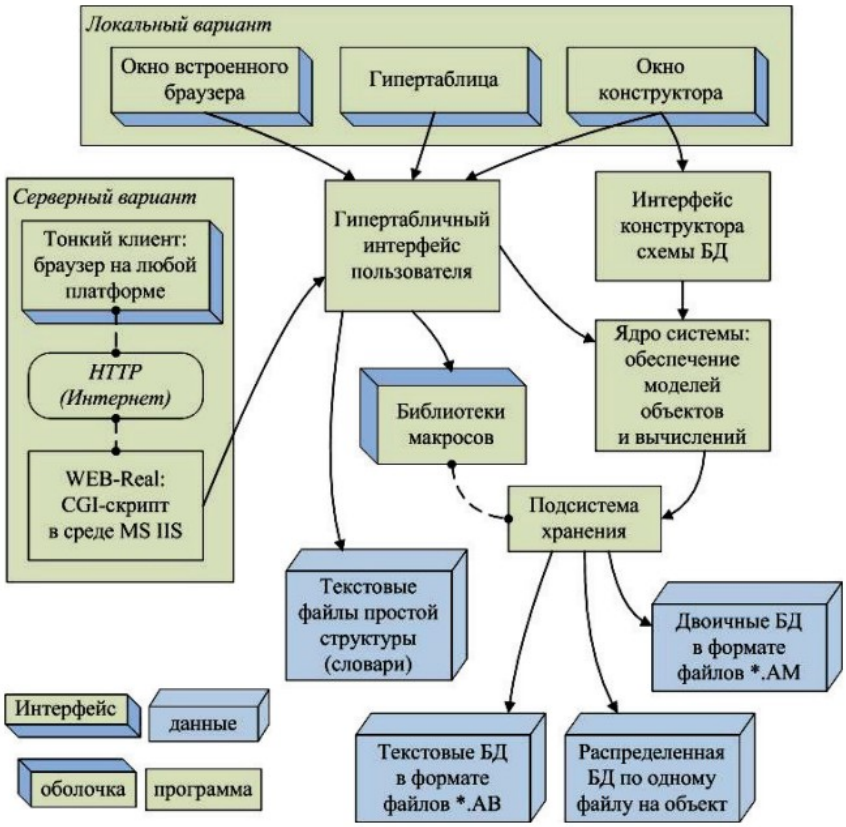
− архитектурный стиль, который направляет всю организацию – все элементы, их интерфейсы, их сотрудничество и их соединение.

Архитектурное проектирование является начальным этапом процесса проектирования, когда определяются основные подсистемы, процессы и структура управления и взаимодействия. Его результатом должно стать формирование структуры программного обеспечения. Архитектурный вид состоит из двух основных компонентов: структурных элементов и отношений между ними. При этом схематически архитектура может быть представлена в логическом или физическом

виде. Если структурные элементы представляют собой концептуальные (логические) компоненты ПО, такие как прецеденты, классы, процессы, состояния и др., то такая модель программной архитектуры будет соответствовать логической. Физическая же архитектура будет являться реализацией логической структуры ПС, когда ее элементами станут физически существующие компоненты, такие как программы, базы и файлы данных, интерфейсы.

На рис. 4 приведен пример логической структурной схемы программной архитектуры на примере системы управления эффективностью деятельности организации (EPM-система). На рис. 5 представлена физическая модель программной архитектуры WEB-системы.





Модели архитектуры определяются и рядом нефункциональных системных требований, к которым следует отнести следующие.

1.Производительность. Требование высокой производительности системы предполагает такую архитектуру, где за все критические операции отвечало бы минимальное число подсистем с ограниченным взаимодействием между ними. В таких случаях лучше использовать компоненты в виде крупных модулей, а не мелкие структурные элементы.

2.Защищенность, для обеспечения которой требуется архитектура с многоуровневой структурой, где наиболее критические системные элементы защищены на внутренних уровнях, а проверка их безопасности осуществляется на более высоком уровне.

3.Безопасность, для обеспечения которой архитектуру следует спроектировать так, чтобы при этом за операции, влияющие на безопасность системы, должно отвечать как можно меньше подсистем.

4.Надежность, повысить которую можно путем разработки архитектуры с включением избыточных компонентов, которые можно было бы заменять и обновлять, не останавливая общее функционирование системы.

5.Удобство сопровождения предполагает, что системная архитектура системы должна основываться на уровне мелких структурных компонентов, которые можно легко заменять и модифицировать. Программные модули, создающие данные, необходимо отделить от модулей, их использующих. Кроме того, нужно избегать структур совместного использования данных.

3.2 Структурное моделирование процессов

На первом этапе разработки программной архитектуры система разбивается на несколько взаимодействующих подсистем. На абстрактном уровне это показывается графически с помощью структурной схемы, в которой подсистемы представлены отдельными блоками. Стрелками, связывающими блоки, обозначаются потоки данных или потоки управления.

Характерными представителями методов визуализации структурных моделей системы являются такие методологии структурного анализа 33 SADT, как функциональное моделирование IDEF0 и моделирование потоков данных DFD.

В подходах IDEF0 структурной единицей системы является функциональный блок, выполняющий некоторое преобразование и связанный с внешним окружением и другими функциональными блоками интерфейсами (рис. 6).

Каждый функциональный блок в методологии IDEF0 может быть подвергнут декомпозиции, результатом которой является представление этого блока новой диаграммой, состоящей из функциональных блоков нижнего уровня структурной иерархии.

Диаграммы потоков данных (DFD) используются в качестве основного средства моделирования функциональных требований к системной архитектуре. С их помощью структурные элементы разбиваются на процессы (поведенческие компоненты) и представляются в виде связанной потоками данных сети.

3.3 Структурное моделирование информации

Большинство информационных хранилищ используют технологию реляционных баз данных, поскольку она предлагает надежные, проверенные и эффективные средства хранения и управления большими объемами данных. Важнейшим вопросом, связанным с конструированием хранилищ данных, является структура базы данных как логическая, так и физическая. Создание логической схемы информационной базы требует всеобъемлющего моделирования автоматизируемой деятельности. Цель моделирования данных состоит в обеспечении разработчика ПС концептуальной схемой БД в виде одной или нескольких локальных моделей, которые относительно легко преобразуются в систему информационных баз.

Самым известным и популярным средством моделирования данных являются диаграммы «сущность – связь» (ERD), позволяющие осуществить детализацию накопителей данных в моделях потоков данных, а также документировать информационные аспекты автоматизированной системы, важные для предметной области объекты (сущности), их свойства (атрибуты) и связи с другими объектами (отношения).

Сущность (Entity) характеризует множество однотипных экземпляров реальных или абстрактных объектов (людей, событий, состояний, предметов), обладающих общими атрибутами или характеристиками. Любой информационный объект может быть представлен одной сущностью, при этом имя которой должно отражать тип объекта, а не его конкретный экземпляр. Сущность должна обладать уникальным идентификатором, который и используется для однозначной идентификации экземпляра данного типа сущности.

Сущность должна обладать следующими свойствами:

− иметь уникальное имя; − иметь один или несколько атрибутов, которые могут принадлежат сущности или наследоваться через связь

− иметь один или несколько атрибутов, однозначно идентифицирующих каждый экземпляр. Сущность может иметь произвольное количество связей с другими сущностями. Связью (Relationship) называется поименованная ассоциация между двумя сущностями, значимая для рассматриваемой предметной области.

Атрибутом (Attribute) является любая характеристика сущности, значимая для рассматриваемой предметной области и предназначенная для идентификации, классификации, количественной характеристики или выражения состояния сущности. Атрибут – это тип характеристики или свойства, ассоциированных с множеством реальных или абстрактных объектов.

Среди методов визуализации моделей данных путем построения ER диаграмм наиболее популярными методами являются метод Баркера и метод IDEF1X, входящий в методологию SADT. Рассмотрим основные особенности решения этой задачи на примере последней.

IDEF1X является методом разработки реляционных БД и использует условную нотацию, специально разработанную для удобного построения концептуальной схемы, которая называется универсальное представление структуры данных, независимое от конечной реализации БД и аппаратной платформы.

Пример диаграммы «сущность – связь» представлен на рис. 7 моделью информационной базы системы учета заказов по обслуживанию компьютерной техники.

Создание модели данных начинается с разработки логической модели, после чего проектировщик может выбрать необходимую СУБД и создать соответствующую физическую модель

Глава 4. Архитектуры распределенных программных систем

Основным критерием при определении понятия распределенной автоматизированной системы (РАС) является разделение ее функций между несколькими компьютерами. При таком подходе к распределенной можно отнести любую программную систему, обработка данных в которой разделена между двумя и более компьютерами. Э. Таненбаум дает более узкое и точное определение, согласно которому распределенная система определяется как набор соединенных каналами

связи независимых компьютеров, которые с точки зрения пользователя выглядят единым целым программным комплексом. В таких системах функции одного уровня программного приложения могут быть разнесены между несколькими компьютерами, и в то же время программное обеспечение, установленное на одном компьютере, может отвечать за выполнение функций, относящихся к разным уровням. Поэтому подход к определению распределенной системы, считающей ее простой совокупностью компьютеров, является достаточно условным. Для более точного описания и реализации распределенных систем используется понятие программной компоненты. Под ней понимается единица программного обеспечения, исполняемая на одном компьютере в пределах одного процесса и предоставляющая определенный набор сервисов, которые используются через внешний интерфейс другими компонентами, выполняющимися как на том же компьютере, так и на удаленных компьютерах. Совокупность компонент пользовательского интерфейса предоставляют интегрированный сервис конечному пользователю.

4.1 Характеристики современных распределение систем

Чтобы достигнуть основной цели своего функционирования – обеспечения эффективного выполнения запросов пользователя – распределенная система должна удовлетворять ряду необходимых требований, который можно представить приведенным нижеследующим набором характеристик

1. Открытость. Внутри РАС все протоколы взаимодействия компонентов в идеале должны быть основаны на общедоступных стандартах, что позволяет использовать для их создания универсальные средства разработки и операционные системы. При этом каждый компонент должен содержать точную и полную спецификацию своих сервисов.

2.Масштабируемость. Наиболее важный аспект этой характеристики состоит в потенциальной возможности добавления в РАС новых компонентов и компьютеров для увеличения производительности системы и обеспечения балансировки нагрузки на серверы системы. К масштабированию относят также обеспечение эффективного распределения ресурсов сервера, обслуживающего запросы клиентов.

3.Поддержание логической целостности данных. Данное свойство предполагает, что запрос пользователя в распределенной системе должен корректно выполняться целиком или не выполняться вообще. Должны быть исключены ситуации, когда часть компонентов системы корректно обработали поступивший запрос, а часть – нет.

4.Устойчивость. Под этой характеристикой понимается возможность автоматического перераспределения функций внутри системы при выходе из строя одного из узлов, что обеспечивается дублированием несколькими компьютерами одних и тех же функций. В идеале это означает практически полное отсутствие ситуации, когда выход из строя одного любого компьютера приводило бы к невозможности обслужить пользовательский запрос.

5. Безопасность. Данные, передаваемые между компонентами, должны быть защищены как от искажения, так и от просмотра третьими сторонами, а функции системы должны использоваться авторизированными на это компонентами или пользователями.

6.Эффективность. Под свойством понимается минимизация накладных расходов, связанных с распределенным характером системы. Эффективность может противоречить безопасности, открытости и надежности системы, поэтому требование эффективности обычно является наименее приоритетным. Так, на поддержку логической целостности данных в распределенной системе можно потратить значительные ресурсы времени и памяти, но система с недостоверными данными вряд ли вообще будет нужна пользователям.

4.2 Проблемы проектирование распределенных систем

1. Сложность. РАС имеют более сложную организацию по сравнению с централизованными системами, что обусловливает определенные трудности при понимании, оценивании и тестировании их свойств. В частности, производительность системы зависит не от скорости работы отдельного процессора, а от полосы пропускания сети и совокупной скорости обработки информации разными процессорами. На производительность системы может радикально повлиять перемещение ресурсов из одной части в другую.

2.Безопасность. Поскольку доступ к системе можно получить с разных компьютеров системы, то сообщения в сети могут просматриваться и перехватываться. Поэтому в РАС намного сложнее поддерживать информационную безопасность.

3.Управляемость. Так как система может состоять из разнотипных компьютеров с разными версиями операционных систем, то ошибки, возникшие на одной машине, могут распространиться на другие с непредсказуемыми последствиями. Поэтому для того, чтобы управлять и поддерживать систему в рабочем состоянии, требуется значительно больше усилий. 4. Непредсказуемость. При возникновении различных событий реакция РАС может быть непредсказуемой, так как она зависит и от полной загрузки системы, и от ее организации и сетевой нагрузки. Причем эти параметры могут периодически изменяться, поэтому и время, потраченное на выполнение

запроса пользователя, может существенно различаться. Перечисленные трудности вызывают и соответствующие проблемы проектирования РАС, которые можно разделить на следующие группы.

1.Идентификация ресурсов. Так как ресурсы распределенной системы размещаются на разных компьютерах, то систему имен ресурсов следует продумать так, чтобы пользователи без труда открывали и ссылались на необходимые им ресурсы. Примером является система унифицированного указателя ресурсов URL, которая определяет адреса WEB-страниц.

Если не иметь легко воспринимаемую и универсальную систему идентификации, то большая часть ресурсов окажется недоступной.

2.Коммуникации. Наиболее эффективным способом организации взаимодействия между компьютерами для большинства распределенных систем является Internet и реализация протоколов TCP/IP. Однако в случаях, когда на производительность, надежность и другие характеристик накладываются специальные требования, можно воспользоваться и другими, альтернативными способами системных коммуникаций.

3.Качество системного сервиса. Под ним обычно понимается производительность, работоспособность и надежность сервиса, предлагаемого системой. Качество сервиса в РАС определяется целым рядом дополнительных факторов, таких как распределение системных процессов, системные и аппаратные средства, возможности адаптации системы.

4.Архитектура ПО РАС. Определяет распределение системных функций по компонентам системы, а также развертывание этих компонентов по процессорам. Для поддержки высокого качества системного сервиса выбор правильной архитектуры может оказаться решающим фактором.

4.3 Основные типы архитектур РАС

Архитектура РАС строится на основании принципов разделения, которые можно разделить на два типа: функциональное и естественное. Функциональное разделение основывается на выделении специфичных задач, решаемых узлами, например клиент – сервер, хост – терминал, сборка данных – обработка данных – предоставление данных. При естественном разделении учитываются требования, определяемые структурой самой задачи автоматизации, например обслуживание сети супермаркетов, сеть для поддержки коллективной работы и др.

Основными мотивами разделения являются:

1) обмен информацией между системами, при котором распределение нагрузки и балансировку загрузкой процессоров системы стремятся осуществлять так, чтобы оптимизировать общую загрузку системы;

2) усиление мощности, когда различные узлы должны работать над одной задачей так, чтобы система, содержащая набор процессоров, по мощности могла приближаться к суперкомпьютеру;

3) физический мотив, при котором система строится в предположении, что узлы физически разделены исходя из требования к надежности и устойчивости к сбоям

4) экономические мотивы, так как набор дешевых чипов может обеспечить лучшие показатели отношения цена/производительность, чем мэйнфрэйм

5) специализация компонентов, при которой происходит их упрощение и удешевление

− архитектура клиент – сервер как модель, представленная набором системных сервисов, которые предоставляются серверами клиентам, при этом по назначению и характеру использования серверы и клиенты значительно отличаются друг от друга;
− архитектура распределенных объектов как модель, в которой между серверами и клиентами не делается никаких различий, такая система представляется как набор взаимодействующих объектов, место расположения которых не имеет особого значения.

РАС проектируются на основе объектно-ориентированного подхода, конструируются из слабо интегрированных частей, каждая из которых может непосредственно взаимодействовать как с пользователем, так и с другими частями системы. Такие компоненты должны по возможности реагировать и на независимые события. Программные компоненты, разработанные на основе таких принципов, являются естественными частями РАС. Архитектура распределенных объектов представляется наиболее общим и универсальным подходом в проектировании РАС, при котором различия между клиентом и сервером стираются. Компонентами такой системы являются объекты, реализующие сервисы. При этом не делается различий между клиентом (пользователем сервиса) и сервером (поставщиком сервиса).

4.5 Технологии проектирование РАС

В инфраструктуру системной поддержки проектирования РАС входят:

− язык описания интерфейсов

− транслятор с этого языка

− библиотеки, реализующие функциональность, необходимую для удаленного вызова процедуры. Часть ее используется при разработке программ, использующих удаленный вызов процедур, другая часть необходима для реализации во время выполнения такого вызова. Аналогично работают и другие виды взаимодействия распределенных компонентов.

Системная поддержка РАС может иметь следующие формы организации системного взаимодействия.

1. Системы на базе удаленного вызова процедур. Являются наиболее общим способом системного взаимодействия. Они содержат программы для прозрачного преобразования обычных вызовов процедур в удаленные. Такие системы лежат в основе почти всех других форм системного программного обеспечения, включая сетевые службы.

2. Транзакционные мониторы, которые относятся к наиболее надежной и стабильной технологии интеграции прикладных распределенных систем. Они могут рассматриваться как системы удаленного вызова процедур с транзакционными расширениями.

3. Брокеры объектов. Появились в результате применения объектноориентированного подхода к системам на базе систем удаленного вызова. Такие системы более развиты, чем системы удаленного вызова, хотя и мало от них отличаются. Наиболее распространены брокеры объектов, построенные на основе подхода CORBA, предложенного группой OMG при том, что основная часть их функциональности уже была реализована в транзакционных мониторах. Они были переведены на объектно-ориентированные языки программирования. В настоящее время на смену технологии CORBA пришли стандартизованные протоколы WEB-сервисов, такие как XML, WSDL, SOAP и др.

Глава 5. Объектное моделирование и программирование

В начальный период развития компьютерной техники, когда мощность компьютеров была недостаточно высокой, к исследованию модельного мира применялись естественные подходы к разработке программных систем, которые имели реляционный (процедурный, функциональный и другие) характер. Их сущность состоит в систематическом использовании декомпозиции функций или процессов для описания и построения структурных моделей. При этом сами моделируемые объекты представлялись фрагментарно в том объеме, который необходим для выполнения этих функций, и в форме, удобной для реализации этих функций. Это

обеспечивает эффективную реализация требуемых функций, однако не создает цельного и адекватного представления моделируемого пространства. В конечном счете и пользователю, и разработчику ПО необходимо наблюдение за изменением состояний объектов модельного мира в результате их взаимодействий, что требует использования комплексных информационных моделей таких объектов и создания программных средств, предоставляющих пользователю доступ к ним как на уровне свойств, так и на уровне поведения.

При объектно-ориентированном подходе структурный анализ требований к системе сводится к разработке ее модели, т. е. формальному описанию, в котором выделены основные объекты, составляющие систему, и отношения между ними.

5.1 Объективный подход и объектная декомпозиция

Объектный подход ориентирован на описание объектов автоматизации с построением их комплексных информационно-поведенческих моделей. При этом многие процессы разработки ПС приобретают специфические объектные черты, такие как:

− использование новой системы понятий, позволяющих описывать объекты и их классы

− декомпозиция объектов, являющаяся основным средством упрощения программной системы

− использование внепрограммных абстракций для упрощения процессов разработки; − предпочтение (приоритет) разработки структуры данных перед реализацией функций.

В объектной модели отношение между объектами обобщается в отношения между этими классами. В самом общем смысле объект

– это сущность, идентифицирующая реальный предмет моделирования, а класс представляет собой описание множества однотипных объектов. При этом каждый объект обладает:

− идентичностью, т. е. его можно поименовать и отличать от иных объектов; − состоянием, характеризуемым некоторой совокупностью данных; − поведением, показывающим, что с ним можно делать или что он сам может делать с другими объектами.

5.2 Язык объектного моделирование UML

Язык UML представляет собой универсальный язык визуального моделирования, разработанный для спецификации, визуализации, системного проектирования и документирования отдельных компонент ПО и программных систем на основе объектного подхода. Он эффективен при построении концептуальных и логических визуальных моделей сложных программных систем разного назначения. Поддержкой языка как открытого стандарта занимается консорциум OMG. Последней версией, опубликованной в июне 2015 года, является UML 2.5. В нем реализованы основные базовые принципы, которые могут быть применены большинством программистов и разработчиков, использующих методы объектно-ориентированного анализа и программирования. Эти базовые понятия могут комбинироваться и расширяться для того, что разработчики объектных моделей имели возможность самостоятельно разрабатывать модели сложных программных систем. Эффективное использование языка UML базируется на понимании общих принципов моделирования сложных систем и использовании специфики проведения объектно-ориентированного анализа и программирования. В частности, одним из базовых принципов объектного моделирования систем является принцип абстрагирования. Он предусматривает включение в создаваемую модель только те аспекты программной системы, которые имеют непосредственное отношение к выполнению системой своего назначения и состава выполняемых функций. Второстепенные детали при этом не учитываются для того, чтобы не усложнять процесс исследования формируемой модели.

В объектном моделировании на языке UML используется и такой важный принцип прикладного системного анализа, как принцип иерар60 хичности формирования моделей сложных систем. Данный принцип предусматривает представление процесса моделирования на разных уровнях абстрагирования или детализации в рамках фиксированных представлений. Исходная модель при этом имеет наиболее общее мета представление, которое формируется в начале этапа проектирования и не содержит несущественных деталей и аспектов моделируемой системы.

UML-диаграмма – это специализированный язык графического описания, предназначенный для объектного моделирования в сфере разработки различного программного обеспечения. Данный язык имеет широкий 61 профиль и представляет собой открытый стандарт, в котором используются различные графические обозначения, чтобы создать абстрактную модель системы. UML создавался для того, чтобы обеспечить определение, визуализацию, документирование, а также проектирование всевозможных программных систем.

Всего в языке UML определены девять типов диаграмм.

Диаграммы классов (class diagram), которые предназначены для отображения классов, интерфейсов, объектов и их кооперацій, связанных различными отношениями. При моделировании систем на основе объектно-ориентированного подхода этот тип диаграмм применяется наиболее часто. Он соответствует статическому представлению системы с точки зрения проектирования.

Диаграммы объектов (object diagram) используются для отображения объектов и связей между ними. Объекты представляют собой статические копии экземпляров сущностей, показываемых на диаграммах классов. Так же как и диаграммы классов диаграммы объектов относятся к статическому отображению системы с точки зрения проектирования или процессов, но рассчитаны на программную или макетную реализацию. Диаграммы прецедентов (use case diagram) представляют прецеденты (варианты использования) и актеров, участвующих в их выполнении, также связанных между собой отношениями. Этот тип диаграмм относится к статическому виду системы и дает точку зрения вариантов ее использования или процессов.

Диаграммы прецедентов особенно важны при концептуальном моделировании системы. Диаграммы взаимодействия предназначены для представления способов и характера информационного взаимодействия между объектами и показывают, в частности, сообщения, с помощью которых объекты обмениваются данными. Используются два частных случая этих диаграмм:

диаграммы последовательностей (sequence diagram), отражающие временной порядок обмена сообщениями, и диаграммы кооперации (collaboration diagram), на которых отображается структурная модель объектов, обменивающихся сообщениями. Эти типы диаграмм являются изоморфными, т. е. они могут свободно трансформироваться друг в друга.

Диаграммы состояний (statechart diagram) отображают автомат, включающий в свою структуру состояния, переходы, события и виды опе62 раций. Эти диаграммы относятся к динамическому представлению программной системы, особенно они необходимы при моделировании изменения состояний интерфейсов, классов или кооперацій. Поведение объекта в диаграмме состояний определяется последовательностью событий, что важно для моделирования реактивных систем.

Диаграммы деятельности (activity diagram) являются частным случаем диаграмм состояний. На них представляется поток управления путем переходов от одной деятельности к другой. Этот вид диаграмм относится к динамическим видам системы и является незаменимым при моделировании процессов ее функционирования, так как описывает поток управления между объектами.

Диаграммы компонентов (component diagram) используются для представления организации связанной совокупности программных компонентов. Они относятся к

статистическому представлению системы с точки зрения реализации, могут быть связаны с диаграммами классов в силу следующего обстоятельства: один компонент обычно включает реализацию одного или нескольких классов, интерфейсов или кооперaций.

Диаграммы развертывания (deployment diagram) представляют конфигурацию узлов программной системы с указанием размещенных в них компонентов. Эти диаграммы относятся к статическому представлению системы и представляют точку зрения развертывания. Они тесно связаны с диаграммами компонентов, так как на узле размещается один или несколько компонентов.

5.3 Объектно-ориентированное программирование

Объектно-ориентированное программирование (ООП) представляет собой метод программирования, основанный на использовании объектов в качестве основных элементов программ. В объектно-ориентированных языках программирования понятие объекта реализовано в виде интегрированной совокупности свойств (данных, характеризующих данный объект), операций по их обработке (функций изменения свойств), а также событий, на которые данный объект может реагировать. Основным фундаментальным понятием ООП является класс, который представляет собой типизированный шаблон, на основе которого может быть создан конкретный исполняемый экземпляр программного объекта. Класс объединяет свойства и методы, определяя сущность и поведение объектов. Объединение данных и процедур обработки в одном объекте с использованием элементов защиты от внешнего использования получило название инкапсуляции и является одним из основополагающих принципов ООП. Кроме того, к важнейшим принципами ООП относятся наследование, полиморфизм и модульность.

Под наследованием понимают такой способ организации классов, при котором новый класс может быть создан на базе существующих, причем класс-потомок сохраняет (наследует) все свойства базового класса – родителя.

Полиморфизм – это свойство, которое означает, что наследуемые объекты содержат информацию о том, какие методы они должны использовать в случае их программного вызова в ситуации неизменности имен элементов класса при наследовании. Иными словами, это концепция, реализующая «множество методов в одном интерфейсе» в зависимости от того, в каком месте дерева иерархии классов они находятся

К основным современным языкам объектно-ориентированного программирования относятся C++, C#, Object Pascal, Java, Ruby, Python и др. С середины 90-х годов прошлого века объектно-ориентированные языки включаются в системы визуального программирования с применением технологии RAD. В такой системе разработки приложений интерфейсная часть программы создается в диалоговом режиме, практически без непосредственного написания программных операторов. К наиболее распространенным и популярным инструментальным системам визуального проектирования относятся: − Embarcadero RAD Studio – среда быстрой разработки приложений для Microsoft Windows компании Embarcadero Technologies., которая объединяет популярные системы программирования Delphi и C++ Builder в единую интегрированную среду.

− Microsoft Visual Studio – среда разработки для всех платформ, поддерживаемых Windows: Windows Mobile, Windows CE, Xbox, .NET Framework, Windows Phone .NET Compact Framework и Silverlight, включающая такие компоненты поддержки различных языков, как Visual Basic.NET, Visual C++, Visual C#, Visual F#, Microsoft SQL Server

− Eclipse – свободная интегрированная среда разработки модульных кросс-платформенных приложений на языке программирования Java производства компании Eclipse Foundation, которая включает в настоящее время и расширения для поддержки других языков, таких как C/C++, Fortran, Perl, PHP, Python, Ruby, 1C V8.

Глава 6. Обеспечение качества программных продуктов

Качество программных систем – это совокупность его черт и характеристик, которые влияют на их способность удовлетворять заданные потребности пользователя. Это не значит, что программные средства можно однозначно оценить некоторым фиксированным набором свойств с высокими значениями показателей качества. Во-первых, потому что показатели качества являются противоречивыми, т. е. улучшение одних показателей качества может приводить к ухудшению других. А во-вторых, ПО можно считать качественным тогда, когда гарантируется успешное его использование в течение заданного срока эксплуатации, т. е. качество необходимо оценивать комплексно, хотя отдельные показатели могут считаться неудовлетворительными.

Системный подход в процессах управления качеством поддержан рядом специализированных инструментальных средств, ориентированных на управление производством программной продукции.

Применение этого комплекса может служить основой для систем обеспечения качества программных средств, однако требуется корректировка, адаптация или исключение некоторых положений стандартов применительно к принципиальным особенностям технологий и характеристик этого вида продукции. Кроме того, при реализации систем качества ПО необходимо привлечение ряда стандартов, формально не относящихся к этой серии и регламентирующих показатели качества, жизненный цикл, верификацию и тестирование, испытания, документирование и другие особенности жизненного цикла программ.

Для управления качеством необходим постоянный контроль качества разрабатываемого ПО через метрики качества, такие как плотность дефектов, размер переделок, среднее время между отказами и др., а также контроль качества отдельных стадий процессов проектирования, составляющих целостный процесс создания программной системы

У каждого программного проекта есть своя специфика, требующая и различного набора методов обеспечения качества.

6.1 Проблемы надежности и качества программных систем

В настоящее время ни разработчики программных продуктов, ни пользователи не могут абсолютно обоснованно сравнивать качество существующих программных продуктов по экономическим критериям и оценивать, насколько один программный продукт эффективнее другого в эксплуатационных расходах, производительности, затратах живого труда и машинного времени.

Можно выделить следующие существующие проблемы в разработке программного обеспечения:

− существующее в ряде случаев несоответствие процессов разработки ПО международным стандартам;

− наличие ошибок в CASE-инструментах, используемых для разработки программных продуктов;

− сжатые сроки выполнения проекта;

− недостаточно опытные разработчики;

− плохая организация процессов разработки ПО;

− недопонимание той функциональности программы, которую желает видеть заказчик.

Известно большое количество показателей, используемых для характеристики качества программных систем, а также сравнительной оценки их потребительской ценности.

функциональная полнота

− завершенность разработки

− быстродействие как затраты времени на решение задачи пользователя

− уровень требований к комплексу технических средств (занимаемый объем памяти, быстродействие процессора, свободное пространство на диске

− степень и простота настройки на техническую среду, сетевые и периферийные устройства

− стоимость установки, обучения и обслуживания;

− комплексность решения задачи;

− возможность перенастройки на новые условия применения, например, в связи с изменением законодательства, реструктуризацией фирмы и т. д.;

− возможность работы в сети;

− качество помощи пользователю в процессе работы, например наличие ситуативной, контекстно-зависимой и гипертекстовой помощи с оглавлением;

− требования к уровню квалификации пользователя;

− трудоемкость освоения и внедрения; − качество пользовательского интерфейса.

6.2 Тестирование и рефакторинг программного кода

Тестирование кода позволяет на протяжении всего жизненного цикла программной системы гарантировать соответствие всех атрибутов программных проектов заданным параметрам качества. Основная цель тестирования состоит в определении отклонений при реализации функциональных требований, степени их значимости, обнаружении ошибок в ходе выполнения программ и своевременном их исправлении. На протяжении всего процесса разработки ПО необходимо применять комплексные методы проверки качества кода, включающие различные типы тестирования. При этом нужно гарантировать соответствие заданным показателям качества всех промежуточных версий проектируемой системы. Применяются как автоматические, так и ручные тесты. Основными видами тестирования ПО являются следующие.

1. Модульное тестирование (тестирование программных модулей) используется для проверки правильности функционирования функций, подпрограмм и методов классов ПО. Модульные тесты создаются и реализуются проектировщиками непосредственно в процессе написания кода. Как правило, модульное тестирование применяется как для проверки самого кода приложения, так и для проверки функционирования БД.

2.Исследовательское тестирование предназначается для проверки качества кода в случае, когда тестировщик пытается интуитивно исследовать возможности программной системы и обнаружить, и зафиксировать неизвестные ошибки, не имея заранее определенных тестовых сценариев.

3. Интеграционное тестирование применяется для проверки правильности совместного взаимодействия компонентов программного продукта.

4.Функциональное тестирование предназначено для проверки конкретных функциональных требований к ПО и осуществляется в случае добавления к системе новых функций.

5. Нагрузочное тестирование используется для проверки работоспособности программной системы при максимальной (предельной) входной нагрузке.

6. Регрессионное тестирование применяется в случае внесения изменений в ПО для того, чтобы проверить корректность работы тех компо69 нентов системы, которые потенциально могут взаимодействовать с измененным компонентом.

7. Комплексное тестирование используется для проверки полноты всех функциональных и нефункциональных требований программной системы.

8. Приемочное тестирование является завершающим функциональным испытанием, которое должны подтвердить, что созданная программная система соответствует требованиям и ожиданиям пользователей и заказчиков. Приемочные тесты пишутся независимыми специалистами по контролю качества и тестировщиками.

Современные инструментальные среды разработки приложений, такие как MS VisualStudio последних версий, предоставляют разработчикам программного обеспечения возможности создавать модульные и нагрузочные тесты, а также тесты для проверки пользовательского интерфейса.

Для этого используются шаблоны тестовых проектов, к которым можно отнести следующие:

− макет модульного теста, который позволяет создавать тесты проверки модулей в процессе проектирования;

− макет с WEB-тестами производительности и нагрузочными тестами; − макет с кодированными тестами пользовательского интерфейса.

6.3 Качество программного обеспечение ISO 9126

Качество программного обеспечения - это степень, в которой ПО обладает требуемой комбинацией свойств . Качество программного обеспечения - это совокупность характеристик ПО, относящихся к его способности удовлетворять установленные и предполагаемые потребности.

Стандарт ISO 9126 состоит из четырех частей, в которых излагаются следующие категории:

− модель качества;

− внешние метрики;

− внутренние метрики;

− метрики качества в использовании.

Модель качества классифицирует качество ПО с шестью структурными наборами характеристик – показателей качества ПО. Эти показатели, в свою очередь, детализируются атрибутами (субхарактеристиками), как показано на рис. 8.

Критерии качества программного обеспечения

8. корректность
9. устойчивость
10. расширяемость
11. повторное использование
12. совместимость, эффективность
13. переносимость
14. простота использования
15. Функциональность

Глава 7. Документирование программных систем

7.1 Документация в АСОИУ

При создании сложной программной системы требуется серьезная организация процесса проектирования, неотъемлемой частью которой является создание программной документации, которая подвержена постоянным изменениям, вносимым различными специалистами. Проектирование и эксплуатация программных систем сопровождается документированием объектов и процессов их жизненного цикла для обеспечения возможности успешного функционирования и развития функций программных средств и баз данных на любых этапах проекта, для обеспечения удобного интерфейса между разработчиками и пользователями.

Программная документация должна быть адекватна формальным требованиям, актуальному состоянию текстов, диаграмм и объектных кодов программ. Она должна проверяться, контролироваться и удостоверяться (подписываться) ответственными руководителями и заказчиками проекта. Следует иметь в виду, что ошибки и дефекты документов не менее опасны для использования программной системы, чем ошибки в структуре, интерфейсах, файлах текстов программ и в содержании данных. Поэтому к структуре, полноте, правильности и качеству программной документации нужно относиться не менее тщательно, чем к разработке и модификациям проектных схем, текстов программ и описаниям данных.

Под процессом документирования понимается совокупность действий по записи информации, произведенной при реализации процессов жизненного цикла.

При разработке ПС создается и используется большой объем разнообразной документации. Она необходима как средство передачи информации между разработчиками ПС, как средство управления разработкой ПС и как средство передачи пользователям информации, необходимой для применения и

сопровождения ПС. На создание этой документации приходится большая доля стоимости ПС.

Эту документацию можно разбить на две группы:

1. Документы управления разработкой ПС.
2. Документы, входящие в состав ПС.

7.2 Требования к программной документации

Документация является органически составной частью программной системы и требует определенных ресурсов для ее создания и эффективного применения. Описание технического проекта, исходный код и другие программные документы в совокупности должны полностью соответствовать структуре и содержанию программной системы и быть достаточными для ее освоения, применения и изменения. В связи с этим программные документы должны быть:

− корректными и точными в изложении

− строго адекватными функционированию программ и содержанию БД

− изложенными систематически, структурированно, технически грамотно и понятно;
− доступными и удобными для использования специалистами и пользователями различной квалификации, рангов и назначения.

Важной задачей документирования является связывание объективными экономическими категориями взаимодействия разных специалистов в типовой производственной цепочке: заказчик – разработчик – программист – пользователь документации. Поэтому программный продукт как продукт потребления, а также его документация должны быть связаны со всеми процессами взаимодействия в этой цепочке совокупностью экономических и технических характеристик, использующих в той или иной степени основные экономические показатели, такие как реальные затраты материальных и финансовых ресурсов, труда и времени специалистов.

Эти процессы должны поддерживаться CASE-средствами документирования, адекватными тем инструментальным средам, которые применяются для непосредственной разработки комплекса программ и данных. В них должна быть включена поддержка оперативного управления и контроля процессами создания документов, регистрации и утверждения версии.

Для выполнения оценки и обоснования спецификаций требований комплекса программных документов нужны такие исходные данные, как:

− обобщенные характеристики использованных ресурсов;

− технико-экономические показатели завершенных разработок (прототипов);

7.3 Стандартизация программной документации

Общие требования к составу и содержанию документов, поддерживающих разработку программной системы, представлены в ряде стандартов разного уровня, а также в корпоративных инструкциях и публикациях по управлению проектами. Состав и формы программных документов могут широко варьироваться в зависимости от класса и характеристик про ектируемой системы, а также от используемых при этом методов и технологий.

Руководители и администраторы программных проектов должны осуществлять организацию работ по документированию и поддержке этих работ, для чего необходимо управление и стимулирование персонала при проведении документирования, а также обеспечение его требуемыми ресурсами. В процессе проектирования административно-управленческому персоналу необходимо оценивать ход работы, анализировать возникающие проблемы и обеспечивать развитие процесса документирования.

Технологическая документация, описывающая процесс разработки, определяет требования, которым должна удовлетворять проектируемая система, определяет то, как проект контролируют и обеспечивают качество. Она должна включать подробное техническое описание программной системы, в том числе спецификацию программной логики, взаимосвязей, форматов хранения данных. Технологическая документация представляет собой средство связи между всеми лицами, вовлеченными в процесс разработки, так как описывает подробности решений, принимаемых в требованиях к проекту, программированию и тестированию, а также обязанности группы разработки, поясняя кто, что и когда делает с учетом роли объекта работ, документации и каждого специалиста в процессе разработки. Эта документация образует основу сопровождения и описывает в хронологическом порядке историю разработки программной системы. Если технологические документы разработки отсутствуют, неполны или устарели, то руководители и менеджеры теряют важное средство для мониторинга состояния проекта.

Эксплуатационная документация обеспечивает информацию, которая необходима для процессов эксплуатации, сопровождения, модернизации, преобразования и передачи программного изделия пользователю. Она:

− включает необходимую учебную и справочную информацию, сведения для специалистов, использующих или эксплуатирующих программную продукцию;

− облегчает системным администраторам, не разрабатывавшим программную систему, ее сопровождение и модернизацию;

− способствует продаже или приемке программной продукции. Эксплуатационная документация должна содержать следующие инструктивные материалы:

− руководства для пользователей, выполняющих ввод данных, восстановление информации и решение служебных задач с помощью программного комплекса;

− руководства для операторов, которые применяют программный комплекс в АСОИУ;

− руководства для сопровождающих программистов и системных администраторов;
− руководства для руководителей, которые следят за использованием комплекса программ. Типовыми эксплуатационными документами являются:

− учебные и справочные материалы;

− руководства по сопровождению программного комплекса;

− брошюры и информационные листовки, посвященные рекламе программного продукта. Документация управления проектом содержит:

− графики выполнения работ для каждой стадии процесса разработки, а также отчеты об их изменениях;

− отчеты о согласованиях при внесении изменений в программы;

− отчеты о решениях, связанных с ходом проектирования;

− распределение обязанностей специалистов.

Руководители и менеджеры программных проектов обязаны применять стандарты качества ПО в соответствии с различными типами документов и проектов, а также устанавливать, как и когда будут достигнуты и поддержаны нужные показатели качества. Под качеством документации понимается: качество содержания, структура представляемой информации, использование иллюстраций.

В частности, стандартом устанавливается, что результатом успешного осуществления процесса менеджмента программной документации являются:

− разработка стратегии идентификации документации, которая реализуется в течение жизненного цикла программного продукта или услуги;

− определение стандартов, которые необходимо применять при разработке программной документации;

− определение документации, которая производится основным или вспомогательным процессом жизненного цикла ПО;

− рассмотрение и утверждение содержания и целей всей документации программного проекта;

− разработка и обеспечение доступности к документации в соответствии с определенными стандартами;

− сопровождение документации в соответствии с определенными критериями.

Документы жизненного цикла ПО могут иметь различные формы. Например, они могут быть подготовлены как компьютерный файл, хранящийся на магнитных носителях, или как отображение на удаленном терминале. Программная документация может быть оформлена в виде отдельных документов, может объединять несколько документов или быть разделена на несколько документов.

Успешное проектирование программных систем определяется большим количеством факторов, к числу которых относятся: − знание теоретических основ современных методологий и технологий создания программных комплексов; − правильное применение методологических основ выполнения основных видов проектных работ; − практические навыки и опыт работы разработчиков; − способность проектировщиков к творческому и нетривиальному подходу при принятии решений; − грамотная организация управления проектными работами и коллективом разработчиков. Эффект от использования этих факторов возможен только в результате системного видения последствий применения автоматизированных систем обработки информации управления в деятельности предприятий и в социальной сфере. При этом важно уметь анализировать и прогнозировать не только положительные стороны внедрения систем автоматизации в форме экономической выгоды, но и негативные, такие как сокращение потребности в живом труде, уменьшение требований к квалификации и интеллектуальным способностям работников и т. д. Таким образом, создание программных систем имеет многогранный комплексный характер и требует от менеджеров, системных аналитиков, разработчиков архитектуры и информационных баз, программистов и тестировщиков не только знаний технических сторон проектирования, но и

творческих способностей, умения документировать, отстаивать и обосновывать принимаемые решения, моральных и этических качеств.