МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТАШКЕНТСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРЕЗМИЙ

КАФЕДРА «ПРОГРАММНЫЙ ИНЖИНИРИНГ»

УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС

по дисциплине

АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ

Научная сфера: 300000 – Производственно-техническая сфера
Сфера образования: 350 000 – Связь, информатизация и телекоммуникационные технологии
Направление образования: 5330600 – Программный инжиниринг

Нукус -2018

Составила:

Юлдашев К.Р., ассистент кафедры «Программный инжиниринг» НФ ТУИТ имени Мухаммада ал-Харезмий.

Рецензент:

Кожаметов А., доцент кафедры НФ ТУИТ имени Мухаммада ал-Хорезмий.

Рабочая программа утверждена на заседании научно-методического совета факультета Компьютер инжиниринг от «___»_____ 2018., протокол №____.

Председатель научно-методического совета: _____ д.т.н К.Сеитназаров

СОДЕРЖАНИЕ

1.1. Темы лекционных занятий.

1-ЛЕКЦИЯ. АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

План:
1. Архитектура программного обеспечения.
2. Проектирование архитектуры систем предметной области.

Ключевые слова: *Архитектура, инженерия, архитектура программного обеспечения, архитектурное проектирование программного обеспечения, порождающее программирование.*

“Архитектура” и “инженерия”, как виды человеческой деятельности, существовали задолго до появления компьютерных технологий. Прежде всего, эти виды деятельности связывал процесс создания проекта — прототипа, прообраза предполагаемого или возможного объекта. Иными словами проектирование содержит в своем составе понятия “архитектура” и “инженерия”, а проектирование программного обеспечения немногим отличается в этом смысле от проектирования, например, зданий и сооружений. Тенденции развития строительной архитектуры последних десятилетий связаны с максимальной функциональностью проектируемых объектов. Архитектурное проектирование ПО также преследует аналогичную цель.

Согласно энциклопедии «Википедия», архитектура программного обеспечения — это представление системы программного обеспечения, дающее информацию о компонентах составляющих систему, о взаимосвязях между этими компонентами и правилах, регламентирующих эти взаимосвязи, которое предназначено для эффективной разработки проекта такой системы. Проектирование программного обеспечения, в свою очередь, подразумевает выработку свойств системы на основе анализа постановки задачи (моделей предметной области (Domain Design) и требований к ПО), а также опыта проектировщика.

Авторы книги ’’Порождающее программирование: методы, инструменты, применение” К. Чарнецки и У. Айзенекер определяют проектирование

архитектуры ПО как “высокоуровневое проектирование, целью которого является создание гибкой структуры, удовлетворяющей всем основным требованиям и предусматривающей некоторую степень свободы реализации. Как правило, из тех деталей, которые менее других подвержены изменениям, формируется «скелет». При этом все остальные детали делаются как можно более гибкими, с тем, чтобы впоследствии их можно было без труда обновить. Впрочем, изменения иногда вносятся даже в скелет”.

Архитектура ПО — это артефакт, представляющий собой результат процесса разработки программного обеспечения. Элементы архитектуры ПО и модели их соединения предназначены для удовлетворения требований к проектируемым системам. В проекте архитектуры ПО должны быть учтены функциональные и нефункциональные требования к эффективности, выносливости, расширяемости, отказоустойчивости, производительности, возможности повторного использования, а также адаптирования разрабатываемого ПО. Архитектурный проект ПО, позволяет оперативно определить, насколько данный программный продукт соответствует предъявляемым к нему требованиям.

Целью архитектурного проектирования предметной области является следующие артефакты:

- разработка архитектуры множества (семейства) систем, входящих в данную предметную область;
- составление плана реализации модели предметной области;
- реализация модели предметной области.

Проектирование архитектуры систем предметной области

В состав архитектурного проекта ПО входят: описание элементов, из которых состоит данная система, схемы взаимодействий между этими элементами, документация образцов (patterns), на основе которых

осуществляется их компоновка, а также список и содержание ограничений (требований), характерных для этих образцов.

В гражданском и промышленном строительстве языком описания проекта являются архитектурно-строительные чертежи и объемные модели, а также текстовые описания возводимых объектов и технологий их возведения. В качестве иллюстративных средств выражения характеристик ПО, в архитектурном проекте используются различные нотации - блок-схемы (схемы алгоритмов), ER-диаграммы, UML- диаграммы, DFD-диаграммы, а также макеты. Каждая подсистема ПО, состоящая из совокупности ее компонентов и взаимодействий между ними, должна быть детально описана в соответствующей части проекта с использованием этих нотаций. Поскольку такая подсистема может выступать в качестве составного элемента более масштабной системы, в архитектурном проекте ПО обязательно содержится подробное описание укрупнённых частей системы с помощью этих же средств описания проекта.

Относительно употребленного термина “компонент”, Питер Илес (старший разработчик архитектуры информационных технологий, IBM) в статье "Что такое архитектура программного обеспечения?" пишет, что “...большая часть определений архитектуры не определяет термина "компонент", и стандарт IEEE 1471 - не исключение, поскольку намеренно оставляет это понятие неопределенным, чтобы оно соответствовало множеству толкований, возможных в отрасли. Компонент может быть логическим или физическим, технологически- независимым или технологически-связанным, крупно или мелко гранулированным...”.

Проектирование вообще, а также проектирование ПО, является прикладным видом деятельности. Поскольку в любом из вариантов, проектирование - это искусство создания того, чего нет в природе, архитектор (проектировщик) ПО должен овладеть искусством проектирования и самовыражения, позволяющим участникам и заказчикам проекта “строить”

требуемое ПО и управлять этим “строительством” и эксплуатацией дальнейшей эволюцией системы.

2-ЛЕКЦИЯ. СОСТАВЛЕНИЕ ПЛАНА РЕАЛИЗАЦИИ МОДЕЛИ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

План:
1. Составление плана реализации модели предметной области программного обеспечения.
2. Реализация модели предметной области программного обеспечения.

Ключевые слова: *Процессов сборки компонентов, измерения, сопровождения и оптимизация бизнес- процессов, артефакты предметной области.*

Реализация модели предметной области представляет собой архитектурное моделирование проектируемого объекта. План реализации, составленный архитектором ПО, регламентирует способы получения конкретных систем из общей архитектуры и компонентов. Архитектурная часть проекта сборки модели предметной области содержит описания:

- интерфейса заказчика для запуска конкретных подсистем;
- процессов сборки компонентов;
- обработки запросов на изменения и разработку;
- измерений, сопровождения и оптимизации бизнес- процессов.

Данная часть проекта описывает сборку разрабатываемого объекта с вероятным использованием автоматизированных средств для сборки модели. Уровень автоматизации сборки зависит от множества факторов и связан как с программно-технической оснащённостью проекта, так и с наличием достаточного уровня применяемых артефактов предметной области (в том числе с применением повторного кода).

В общем случае возможны следующие варианты сборки моделей проектов:

- Сборка приложений из компонентов производится вручную. Детальный процесс сборки содержится в Руководстве разработчика. В состав проекта также входят описания архитектуры и компонентов, реализации предметно--ориентированных языков, руководства по размещению графических пользовательских интерфейсов;
- Для сборки компонентов применяются разнообразные инструментальные средства. К ним относятся средства поиска и просмотра компонентов, описания применения генераторов для автоматизации определенных аспектов разработки приложений. Детальный процесс сборки содержится в Руководстве разработчика. В состав проекта также входят описания архитектуры и компонентов, реализации предметно-ориентированных языков, руководства по размещению графических пользовательских интерфейсов, генераторов и инфраструктуры, при помощи которой проводится поиск, классификация, распространение компонентов. Документация формируется автоматически;
- Автоматическая сборка модели объекта с применением инструментальных средств для заказчика (средств порождающего программирования), при помощи которых формируется запрос на необходимое ПО. Производство приложения может быть достигнуто одним запросом, если в нём не содержится частей, создаваемых традиционными методами разработки. Детальный процесс сборки также содержится в Руководстве разработчика. В состав проекта также входят описания архитектуры и компонентов, реализации предметно-ориентированных языков, руководства по размещению графических пользовательских интерфейсов, генераторов и инфраструктуры, при помощи которой проводится поиск, классификация, распространение компонентов и тому подобные операции, а также организации процессов производства приложений. Вся документация также формируется автоматически.

Реализация модели предметной области по на данном этапе производится реализация архитектуры, компонентов и плана реализации при помощи методик, содержащихся в проекте.

Вопросы по темам:

1. Что такое архитектура ПО?
2. Какие артефакты являются целью проектирования архитектуры ПО”?
3. Какова роль архитектора при создании ПО?
4. В каком порядке (очередности) выполняются процессы проектирования ПО: проектирование архитектуры систем предметной области, составление плана реализации модели и реализация модели?
5. Назовите состав архитектурной части проекта разработки программного обеспечения.
6. Какова роль архитектора при создании ПО?

3-ЛЕКЦИЯ. ГЕНЕРАТИВНОЕ, ИНТЕНЦИОНАЛЬНОЕ И АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ.

План:

1. Генеративное, интенциональное и автоматное программирование.
2. Проблема повторного использования кода.

Ключевые слова: *Повторного использования кода, метапрограммирования.*

Что должен знать архитектор программного проекта. Немногие из опрошенных профессионалов отвечали на этот вопрос так: «то же, что и хороший программист», «всё», «то же, что и менеджер проекта/директор/технический директор». Интересен также ответ «каждый из нас время от времени работает в роли то архитектора, то технического писателя, а то и программиста одновременно».

Изучение, применение и развитие повторного применения кода выполняется архитектором программного обеспечения. Для выполнения таких работ ему необходимо знать теорию Порождающего программирования.

Проблема повторного использования кода

Каждый программист, выполняя свою работу, производит код для повторного использования и применяет код повторного использования. Любой макрос или библиотека представляют собой ранее произведённый и оптимизированный продукт, применение которого вносит очевидные преимущества в текущую разработку. Если при проектировании следующей версии разработки или при производстве заказа сходного с ранее выполненным заказом в текущий проект вносятся фрагменты ранее произведённого кода, изменённые в соответствии с данным техническим заданием, и в этом случае повторное использование кода также вносит преимущества в новый проект. Надежды разработчика на получение надёжного и устойчивого кода в текущей разработке могут неожиданно омрачаться в связи с выявлением ошибок в ранее произведённом коде и/или с выявлением несоответствия части практически готового кода текущим потребностям заказчика. Формально ранее произведённый код может соответствовать или не соответствовать требованиям текущего проекта. Фактически любая группа разработчиков или отдельных исполнителей применяет на практике ранее разработанный код так же, как это делают мастера в других областях деятельности, имея для каждого вида работ свой набор инструментов. Для программирования мы также применяем множество доступных инструментов, но создаваемый нами код является наиболее ценным и постоянно модифицируемым набором инструментальных программных средств повторного применения.

Проблемы, связанные с повторным применением кода, гораздо интересней и сложней, чем кустарное производство программ отдельными коллективами разработчиков, выполняемое с применением повторного кода собственного производства. Кроме того, использование в проекте библиотек макросов, генераторов, интерпретаторов и других промышленных средств

метапрограммирования, также свидетельствует об индивидуальных квалификациях разработчиков конкретного коллектива. Эти инструменты, а также компиляторы, операционные системы, СУБД и, наконец, компьютеры, а также навыки и опыт по их применению относятся к базовым средствам программирования. Тем не менее, повторное применение кода также является метапрограммированием, близким по сложности к встреченным Первопроходцами программирования «тяготам и лишениям» программистской службы. Кстати, в книге Фредерика П. Брукса «Мифический человеко-месяц, или как создаются программные системы» про метапрограммирование сказано, что «...Это действительно наступление на сущность. Поскольку на среднего программиста информационно-управляющих систем феномен разработки на основе пакетов еще не оказал воздействия, он пока не очень замечаем программной инженерией». Иными словами, если мы не владеем приёмами повторного применения кода, то мы всё ещё вне программной инженерии.

Поскольку программирование, как вид деятельности, контрастно гармонирует между элементарным кодированием разрабатываемого проекта и владением и управлением, наряду с этим, параметрами (характеристиками) высокоуровневых абстрактных представлений о проекте, разработчики также должны уметь применять повторный код других авторов и быть готовыми «оставить» (передать) свой код для своих последователей. Если этого не делать, то будет продолжаться ещё существующая тенденция параллельного программирования одних и тех же бизнес-процессов различными средствами. Наши клиенты будут иметь возможность выбора продуктов, отличающихся реализациями, названиями и ценами и состоящих из одних и тех же функциональных возможностей. Наверное, критерий выбора здесь очевиден - цена разработки и практика тендеров по данному критерию. Всем это практически знакомо и неинтересно.

Поскольку критерием любого продукта является его качество, для производства этого продукта применяется всё лучшее, что доступно при программировании. Вполне допустимо, что для выполнения работ Вашими конкурентами им необходима часть Вашего кода для его повторного применения в их разработках. Для того чтобы повторно применить этот код, Вы должны заранее разрабатывать все части текущих разработок проекта таким образом, что бы каждая из его частей могла являться отдельным товаром - кодом для повторного применения. Тем из Вас, кому приходится сталкиваться со сборками конфигураций и анализировать исходные коды, хорошо известно, что вариантов качества таких кодов всего два. Вариант кода пригодный для применения, как минимум, должен соответствовать облику исходного кода, в котором способен разобраться человек, причём этот человек не является автором этого кода. Если такое определение не действует, скорее всего, качество такого кода соответствует второму варианту, и применение такого кода может привести к программированию части текущего проекта заново.

4-ЛЕКЦИЯ. ГЕНЕРАТИВНОЕ ПРОГРАММИРОВАНИЕ.

План:
1. Понятие о генеративном программировании.

Ключевые слова: *Порождающее программирование, генераторы программного обеспечения.*

При отсутствии однозначной методики по применению ранее разработанного кода, кроме опыта и интуиции разработчиков, имеется наука такого программирования, являющаяся частью программной инженерии. Она называется Порождающее программирование (Generative programming).

Идеи порождающего программирования основываются на проектировании и построении порождающих моделей для семейств систем с целью

генерирования по этим моделям конкретной системы. При наличии практического опыта при разработке семейства систем в определенной предметной области (что характерно для прикладного программирования), появляется возможность разрабатывать многократно используемое ПО.

Разработка многократно используемых компонентов позволяет выявить как общности членов семейства, так и наличие существенных параметров изменчивости. Выявление особенностей предметной области с возможностью моделирования характеристик проекта предоставляют возможность определить, какие характеристики и изменяемые параметры могут быть реализованы сразу, а какие целесообразно запланированы на будущее.

Генеративное программирование предполагает использование генераторов программного кода, которые на основе ранее разработанного кода и с учетом спецификаций требований позволяют автоматизировать сборку ”пилотной” и рабочей версий проекта. Генеративное программирование, являясь инструментарием архитекторов ПО, предназначено для управления логикой проекта от стадии его разработки и на протяжении жизненного цикла программной системы. Для применения генеративного программирования архитектору ПО необходимы CASE-системы, пригодные для их применения в конкретной предметной области, генераторы ПО и ранее разработанный программный код, пригодный для его повторного применения.

5-ЛЕКЦИЯ. ГЕНЕРАТОРЫ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

План:
1. Генераторы программного обеспечения.

Ключевые слова: *генератор, метафункция, метапрограммирования.*

Программная среда, которая осуществляет сборку готового к применению программного продукта, в том числе: программной системы, компонента, класса,

процедуры и тому подобных частей системы на основе высокоуровневой спецификации, называется “генератор”. Применение генераторов позволяет повышать точность и ясность (ментальности) описаний проектируемых систем за счет использования предметно-ориентированных нотаций, реализуемых при помощи генераторов, автоматизировать процесс определения эффективности реализации проектируемой системы и оптимизировать работу с библиотеками компонентов.

Генераторы представляют собой множество различных технологий, включая препроцессоры, метафункции, компиляторы, генерирующие классы и процедуры, генераторы кода CASE-систем (инструментов автоматизированного проектирования и создания программ), трансформационные компоненты и многое другое. Например, при помощи генератора можно автоматически сгенерировать реализацию программы на машинном языке или в виде байтового кода, исходный код которой был написан на высокоуровневом языке программирования. Другим примером генератора является обработчик метафункции в метапрограммировании на основе шаблонов C++ с генерацией классов и процедур.

Большинство CASE-систем содержат генераторы реализаций графических моделей в виде описаний на языке программирования. Некоторые системы метапрограммирования позволяют использовать генераторы, имея в качестве входных спецификаций графически специфицируемые конфигурации компонентов в виде предметно-ориентированных нотаций и фрагменты кода на высокоуровневом языке.

6-ЛЕКЦИЯ. ПРИМЕНЕНИЕ АРХИТЕКТУРНЫХ ОБРАЗЦОВ ДЛЯ ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ.

План:

1. Применение архитектурных образцов для проектирования программного обеспечения.

Ключевые слова: *Образцы (паттерны) проектирования программного обеспечение, уровневый образец, образец каналов и фильтров, образец "классной доски, образец микроядра.*

Несмотря на 30-летний опыт применения термина архитектура программного обеспечения, практическое внедрение архитектурного проектирования ПО продолжается до сих пор и по-прежнему считается новым технологическим направлением в промышленном программировании. Учитывая компоновочный характер построения проектируемых программных систем, в литературе, посвященной архитектурному проектированию, большое внимание отводится разработкам и внедрению образцов (паттернов) проектирования ПО.

В статье-справочнике Ольги Дубиной "Обзор паттернов проектирования" дается такое определение образцов проектирования ПО "...Любой паттерн проектирования, используемый при разработке информационных систем, представляет собой формализованное описание часто встречающейся задачи проектирования, удачное решение данной задачи, а также рекомендации по применению этого решения в различных ситуациях. Кроме того, паттерн проектирования обязательно имеет общеупотребимое наименование. Правильно сформулированный паттерн проектирования позволяет, отыскав однажды удачное решение, пользоваться им снова и снова. Следует подчеркнуть, что важным начальным этапом при работе с паттернами является адекватное моделирование рассматриваемой предметной области. Это является необходимым как для получения должным образом формализованной постановки задачи, так и для выбора подходящих паттернов проектирования.". К. Чарнецки и У. Айзенекер считают, что во многих случаях при архитектурном проектировании крайне полезно проводить периодическую сортировку паттернов проектирования (architectural patterns) с целью получения новых

вариантов сортировки. Полученный таким образом новый архитектурный образец должен соответствовать определенному набору требований, иметь описание в виде документации, состоящей из секций, таких, как имя, контекст, воздействия, решение, следствия и примеры. В качестве иллюстрации К. Чарнецки и У. Айзенекер приводят перечень примеров архитектурных образцов:

- *Уровневый образец.* Сортировка по группам подзадач, каждая из групп находится на определенном уровне абстракции;
- *Образец каналов и фильтров.* Схема обработки потока данных предполагающая, что некоторое количество этапов обработки инкапсулировано в компоненты фильтрации. Данные передаются по каналам между смежными фильтрами, рекомпоновка фильтров позволяет собирать связанные системы или обеспечивать сходное поведение систем;
- *Образец "классной доски".* Схема, в которой осуществляется объединение знаний нескольких специализированных подсистем; что позволяет находить частное или приближенное решение недетерминированной задачи;
- *Образец-посредник.* Схема, в которой разъединенные компоненты взаимодействуют посредством удаленных служб. Необходимо наличие компонента-посредника, обеспечивающего координацию взаимодействия и передачу результатов и исключений;
- *Образец модель—представление—контроллер.* Разложение системы на три компонента: модель с базовыми функциональными возможности и данными, представления для отображения информации пользователю, и контроллеры для обработки данных пользователя. Непротиворечивость данных пользовательского интерфейса и модели обеспечиваются механизмом распространения изменений.
- *Образец микроядра.* Схема, в которой базовое функциональное ядро отделено от функций и деталей, выполняемых по индивидуальным заказам потребителей. Кроме того, микроядро, к которому подключаются эти расширения, организует их взаимодействие.

Михаил Ксензов в статье "Рефакторинг архитектуры программного обеспечения: выделение слоев", исследуя проблему увеличения продолжительности жизненного цикла успешных программных проектов, особо выделяет паттерны архитектурного рефакторинга, которые применяются к компонентам архитектуры. Анализируя роль архитектурных паттернов на примере паттерна выделения слоев, автор работы утверждает, что изменение существующей архитектуры - хороший шаг на пути внедрения новой функциональности, который, к тому же, облегчает дальнейшую эволюцию системы. Концепция слоев, особо выделяемая Михаилом Ксензовым, - это одна из общеупотребительных моделей, используемых разработчиками программного обеспечения для разделения сложных систем на более простые части. В архитектурах компьютерных систем, например, различают слои кода на языке программирования, функций операционной системы, драйверов устройств, наборов инструкций центрального процессора и внутренней логики микросхем. В среде сетевого взаимодействия протокол FTP работает на основе протокола TCP, который, в свою очередь, функционирует "поверх" протокола IP, расположенного "над" протоколом Ethernet и так далее.

На практике архитектура ПО базируется одновременно на нескольких образцах. В различных частях, представлениях и уровнях конкретной архитектуры могут применяться различные образцы. Получаемые таким образом виды архитектурных решений обладают различными эксплуатационными характеристиками. К. Чарнецки и У. Айзенекер считают, что таких видов архитектур ПО всего две.

Это родовая архитектура и архитектура с высокой степенью гибкости.

- *Родовая архитектура.* Её можно описать как несъемный корпус с некоторым количеством гнезд, через которые можно подключать отдельные изменяемые или расширительные компоненты. Интерфейсы этих компонентов и гнезд - другими словами, их ожидания и возможности - должны быть четко

определены. Таким образом, родовая архитектура характеризуется постоянной топологией и фиксированными интерфейсами;

- *Архитектура с высокой степенью гибкости.* В топологии такой архитектуры могут производиться структурные изменения. Путем некоторой настройки из нее можно получить ту или иную родовую архитектуру. «Скелет» такой архитектуры состоит из компонентов, что позволяет по истечении некоторого времени производить ее настройку и модернизацию, в частности, изменять и настраивать интерфейсы. Важной особенностью архитектуры с высокой степенью гибкости, в отличие от родовой архитектуры, является способность учитывать структурную изменчивость предметной области, в состав которой входят разнотипные системы.

Развитие методов объектно-ориентированного программирования повлияло на использование шаблонов метапрограммирования. Использование архитектурных образцов в виде шаблонов метапрограммирования представляют собой практические примеры внедрения генераторов в библиотеки C++. К. Чарнецки и У. Айзенекер считают, что применение шаблонов для областей с высокой производительностью обработки является характерными примерами внедрения генераторов архитектурных образцов.

7-ЛЕКЦИЯ. ИНТЕНЦИАЛЬНОЕ ПРОГРАММИРОВАНИЕ.

План:

1. Интенциальное программирование.

Ключевые слова: *метапрограммирования, система IP, языкоориентированное программирование (Language Oriented Programming), языковой инструментарий(Language Workbench).*

Автоматизация применения порождающего программирования сопровождается исследованиями по созданию специальных систем (сред) метапрограммирования, которые предлагают более широкую поддержку

порождающего программирования. Эти системы поддерживают автоматический рефакторинг и позволяют собрать больше знаний по проектированию, чем при традиционном программировании. Этим обеспечивается более высокий уровень автоматизации рефакторинга или другого вида сопровождения (развития) ПО, поскольку уменьшается количество проектных решений, которые должны предшествовать кодированию.

Применение таких систем позволяет снижать ограничения, связанные с возможностями конкретных языков программирования и позволяют перенастроить существующие приложения на новые платформы с минимизацией затрат за счёт исключения этапов переписывания программ на другие языки программирования.

Ярким и пока единственным примером реализаций таких систем является система IP, разработанная Чарльзом (Каролем) Симони (Charles (Karoly) Simonyi) в период его работы в корпорации Microsoft. Работа в среде метапрограммирования IP (Intentional Programming) иногда называется ментальное или интенциальное программирование. Автор терминов "языкоориентированное программирование" (Language Oriented Programming) и "языковой инструментарий" (Language Workbench), Мартин Фаулер, исследуя и развивая идеи порождающего программирования, пишет, что: благодаря использованию” ... “инструментария современных сред разработки (IDE), этот вид программирования становится гораздо более жизнеспособным. Как бы там ни сложилось в будущем, я уверен, что этот вид приложений на настоящий момент является самым интересным явлением на горизонте нашей индустрии...”.

Данное направление также разрабатывается в компании JetBrains, генеральным директором которой является известный специалист и создатель IntelliJ IDE for Java, Сергей Дмитриев. Компьютерные журналисты называют работы Сергея фабрикой кодогенераторов, особенно в связи с созданием в его компании технологии Meta Programming System (MPS). В интервью Сергея

корреспонденту ресурса http://www.codegeneration.net, посвященного теме автоматизации разработки программного обеспечения, содержится изложение выполняемых им работ, развивающих идеи IP.

8-ЛЕКЦИЯ. АВТОМАТНОЕ ПРОГРАММИРОВАНИЕ.

План:

1. Автоматное программирование.

Ключевые слова: *Инструментальное средство UniMod, SWITCH-технология, UML-нотация*

Наиболее известной Российской разработкой в области автоматизации программной инженерии является метод проектирования и реализации реактивных объектно-ориентированных программ с явным выделением состояний. Для поддержки метода разработано инструментальное средство UniMod. Работы Анатолия Абрамовича Шалыто, создавшего автоматное программирование, связаны с идеей исполняемого языка моделирования UML для генерации ПО. Метод основан на использовании автоматного программирования (SWITCH технологии и UML-нотации. Базирующееся на этом методе средство программирования UniMod, является встраиваемым модулем для платформы Eclipse. В работах авторы отмечают, что в настоящее время язык моделирования UML применяется, в основном, как язык спецификации моделей систем. При этом существующие UML-средства позволяют строить различные диаграммы и автоматически создавать по диаграмме классов «скелет» кода на языках программирования, кроме того, предоставляют возможность автоматически генерировать код поведения программы по диаграммам состояний. Однако известные инструменты не позволяют в полной мере эффективно связывать «скелет» кода с моделью поведения, которую можно описывать с помощью диаграмм состояний, деятельностей, кооперации или последовательностей. Отсутствие

однозначной операционной семантики, по мнению авторов, при традиционном написании программ приводит к различию описания поведения в модели и в программе, а также к произвольной интерпретации программистами поведенческих диаграмм, а описание поведения в модели часто носит неформальный характер. Возможна ситуация, когда формальная модель поведения строится архитектором, а программисты при разработке исходного кода ее не используют, выполняя разработку с применением собственного толкования однозначно описанной модели. Появление операционной семантики в языке UML идентифицирует однозначность понимания диаграмм участниками проектирования ПО и позволит создать исполняемый UML, а генерация кода при этом может, в частности, выполняться непосредственно при интерпретации описанной модели.

Особенность реализованного в инструментальном средстве UniMod метода состоит в том, что проектирование программы выполняется так же, как проводится автоматизация технологических процессов для данной предметной области. При этом строится схема связей, содержащая источники информации, систему управления, объекты управления и обратные связи от этих объектов к системе управления. Система управления реализуется в виде системы взаимодействующих конечных автоматов, каждый из которых является структурным автоматом (автоматом, имеющим несколько входов и выходов). Применяемая SWITCH-технология определяет для каждого автомата два типа диаграмм (схема связей и граф переходов) и их операционную семантику. При наличии нескольких автоматов строится схема их взаимодействия. Для каждого типа диаграмм определяется соответствующая нотация.

Авторы инструментального средства UniMod, сохранив автоматный подход, реализовали UML-нотацию при построении диаграмм в рамках SWITCH-технологии. Архитектурное проектирование производится следующим образом: “...Используя нотацию UML-диаграмм классов, строятся схемы связей

автоматов, которые определяют интерфейс автоматов, а графы переходов строятся с помощью нотации UML-диаграммы состояний. При наличии нескольких автоматов их схема взаимодействия не строится, а все они изображаются на диаграмме классов. Диаграмма классов (как схема связей) и диаграммы состояний образуют предлагаемый графический язык для описания структуры и динамики программ".

При этом проектирование программы осуществляется следующим образом:

- на основе анализа предметной области разрабатывается концептуальная модель системы, определяющая сущности и отношения между ними;
- в отличие от традиционных для объектно-ориентированного программирования подходов, из числа сущностей выделяются источники событий, объекты управления и автоматы. Источники событий активны - они по собственной инициативе воздействуют на автоматы. Объекты управления пассивны - они выполняют действия по командам от автоматов. Объекты управления также могут формировать значения входных переменных для автоматов. Автоматы активируются источниками событий и на основании значений входных переменных и текущих состояний воздействуют на объекты управления, переходя в новые состояния;
- используя нотацию диаграммы классов, строится схема связей автоматов, задающая интерфейс каждого их них. На этой схеме слева отображаются источники событий, в центре - автоматы, а справа - объекты управления. Источники событий с помощью UML-ассоциаций связываются с автоматами, которым они поставляют события. Автоматы связываются с объектами, которыми они управляют, а также с другими автоматами, которые они вызывают или которые вложены в их состояния;

- схема связей, кроме задания интерфейсов автоматов, выполняет функцию, характерную для диаграммы классов - задает объектно-ориентированную структуру программы;
- каждый объект управления содержит два типа методов, реализующих входные переменные (xj) и выходные воздействия (zk);
- для каждого автомата с помощью нотации диаграммы состояний строится граф переходов типа Мура-Мили, в котором дуги могут быть помечены событием (ei), булевой формулой из входных переменных и формируемыми на переходах выходными воздействиями; о в вершинах могут указываться выходные воздействия, выполняемые при входе в состояние и имена вложенных автоматов, которые активны, пока активно состояние, в которое они вложены; кроме вложенности автоматы могут взаимодействовать по вызываемости. При этом вызывающий автомат передает вызываемому событие, что и указывается на переходе или в вершине в виде выходного воздействия. Во втором случае посылка события вызываемому автомату происходит при входе в состояние; о каждый автомат имеет одно начальное и произвольное количество конечных состояний;
- состояния на графе переходов могут быть простыми и сложными. Если в состояние вложено другое состояние, то оно называется сложным. В противном случае состояние простое. Основной особенностью сложных состояний является то, что дуга, исходящая из такого состояния, заменяет однотипные дуги, исходящие из каждого вложенного состояния;
- все сложные состояния неустойчивы, а все простые, за исключением начального - устойчивы. При наличии сложных состояний в автомате, появление события может привести к выполнению более одного перехода. Это происходит в связи с тем, что, как отмечено выше, сложное состояние является неустойчивым и автомат выполняет переходы до тех пор, пока не достигнет первого из простых (устойчивых) состояний. Отметим, что если в графе

переходов сложные состояния отсутствуют, то, как и в SWITCH-технологии, при каждом запуске автомата выполняется не более одного перехода;

- каждая входная переменная и каждое выходное воздействие являются методами соответствующего объекта управления, которые реализуются вручную на целевом языке программирования. Источники событий также реализуются вручную;
- использование символьных обозначений в графах переходов позволяет весьма компактно описывать сложное поведение проектируемых систем. Смысл таких символов задает схема связей. При наведении курсора на соответствующий символ на графе переходов во всплывающей подсказке отображается его текстовое описание.”.

Использование инструментального средства UniMod позволяет спроектировать программу в целом.

В автоматном программировании наиболее трудоемким является процесс проектирования автоматов. Сгенерированный по автоматам код может составлять 70-80% от кода программы в целом. Остальная часть кода разрабатывается традиционным программированием и предназначена для постановщиков событий и объектов управления. Уровень автоматизации при создании автоматных программ может быть резко повышен, если автоматы не строить эвристически, а генерировать на основе генетического программировании.

Вопросы по темам:

1. Дайте определение Generative Programming.
2. Перечислите основные понятия порождающего программирования.
3. В чем отличие от просто генерации кода?
4. Существуют ли проблемы при повторном использовании кода?
5. Какие Вы знаете генераторы кода?
6. Дайте объяснение сокращению IP.
7. Что такое автоматное программирование?
8. В чем заключается подход генетического программирования?

9. Назовите отличия генеративного программирования от компонентно-ориентированного программирования?
10. Что такое повторное использование кода?
11. Что такое паттерн проектирования?
12. Что было создано в результате работ Чарльза Симони?
13. Какое направление развивает Сергей Дмитриев в области разработки ПО?
14. Какое направление развивает Анатолий Шалыто в области разработки ПО?

9-10-ЛЕКЦИИ. АВТОМАТИЗАЦИЯ АРХИТЕКТУРНОГО ПРОЕКТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ. АРХИТЕКТУРА НА БАЗЕ МОДЕЛЕЙ.

План:

1. Архитектура на базе моделей.

Ключевые слова: *Унифицированный язык моделирования, архитектура на базе моделей, платформонезависимой модели (Platform Independent Model, PIM), платформозависимой модели (Platform Specific Model, PSM), механизма хранения объектных метаданных (Meta-Object Facility, MOF).*

Для моделирования архитектуры разрабатываемого программного обеспечения в качестве языка описания проектируемых моделей используют унифицированный язык моделирования - UML (Unified Modeling Language). Графическое представление проекта ПО началось с применения разработанных инженерами компании IBM шаблонами для изображения блок-схем. Обычно UML применяется в качестве ручного инструмента моделирования с использованием простейших автоматизированных средств автоматизации черчения (“электронного кульмана” для получения “чертежей” программного обеспечения) наподобие базовых средств автоматизированного проектирования CAD (Computer Aided Design).

Разработанный Гради Бучем (Grady Booch), Джимом Рэмбо (James Rumbaugh) и Иваром Якобсон (Ivar Hjalmar Jacobson) графический язык UML позволяет архитектору программного обеспечения визуализировать,

документировать, специфицировать и конструировать проекты ПО. Принятый организацией по стандартам Object Management Group (OMG http://www.omg.org/) в качестве стандарта моделирования алгоритмов, язык моделирования UML широко применяется сообществом разработчиков программного обеспечения. Язык UML является формальным языком спецификаций и отличается тем самым от синтаксиса традиционных формально-логических языков и языков программирования. Использование UML связано с последовательностью ведения проектных работ. Сначала проект алгоритма описывается на языке UML. После этого алгоритм вручную переписывается на языке программирования. Полученный результат компилируется в машинный код и подвергается тестированию. Если для такой последовательности применять CASE-средства, большинство из которых имеют встроенные или интегрируемые средства построения UML моделей, то в зависимости от заложенных в этих средах программирования возможностей архитектор получает возможность существенно снижать уровень ручного программирования при формировании макетов проектных решений.

Естественное развитие уровня автоматизации процесса разработки проектов ПО, а также развитие языка UML привели к смене парадигмы и к появлению идей MDA (Model Driven Architecture - "Архитектура на базе моделей"). Обозреватель журнала Computerworld Ян Метлис пишет в статье “Архитектура на базе моделей”: ...”Идея, лежащая в основе MDA, заключается в предельной автоматизации процесса генерации кода, благодаря чему разработчики могут сосредоточиться на создании самого алгоритма”. Далее Ян Метлис отмечает, что: “...OMG перенесла фокус своего внимания с архитектуры Common Object Request Broker Architecture (CORBA) на MDA в 2000 году. Тогда появилось официальное описание, положившее начало процессу классификации и стандартизации, а также создания новой лексики, в том числе основных понятий платформонезависимой модели (Platform Independent Model, PIM),

платформозависимой модели (Platform Specific Model, PSM) и механизма хранения объектных метаданных (Meta-Object Facility, MOF)

Целью деятельности организации OMG является разработка стандартов и спецификаций, регламентирующих применение новых информационных технологий на различных аппаратных и программных платформах. Стандарты и технологии UML, CORBA и MDA, разрабатываемые при участии OMG предлагают интегральный подход к созданию многоплатформенных приложений и обеспечивают возможность взаимодействия между этими приложениями.

Концепцией MDA является описание представления алгоритмов на языке моделирования с последующим автоматическим преобразованием моделей в компьютерный код, причем программирование на базе моделей предполагает, что проектировщики ПО прежде всего создают наиболее подходящую модель, не "привязываясь" к платформе, на которой система будет реализована. Таким образом, при создании модели разработчик ПО полностью абстрагируется от особенностей конкретных программных и аппаратных средств реализации ПО. Следовательно, основным элементом программирования в MDA является платформенно-независимая модель PIM (Platform Independent Model). Формируемая платформеннонезависимая модель создается на языке унифицированного моделирования UML. Перевести замысел в практическую плоскость позволили технологии объектно-ориентированного программирования (ООП), языки UML, XML, MOF и т.д.

Для моделирования архитектуры разрабатываемого программного обеспечения в качестве языка описания проектируемых моделей используют унифицированный язык моделирования - UML (Unified Modeling Language). Графическое представление проекта ПО началось с применения разработанных инженерами компании IBM шаблонами для изображения блок-схем. Обычно UML применяется в качестве ручного инструмента моделирования с

использованием простейших автоматизированных средств автоматизации черчения (“электронного кульмана” для получения “чертежей” программного обеспечения) наподобие базовых средств автоматизированного проектирования CAD (Computer Aided Design).

Разработанный Гради Бучем (Grady Booch), Джимом Рэмбо (James Rumbaugh) и Иваром Якобсон (Ivar Hjalmar Jacobson) графический язык UML позволяет архитектору программного обеспечения визуализировать, документировать, специфицировать и конструировать проекты ПО. Принятый организацией по стандартам Object Management Group (OMG http://www.omg.org/) в качестве стандарта моделирования алгоритмов, язык моделирования UML широко применяется сообществом разработчиков программного обеспечения. Язык UML является формальным языком спецификаций и отличается тем самым от синтаксиса традиционных формально-логических языков и языков программирования. Использование UML связано с последовательностью ведения проектных работ. Сначала проект алгоритма описывается на языке UML. После этого алгоритм В соответствии с идеей технологии MDA первоначально выделяется разработка бизнес-логики функционирования приложения. Создаваемая модель приложения определяет поведение, состав и структуру проектируемого программного продукта.

На следующем этапе, после создания модели PIM, создаются одна или несколько платформенно-зависимых моделей, так называемые PSM (Platform Specific Model), назначение которых обеспечение интеграции PIM с одной или несколькими технологиями разработки программных продуктов. На этом же этапе создаются программные интерфейсы для взаимодействия данного приложения с другими.

На заключительном этапе, на основании PIM и PSM, генерируется код приложения и, при необходимости, база данных. Для нескольких PSM генерация проводится несколько раз - для каждой из используемых платформ. Генерация

кода и баз данных при этом осуществляется автоматически, с использованием специализированных инструментальных программных средств. Важным преимуществом концепции MDA является то, что при разработке приложений основные усилия разработчиков ПО переносятся с этапа программирования на этап создания модели ПО. Кроме того, создав модель один раз, разработчик получают принципиальную возможность генерации приложений для разных аппаратных и программных платформ. Преимущества, которые MDA предоставляет разработчикам ПО, очевидны: локализация в модели всей логики приложения и автоматическая генерация кода и баз данных.

MDA не является конкурентным подходом в сравнении какой- либо из существующих технологий создания ПО (CORBA, J2EE, Sun ONE и .NET). MDA находится на более высоком уровне обобщения процесса разработки, позволяя на этапе создания PIM-модели абстрагироваться от этих платформ, на следующем этапе выбрать одну или несколько платформ разработки и создать соответствующий набор PSM-моделей и, наконец, на этапе генерации кода получить приложение, функционирующее на этих платформах. Разработчики из OMG относятся к MDA не только как к новой технологии, а считают MDA «метатехнологией» создания ПО, которая уже «заранее интегрировала» в себя будущие средства разработки программного обеспечения. Сценарий создания приложений по технологии MDA полностью соответствует технологии генеративного программирования: создается модель, которая поступает на вход специальной программы, а на выходе генерируются готовое приложение и база данных. Изменения, связанные с модификацией разработки также вносятся в модель, и затем процедура генерации повторяется, причем без внесения изменений в код приложения. Из этого также следует, что само понятие «разработчик программного обеспечения» позволит специалистам предметной области наиболее плодотворно участвовать в таком программировании.

В основе архитектуры MDA содержится идея о полном разделении этапов общего проектирования (моделирования) и последующей реализации приложения на конкретной программной платформе. Первоначально при помощи специальных средств проектирования создается общая и независимая от способов реализации модель приложения, а затем осуществляется реализация программы, в какой- либо среде разработки. При этом процесс разработки полностью основан на модели, которая должна содержать всю необходимую для программирования информацию.

Такой подход позволяет теоретически обеспечить:

- Независимость модели от средств разработки, которая позволяет реализовать модель на любой программной платформе.
- Реализованное в архитектуре MDA программное обеспечение, может быть перенесено из одной операционной системы в другую.
- Экономия ресурсов при реализации ПО для нескольких программных платформ одновременно.

Технология MDA практически позволяет автоматизировать процесс программирования. Реализация ПО в соответствии с технологией MDA позволяет автоматизировать создание тех типовых частей приложения, разработка которых поддается автоматизации, так, например создание пользовательского интерфейса, программирование типовых операций, создание базы данных и организация доступа к данным.

Автор книги «Delphi и Model Driven Architecture. Разработка приложений баз данных" [15] и сайта http://www.mda- delphi.com/index.php?lng=ru Константин Грибачев пишет: “Циклограмма создания MDA-приложений также содержит потенциальную возможность итерационной разработки. Однако в этом случае разработчик возвращается на этап I и при необходимости корректирует PIM-модель (рис. 3) приложения.

Рисунок 3. Создание MDA-приложений

Поскольку (по крайней мере, такие намерения декларирует концепция MDA) PSM-модель и генератор кода в идеале должны быть полностью отработаны и функционировать «в автоматическом режиме», постольку все изменения PIM должны реализоваться в измененном коде приложения без искажений. Здесь уместно провести некоторую аналогию с прикладной программой и драйверами операционной системы: если прикладная программа использует некий стандартный драйвер, и он штатно функционирует, то при корректном изменении прикладного кода не произойдет никаких неожиданностей в работе программы в целом. Аналогию можно и несколько расширить: если мы заменим имеющийся драйвер на драйвер другого устройства (например, модернизировав в персональном компьютере звуковую карту), а прикладную программу оставим без изменений, то наше приложение должно по-прежнему штатно функционировать, взаимодействуя уже с другим устройством. Таким образом, создав один раз PIM-модель и заменяя потом «драйверы» (PSM), - добьемся функционирования нашего приложения на совершенно разных платформах. Уже из сказанного, очевидно, какие преимущества дает архитектура MDA. К этому можно добавить еще ряд полезных качеств нового подхода.

- Кардинальное повышение производительности разработки. По сути, при использовании MDA-архитектуры вся разработка сводится к корректному формированию PIM-моделей, устраняется этап «ручного» программирования.

- Документированность и легкость сопровождения. PIM-модель в MDA играет роль как проекта, так и основного документа — описания приложения в достаточно компактном виде.
- Централизация логики функционирования. В отличие от традиционного подхода, где логика работы приложения «разбросана» по программному коду, в MDA она сосредоточена в одном месте — в PIM-модели. Приложение изменяет свое поведение при изменении PIM-модели.
- Облегчение доступности и управляемости разработки. С точки зрения заказчика или менеджера наличие платформеннонезависимой PIM-модели резко облегчает понимание проекта в целом и управление разработкой. Это объясняется тем, что PIM-модель «не привязана» к специфическим особенностям сред программирования и по этой причине не содержит сложных или непонятных заказчику/менеджеру элементов и конструкций. UML-диаграммы, представленные в графическом виде, являются достаточно наглядными и по существу не требуют знания программирования или теоретических основ разработки реляционных баз данных...".

Не случайно аббревиатура MDA расшифровывается как ***Model Driven Architecture*** (архитектура, управляемая моделью). MDA это архитектура, описывающая новый способ разработки программного обеспечения. Создание приложений этой архитектуры базируется на разработке модели приложения.

MDA предствляет собой концепцию модельно ориентированного подхода к разработке программного обеспечения. Его суть состоит в построении абстрактной метамодели управления и обмена метаданными (моделями) и задании способов ее трансформации в поддерживаемые технологии программирования (Java, CORBA, XML и т.д.). Создание метамодели определяется технологией моделирования MOF (Meta Object Facility), являющейся частью концепции MDA.

По мнению создателей, архитектура MDA является новой технологией программирования, так как описывает процесс разработки в целом. Новизна MDA заключается в том, что описание процесса разработки в ней выполнено с использованием современных средств представления и позволяет автоматизировать создание приложений.

Процесс разработки ПО состоит из трёх этапов.

На первом этапе разрабатывается вычислительно-независимая модель (CIM). Модель, создаваемую на этом этапе, также называют доменной или бизнес-моделью. Цель данного этапа разработка общих требований к системе, создание общего словаря понятий, описание окружения, в котором система будет функционировать. Сущности, описываемые в модели CIM, должны тщательно анализироваться и отрабатываться. Право на включение в модель должны иметь только те элементы, которые будут использованы и развиты на последующих этапах разработки. Для создания модели CIM на данном этапе желательно иметь описание модели на языке UML.

Модель CIM первого этапа не является необходимой для процесса разработки приложения, и представляет собой общую концепцию системы. В случаях разработки сложных или крупных программных систем этот этап является обязательным.

На втором этапе разрабатывается платформенно-независимая модель (PIM). Если модель CIM не разрабатывалась, модель PIM разрабатывается «с нуля», а при наличии модели CIM модель PIM основываться на CIM. Преобразование модели CIM в модель PIM осуществляется на основе описания на языке UML, созданного на первом этапе. В модель PIM добавляются элементы, описывающие бизнес-логику, общую структуру системы, состав и взаимодействие подсистем, распределение функционала по элементам, общее описание и требования к пользовательскому интерфейсу. На этом этапе

производится включение модели PIM во все автоматизированные среды разработки приложений на основе MDA (рис 4).

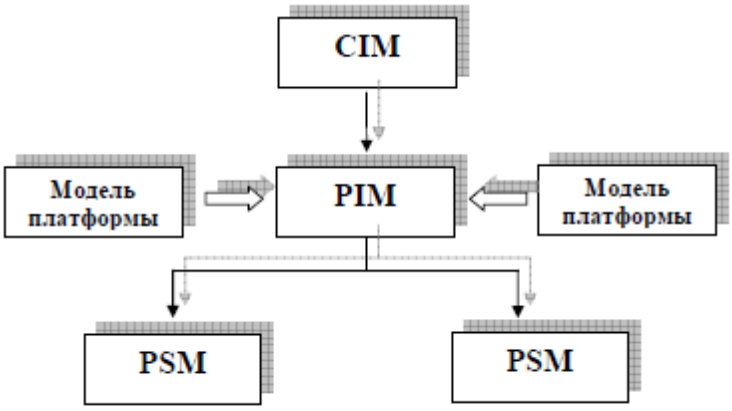


Рисунок 4. Схема взаимодействия

На третьем этапе создаются платформенно-зависимые модели (PSM) путем преобразования модели PIM с учетом требований модели платформы. Количество PSM соответствует количеству программных платформ, для которых разрабатывается ПО. На этапе создания модели PSM разработка приложения завершается.

Модель PSM содержит техническую информацию, достаточную для генерации исходного кода (там, где это возможно) и необходимых ресурсов приложения. Собственно генерация кода ПО выполняется средствами генеративного программирования, не относящимися к компетенции MDA.

Архитектура MDA описывает еще один вариант прохождения третьего этапа, который называется ***прямым преобразованием в код.*** В соответствии со спецификацией, допускается применение инструментальных средств преобразующих модель PIM в исполняемый код приложения. Модель PSM, при этом, может создаваться как контрольное описание, позволяющее проверить результат прямого преобразования.

11-ЛЕКЦИЯ. ПРЕОБРАЗОВАНИЕ МОДЕЛЕЙ PIM PSM.

План:

1. Преобразование моделей PIM PSM.
2. Много платформенные модели.

Ключевые слова: *платформонезависимой модели (Platform Independent Model, PIM), платформозависимой модели (Platform Specific Model, PSM), механизма хранения объектных метаданных (Meta-Object Facility, MOF).*

Наиболее сложным и ответственным этапом при разработке приложений в рамках архитектуры MDA является преобразование модели PIM в модель PSM. Именно на этом этапе общее описание системы на языке UML приобретает вид, пригодный для воплощения приложения на конкретной платформе. Как уже отмечалось выше, в процессе проектирования принимает участие модель платформы. Преобразование моделей проходит три последовательные стадии:

- Разработка схемы преобразования (mapping).
- Маркирование (marking).
- Собственно преобразование (transformation).

Рассмотрим их подробнее. Первоначально необходимо разработать схему преобразования элементов модели PIM в элементы модели PSM. Для каждой платформы создается собственная схема преобразования, которая напрямую зависит от возможностей платформы. Схема преобразования затрагивает как содержание модели (совокупность элементов и их свойства), так и саму модель (метамодель, используемые типы). В схеме преобразования требуемым типам модели, свойствам метамодели, элементам модели PIM ставятся в соответствие типы модели, свойства метамодели, элементы модели PSM. При преобразовании моделей может использоваться несколько схем преобразования. Для связывания используются марки (mark) самостоятельные структуры данных, принадлежащие не моделям, а схемам преобразования и содержащие информацию о созданных связях. Наборы марок могут быть объединены в тематические шаблоны, которые возможно использовать в различных схемах преобразования. Процесс задания марок называется маркированием. В простейшем случае один элемент модели

PIM соединяется маркой с одним элементом модели PSM. В более сложных случаях один элемент модели PIM может иметь несколько марок из разных схем преобразования. Что касается преобразования метамодели, то в большинстве случаев марки могут расставляться автоматически. А вот для элементов модели часто требуется вмешательство разработчика. В процессе маркирования необходимо использовать сведения о платформе. Эти сведения содержатся в модели платформы (рис. 5).

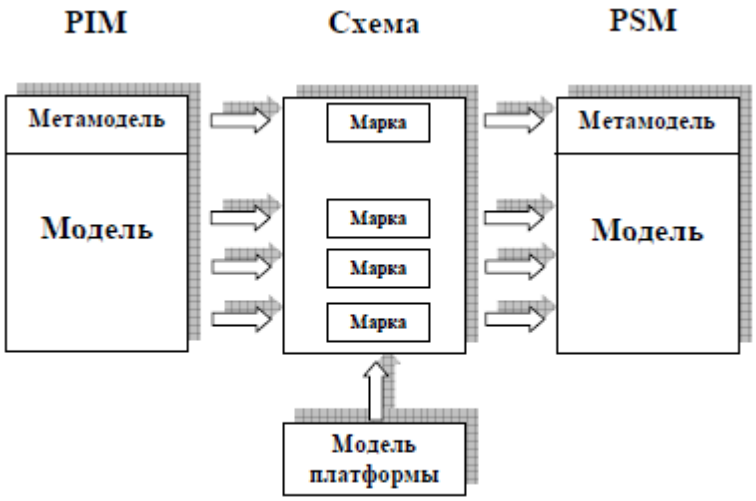


Рисунок 5. Модель платформы

Процесс преобразования моделей заключается в переносе маркированных элементов модели и метамодели PIM в модель и метамодель PSM. Процесс преобразования должен документироваться в виде карты переноса элементов модели и метамодели. Способ преобразования моделей может быть:

- Ручной.
- С использованием профилей.
- С настроенной схемой преобразования.
- Автоматический.

12-ЛЕКЦИЯ. МНОГО ПЛАТФОРМЕННЫЕ МОДЕЛИ.

План:

1. Преобразование моделей PIM PSM.
2. Много платформенные модели.

Ключевые слова: Модель, Платформа, **управление на основе модели, вычислительная независимость, преобразование модели.**

Архитектура MDA учитывает возможность разработки приложений, одновременно функционирующих на нескольких платформах. Для этого марки схемы преобразования моделей PIM PSM устанавливаются в соответствии с распределением средств приложения по платформам. Затем генерируется несколько платформеннозависимых частей приложения. Проблема взаимодействия частей такого гетерогенного приложения решается на уровне бизнес-логики приложения на этапе разработки. Для обмена данными могут использоваться специально разработанные

подсистемы, использующие для организации обмена заранее согласованные механизмы, форматы данных, интерфейсы. Более того, разработка механизмов межплатформенного взаимодействия хорошо поддается автоматизации. Инструментарии MDA могут содержать средства для создания таких механизмов.

Архитектура MDA описывает и структурирует поэтапный процесс разработки любых программных систем на основе создания и использования моделей. При этом используется несколько типов моделей, создаваемых и преобразуемых на различных этапах разработки. Процесс разработки по MDA это последовательное (поэтапное) продвижение от одной модели системы к другой. При этом каждая последующая модель преобразуется из предыдущей и дополняется новыми деталями. Модели, общая схема разработки и процесс преобразования моделей - ключевые составные части архитектуры.

При применении технологии разработки ПО применяются следующие общие термины и определения:

Модель описание или спецификация системы и ее окружения, созданная для определенных целей. Часто является комбинацией текстовой и графической информации. Текст может быть описан специализированным или естественным языком.

Управление на основе модели процесс разработки системы, использующий модель для понимания, конструирования, распространения и других операций.

Платформа набор подсистем и технологий, которые представляют собой единый набор функциональности, используемой любым приложением без уточнения деталей реализации.

Вычислительная независимость - качество модели, обозначающее отсутствие любых деталей структуры и процессов.

Платформенная независимость - качество модели, обозначающее ее независимость от свойств любой платформы.

Вычислительно-независимая модель - модель, скрывающая любые детали реализации и процессов системы; описывает только требования к системе и ее окружению.

Платформенно-независимая модель - модель, скрывающая детали реализации системы, зависимые от платформы, и содержащая элементы, не изменяющиеся при взаимодействии системы с любой платформой.

Платформенно-зависимая модель - модель системы с учетом деталей реализации и процессов, зависимых от конкретной платформы.

Модель платформы набор технических характеристик и описаний технологий и интерфейсов, составляющих платформу.

Преобразование модели - процесс преобразования одной модели системы в другую модель той же системы.

В MDA используются следующие типы моделей:

Вычислительно-независимая модель (Computation Independent Model, CIM) описывает общие требования к системе, словарь используемых понятий и условия функционирования (окружение). Модель не должна содержать никаких сведений технического характера, описаний структуры и свойств системы. CIM максимально общая и независимая от реализации системы модель. Спецификация MDA подчеркивает, что CIM должна быть построена так, чтобы ее можно было преобразовать в платформенно-независимую модель. Поэтому CIM рекомендуется выполнять с использованием унифицированного языка моделирования UML.

Платформенно-независимая модель (Platform Independent Model, PIM) описывает состав, структуру, функционал системы. Модель может содержать сколь угодно подробные сведения, но они не должны касаться вопросов реализации системы на конкретных платформах. Модель PIM создается на основе CIM. Для создания модели используется унифицированный язык моделирования UML.

Платформенно-зависимая модель (Platform Specific Model, PSM) описывает состав, структуру, функционал системы применительно к

вопросам ее реализации на конкретной платформе. В зависимости от назначения модель может быть более или менее детализированной. Модель создается на основе двух моделей. Модель PIM является основой модели PSM. Модель платформы используется для доработки PSM в соответствии с требованиями платформы.

Модель платформы описывает технические характеристики, интерфейсы, функции платформы. Зачастую модель платформы представлена в виде технических описаний и руководств. Модель платформы используется при преобразовании модели PIM в модель PSM. Для целей MDA описание модели

платформы должно быть представлено на унифицированном языке моделирования UML.

В зависимости от уровня детализации платформы, модели (кроме модели платформы) могут содержать сведения о различных функциональных частях системы. В этом случае говорят об уровнях модели. Обычно различают следующие основные уровни модели.

Уровень бизнес-логики содержит описание основного функционала приложения, обеспечивающего исполнение его назначения. Как правило, уровень бизнес-логики хуже всего поддается автоматизации, поэтому столько усилий было направлено на разработку автоматизации проектирования архитектуры. Уровень бизнес-логики составляет львиную долю кода приложения, который приходится писать вручную.

Уровень данных описывает структуру данных приложения, используемые источники, форматы данных, технологии и механизмы доступа к данным. Для приложений .NET чаще всего используются возможности ADO.NET.

Уровень пользовательского интерфейса описывает возможности приложения по взаимодействию с пользователями, а также состав форм приложения, функционал элементов управления (например, контроль ввода данных). Легкость автоматизации этого уровня зависит от того, насколько унифицированы пользовательские операции. Если удается создать типовые шаблоны элементов управления для основных операций, появляется возможность автоматической генерации форм и их.

Вместе с тем, концепция MDA критикуется, поскольку, по меткому замечанию Мартина Фаулера, мода на перспективные технологии программирования постоянно меняется.

Ивар Якобсон (12. 12. 2005): “Сегодня все мы уже фактически ведем разработку на базе моделей. Но то, что предлагает OMG, — сделать модель анализа

формальной, выполняемой и автоматически трансформируемой в модель, отражающую специфику платформы, на самом деле реализовать очень сложно”.

Мартин Фаулер (12. 06. 05): "...Пока не существует никаких стандартов для определения трио "схема, редактор и генератор". Создав язык в каком- либо языковом инструментарии, вы тут же попадаете в зависимость от него. Раз нет никаких стандартных способов обмена данными между разными языковыми инструментария, значит, при переходе на другой языковой инструментарий, придется создавать заново и схему, и редактор, и генератор. Может быть, с течением времени возникнет некий специальный вид хранения данных – специально для таких случаев. Однако если этого не случится, то риск зависимости от поставщика инструментария будет весьма большим.
(Архитектура MDA дает некоторый ответ на эту проблему, но на мой взгляд, он по меньшей мере неполон.)...".
Чарльз Симони: “…MDA is a kitchen-sink standard that is implementation oriented.…” содержимого при создании приложения из модели.

13-14-ЛЕКЦИИ. ПРИМЕНЕНИЕ CASE-ТЕХНОЛОГИЙ.

План:
1. Применение CASE-технологий.

Ключевые слова: *CASE (Computer- Aided Software/System Engineering), методология, метод, нотация, средство.*

Автоматизация архитектурного проектирования программного обеспечения основывается на применении инструментальных программных средств, которые принято называть CASE (Computer- Aided Software/System Engineering). Несмотря на достаточное, на первый взгляд, количество существующих средств автоматизации проектирования архитектуры программного обеспечения, программные архитекторы по- прежнему нуждаются в расширении набора доступных средств автоматизации. По сравнению со

своими коллегами в других областях инженерного творчества, уровень оснащения архитекторов CASE-средствами явно недостаточен и вот почему. Автоматизация структурных методологий, характерных для программирования, и возможность применения современных методов системной и программной инженерии должны позволять CASE-системам:

- улучшать качество создаваемого программного обеспечения за счет средств автоматического контроля проекта;
- создавать за короткое время прототип будущей системы с использованием генераторов программного кода, что позволяет на ранних этапах оценить ожидаемый результат;
- ускорять процесс проектирования, разработки и внедрения;
- позволять разработчику сосредоточиться на творческой части разработки за счет сокращения рутинной работы;
- поддерживать развитие и сопровождение проекта;
- применять технологии повторного использования архитектурных образцов.

Большинство CASE-средств, стремящихся удовлетворять перечисленным требованиям, основано на парадигме программирования – методология→метод→нотация→средство, где:

- методология определяет оценку и выбор проекта разрабатываемого программного обеспечения, последовательность разработки и правила распределения и назначения методов;
- метод представляет собой способ генерации описаний компонентов программного обеспечения;
- нотации это средства описания проектной логики, в том числе: диаграммы, графы, формальные и естественные языки, а также таблицы и блок-схемы, которые предназначены для описания структуры системы, описания элементов данных и назначение этапов обработки;

- средства (инструменты) для поддержки и усиления методов предназначены для участия пользователей при создании и редактировании графического проекта в интерактивном режиме. Средства способствуют организации проекта в виде иерархии уровней абстракции и выполняют роль проверки соответствия компонентов.

Автоматизация проектирования архитектуры программного обеспечения с применением CASE-средств не может основываться на выборе отдельно взятой CASE-системы, поскольку универсальной CASE-системы, отвечающей перечисленным требованиям и достаточной для производства ПО для любой предметной области пока не существует. С появлением новых CASE-систем, а также с развитием генераторов программного кода и с внедрением систем для повторного применения ранее разработанного кода станет возможным создавать на базе предприятий-разработчиков программного обеспечения, системы автоматизированного проектирования программного обеспечения.

Методология и средства анализа и проектирования многокомпонент -ных информационных систем, содержащиеся в большинстве CASE-систем, позволяют применять методологии создания информационных систем с компонентной архитектурой. Значительный вклад в развитие компонентной методологии внесли сотрудники фирмы IBM Rational Software (особенно Г. Буч, Д. Рамбо и И. Якобсон). Анализ и проектирование информационных систем с компонентной архитектурой. основываются на использовании унифицированного языка моделирования UML, и поддерживаются целым спектром инструментальных программных средств визуального моделирования. В CASE-системах поддерживаются основные языки программирования C++, Java, Visual Basic, SmallTalk и т.д., а также популярные среды разработки MS Visual Studio, Delphi, PowerBuilder, средства автоматизированного тестирования и документирования, охватывающих жизненный цикл создания программных систем.

Наиболее известной CASE-системой объектно-ориентированного моделирования является Rational Rose компании IBM Rational Software.

Все продукты Rational Rose поддерживают язык Unified Modeling Language (UML). Тем не менее, эти продукты различаются технологиями реализации, которые они поддерживают.

CASE-система IBM Rational Software – Rational Rose позволяет автоматизировать этапы анализа и проектирования разрабатываемого программного обеспечения, а также предоставляет возможность генерации кода ПО на различных языках программирования для формирования макетов систем и позволяет автоматизировать выпуск проектной документации.

Rational Rose позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также производить генерацию программного кода на различных языках программирования (C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro). В составе инструментальных программных средств CASE-системы Rational Rose, также содержатся средства реинжиниринга программного обеспечения. Такая возможность, предназначена для повторного использование программных компонент в новых проектах.

Методической основой применения CASE-системы Rational Rose является автоматизация процесса построения диаграмм классов, состояний, сценариев, модулей и процессов, а также формализации спецификаций логической и физической структуры модели, и описания статических и динамических аспектов разрабатываемого программного обеспечения.

Уникальность CASE-системы Rational Rose заключается в обеспечении архитектора ПО (проектировщика ПО) достаточными средствами проектирования, в том числе: репозиторий, графический интерфейс, средства просмотра проекта, средства контроля проекта, средства сбора статистики и генератор документов, генератор и анализатор программного кода и средства реинжиниринга.

Репозиторий CASE-системы Rational Rose обеспечивают "навигацию" по проекту (включая перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому, средства контроля и сбора статистики, генератор отчётов и др.) позволяют моделировать проект ПО и сопровождать результат разработки в течение всего жизненного цикла программной системы.

Создаваемый встроенным генератором CASE-системы Rational Rose скелет кода программы на языке программирования C++ предназначается для его доработки традиционным методом прямого программирования на языке C++. При генерации программного кода в CASE-системе Rational Rose используется информация из логической и физической моделей проекта ПО. В результате “прогона” генератора формируются заголовки и описания классов и объектов.

Анализатор исходного кода C++ позволяет создавать модули проектов и осуществляет контроль правильности исходных текстов и диагностику ошибок. Получаемая модель проекта пригодна для её использования в качестве повторно применяемого кода.

CASE-система Rational Rose позволяет формировать такие проектные документы:

- диаграммы классов;
- диаграммы состояний;
- диаграммы сценариев;
- диаграммы модулей;
- диаграммы процессов;
- диаграммы компонентов;
- спецификации классов, объектов, атрибутов и операций;
- заготовки текстов программ,

а также модель разрабатываемой программной системы в текстовом формате (.mdl-файл).

Вопросы по темам:

1. Что такое модельная архитектура MDA?
2. Что такое CIM?
3. Что такое PIM?
4. Что такое PSM?
5. Дайте объяснение понятию управление на основе модели.
6. Чем отличаются платформенно-независимая модель от вычислительно-независимой модели?
7. Что Вы знаете о CASE-технологиях и о CASE-системах?
8. Назовите CASE-систему, использование которой, на Ваш взгляд, является предпочтительным для разработки ПО?
9. Что объединяет CASE-технологии и почему?

15-ЛЕКЦИЯ. КОМПОНЕНТНАЯ АРХИТЕКТУРА.

План:

1. Компонентная архитектура.

Ключевые слова: *Многоплатформенный, технология CORBA, стандартная библиотека шаблонов STL.*

Для понимания возможностей и целей использования хорошо описанной технологии применения компонентной архитектуры ПО, продолжим проведение аналогий между архитектурами в строительном проектировании и в проектировании программного обеспечения.

Основоположник серийного строительства, выдающийся архитектор современности Шарль Ле Корбюзье, оказал существенное влияние на разработки и массовое применение унифицированных строительных блоков. Проектируя здания и сооружения, Корбюзье преследовал цели создания архитектурно-пространственных композиций, относясь к блокам так же, как задолго до него зодчие относились к кирпичам и к другим подобным блокам возводимых объектов строительства. Таким образом, конечной целью творчества мастера являлась архитектура заказного проекта, основанная на генеративных принципах повторного применения блоков, узлов и деталей, являющихся одним из приёмов быстрого возведения объектов строительства.

Жители советских построек конца 50-х и подавляющего большинства возводимых сегодня домов, к сожалению, знают к чему могут привести искажения в массовом применении современных технологий проектирования и реализации. Поэтому архитектурное проектирование является вполне ответственным и личностным ремеслом, в том числе и при разработке программного обеспечения.

Программные компоненты являются строительными блоками, из которых могут быть построены различные системы ПО. Применяя компоненты, необходимо, чтобы они были совместимы при подключении в ходе проектирования и были максимально сочетаемы друг с другом. На самом деле, использование компонент предназначено для минимизации дублирования кода и максимизации повторного применения кода. Эти и другие свойства определяют качество компонентов.

Компоненты в общем смысле представляют собой части конкретного производственного процесса. Мы не можем применять кирпичи для конструирования технических приборов или машин, мы строим кирпичные постройки и конструируем машины в виде систем узлов и механизмов.

Применительно для программной инженерии, мы используем. компоненты стандартной библиотеки шаблонов если требуется контейнер в языке C++. Для проектирования графического интерфейса пользователями применяем визуальные компоненты (например, такие, как JavaBeans). Для создания многоплатформенных реализаций распределенного ПО мы компонуем и генерируем программную систему с применением технологий MDA. Если перед нами стоит задача проектирования языково-независимого ПО, для этой цели применяется технология CORBA.

С библиотеками шаблонов связаны имена их разработчиков, но скорее всего, всё связанное с шаблонами началось с рождения языка C++. Создателем языка C++ (C с классами) является Бьерн Страуструп (Bjarne Stroustrup). Об этом

ученом, и о созданным им языке написано большое количество книг, в том числе и учебников.

Интересующимся техникой программирования на языке C++ и практической работой с библиотеками шаблонов следует воспользоваться литературой, Отметим, что на появление библиотек шаблонов оказало влияние наличие в языке C++ инструкции template <class T>;

Сам Бьерн Страуструп написал про эту начальную инструкцию: “…она отличается от обычного описания класса и показывает, что описывается не класс, а шаблон типа с заданным параметром-типом”...” “Возможности, которые реализует шаблон типа, иногда называются параметрическими типами или генерическими объектами”. С помощью шаблона, можно определить такие контейнерные классы, как списки и ассоциативные массивы и, не отказываясь от статического контроля типов, реализовать без потерь в эффективности выполнения программы.

Шаблоны типа позволяют определить сразу для целого семейства типов обобщенные (генерические) функции, например, такие, как sort (сортировка). В качестве примера шаблона типов и его связи с другими конструкциями языка можно привести семейство списочных классов.

Одним из самых полезных классов является контейнерный класс (такой класс, который хранит объекты каких-то других типов). Списки, массивы, ассоциативные массивы и множества - все это контейнерные классы. Контейнерные классы обладают тем свойством, что тип содержащихся в них объектов не имеет особого значения для создателя контейнера. Но для пользователя конкретного контейнера этот тип является важным. Таким образом, тип содержащихся объектов должен быть параметром контейнерного класса, и создатель такого класса будет определять его с помощью типа-параметра.

Исследования множества применений шаблонов при программировании на языке C++ привели к разработке различных библиотек стандартных шаблонов,

часть из которых (прежде всего, STL) являются стандартами программирования и включены в компиляторы языка C++.

16-ЛЕКЦИЯ. СТАНДАРТНАЯ БИБЛИОТЕКА ШАБЛОНОВ STL.

План:
1. Стандартная библиотека шаблонов STL.
2. Стандартная библиотека шаблонов STL. Примеры

Ключевые слова: *Стандартная Библиотека Шаблонов STL (Standard Template Library), алгоритм, контейнер, итератор, адаптер.*

Создателем стандартной библиотеки шаблонов STL является Александр Александрович Степанов (Alexander Stepanov). В работе приведены примеры использования библиотеки STL, в частности, для генеративного программирования кода с целью исключения повторения одинаковых фрагментов программы с одними и теми же алгоритмами, предназначенными для обработки разных типов данных.

Стандартная Библиотека Шаблонов STL (Standard Template Library) представляет собой набор обобщённых, совместно работающих компонентов языка C++. Шаблонные алгоритмы STL работают как со структурами данных в библиотеке, так и с встроенными структурами данных языка C++.

В качестве примера отметим, что алгоритмы STL работают с обычными указателями. При этом, возможно, как использование структуры данных библиотеки STL с проектируемыми в разрабатываемой программе алгоритмами, так и использование
алгоритмов STL со структурами данных программы. Этому способствуют определённые стандартные семантические требования использования STL, гарантирующие эффективную работу компонента с библиотекой. Такая гибкость обеспечивает широкую применимость библиотеки.
Библиотека STL состоит из пяти основных видов компонентов:

- Алгоритм (Algorithm), который определяет вычислительную процедуру.
- Контейнер (Container), назначение которого управлять набором объектов в памяти.
- Итератор (Iterator), который обеспечивает средство доступа к содержимому контейнера для алгоритма.
- Функциональный объект (Function object), который инкапсулирует функцию в объекте для её использования другими компонентами.
- Адаптер (Adaptor), который настраивает компонент для обеспечения различного интерфейса.

Скотт Мейерс, в своей работе "Эффективное использование STL", определяет терминологию, применяемую в STL, следующим образом:

- ”… Контейнеры vector, string, deque и list относятся к категории стандартных последовательных контейнеров. К категории стандартных ассоциативных контейнеров относятся контейнеры set, multiset, map и multimap.
- Итераторы делятся на пять категорий в соответствии с поддерживаемыми операциями. Итераторы ввода обеспечивают доступ только для чтения и позволяют прочитать каждую позицию только один раз. Итераторы вывода обеспечивают

доступ только для записи и позволяют записать данные в каждую позицию только один раз. Итераторы ввода и вывода построены по образцу операций чтения-записи в потоках ввода-вывода (например, в файлах), поэтому неудивительно, что самыми распространенными представителями итераторов ввода и вывода являются istream_iterator и ostream_iterator, соответственно.

Прямые итераторы обладают свойствами итераторов ввода и вывода, но они позволяют многократно производить чтение или запись в любой позиции. Оператор – ими не поддерживается, поэтому они позволяют производить

передвижение только в прямом направлении с некоторой степенью эффективности. Все стандартные контейнеры STL
поддерживают итераторы, превосходящие эту категорию итераторов по своим возможностям, но, при этом одна из архитектур хешированных контейнеров основана на использовании прямых итераторов.

Контейнеры односвязных списков … также поддерживают прямые итераторы. Двусторонние итераторы похожи на прямые итераторы, однако они позволяют перемещаться не только в прямом, но и в обратном направлении. Они поддерживаются всеми стандартными ассоциативными контейнерами, а также контейнером list.

Итераторы произвольного доступа обладают всеми возможностями двусторонних итераторов, но они также позволяют переходить в прямом или обратном направлении на произвольное расстояние за один шаг. Итераторы произвольного доступа поддерживаются контейнерами vector, string и deque. В массивах функциональность итераторов произвольного доступа обеспечивается указателями.

- Любой класс, перегружающий оператор вызова функции (то есть operator ()), является классом функтора. Объекты, созданные на основе таких классов, называются объектами функций, или функторами. Как правило, в STL объекты функций могут свободно заменяться ≪обычными≫ функциями, поэтому под термином ≪объекты функций≫ часто объединяются как функции C++, так и функторы.
- Функции bindlst и bind2nd называются функциями привязки (binders)…”

Определения Скотта Мейерса раскрывают архитектуру библиотеки STL, а также и способы её применения для выполнения практических разработок. Кстати, книга Скотта Майерса состоит из 50 советов по практическому использованию STL, расположенных в определенной последовательности.

Тексты исходных кодов STL являются открытыми и будут частично воспроизводиться в настоящей работе.

Существует множество реализаций компонентов на базе STL, соответствующих стандарту, которые отличаются друг от друга индивидуальными свойствами. В некоторых источниках, как, например, в интересной статье Олега Ремизова, такие реализации называются коллекциями16 STL.

Разнообразие реализаций (коллекций) на базе STL может представлять определенную проблему для разработчиков архитектур с возможностью использования кода для его повторного применения.

Однако выполнение разработок в рамках стандарта STL минимизирует. риски неверного использования кода повторного применения на базе STL для разработки последующих проектов. Стандартная библиотека STL содержит достаточное количество

компонентов, необходимых для выполнения основных видов работ.

Следуя структурному описанию Олега Ремизова, перечислим некоторые из них:

vector – это множество элементов T, сохраняемых в массиве, размер которого увеличивается по мере необходимости.

Ниже приведён исходный код контейнера vector.

```
#ifndef __SGI_STL_VECTOR_H
#define __SGI_STL_VECTOR_H

#include <stl_range_errors.h>
#include <algobase.h>
#include <alloc.h>
#include <stl_vector.h>

#ifdef __STL_USE_NAMESPACES
using __STD::vector;
#endif /* __STL_USE_NAMESPACES */
#endif /* __SGI_STL_VECTOR_H */
```

Контейнер vector – чаще всего используемая компонента STL. Внутренняя реализация этого контейнера является массивом и имеет счетчик элементов,

сохраненных в этом массиве. Контейнер vector содержит инструкцию operator [], который позволяет пользоваться контейнером как обычным массивом. Такой же прием использования operator []также применен в контейнерах в map, deque, string и wstring.

Для использования контейнера vector необходима инструкция:

#include <vector>;

list - множество элементов T, сохраненных, как двунаправленный связанный список.

Ниже приведён исходный код контейнера list.

```
#ifndef __SGI_STL_LIST_H
#define __SGI_STL_LIST_H

#include <algobase.h>
#include <alloc.h>
#include <stl_list.h>

#ifdef __STL_USE_NAMESPACES
using __STD::list;
#endif /* __STL_USE_NAMESPACES */
#endif /* __SGI_STL_LIST_H */
Для использования контейнера vector необходима инструкция:
#include <list>;
```

map – это множество элементов (коллекция), сохраняющая пары значений pair<const Key, T>. Этот контейнер предназначен для быстрого поиска значения T по ключу const Key. В качестве ключа может быть использовано все, что угодно. Главной особенностью ключа является возможность применения к нему операции сравнения. Быстрый поиск значения по ключу осуществляется за счет отсортированных хранящихся пар. Как пишет Олег Ремизов, этот контейнер ”…имеет соответственно и недостаток – скорость вставки новой пары обратно пропорциональна количеству элементов, сохраненных в коллекции, поскольку просто добавить новое значение в конец коллекции не получится. Еще одна

важная вещь, которую необходимо помнить при использовании данной коллекции – ключ должен быть уникальным...”

Ниже приведён исходный код контейнера map.

```
#ifndef __SGI_STL_MAP_H
#define __SGI_STL_MAP_H

#ifndef __SGI_STL_INTERNAL_TREE_H
#include <stl_tree.h>
#endif
#include <algobase.h>
#include <alloc.h>
#include <stl_map.h>

#ifdef __STL_USE_NAMESPACES
using __STD::rb_tree;
using __STD::map;
#endif /* __STL_USE_NAMESPACES */
#endif /* __SGI_STL_MAP_H */
```

Для использования контейнера map необходима инструкция:

```
#include <map>;
```

”… Если вы хотите использовать данную коллекцию, чтобы избежать дубликатов, то вы избежите их только по ключу”, сообщает Олег Ремизов про контейнер map.

set – это контейнер уникальных значений const Key каждое из которых также является и ключом (отсортированная коллекция, предназначенная для быстрого поиска необходимого значения). К ключу предъявляются те же требования, что и в случае ключа для map.

Использование контейнера set позволяет избежать повторного сохранения одного и того же значения. Скотт Майерс в Совете 22 пишет, "...Контейнеры set/multi set, как и все стандартные ассоциативные контейнеры, хранят свои элементы в отсортированном порядке, и правильное поведение этих контейнеров зависит от сохранения этого порядка. Если изменить значение элемента в ассоциативном контейнере (например, заменить 10 на 1000), новое значение

окажется в неправильной позиции. Это нарушит порядок сортировки элементов в контейнере. Сказанное, прежде всего, касается контейнеров map и multimap, поскольку программы, пытающиеся изменить значение ключа в этих контейнерах, не будут компилироваться..."

```
#ifndef __SGI_STL_SET_H
#define __SGI_STL_SET_H

#ifndef __SGI_STL_INTERNAL_TREE_H
#include <stl_tree.h>
#endif
#include <algobase.h>
#include <alloc.h>
#include <stl_set.h>
#ifdef __STL_USE_NAMESPACES
using __STD::rb_tree;
using __STD::set;
#endif /* __STL_USE_NAMESPACES */

#endif /* __SGI_STL_SET_H */
```

Для использования контейнера set необходима инструкция:

```
#include <set>;
```

multimap – это модифицированный конейнер map, в котором отсутствует требования уникальности ключа. Как сообщает Олег Ремизов “если вы произведете поиск по ключу, то вам вернется не одно значение, а набор значений, сохраненных с данным ключом”.

Ниже приведён исходный код контейнера multimap.

```
#ifndef __SGI_STL_MULTIMAP_H
#define __SGI_STL_MULTIMAP_H

#ifndef __SGI_STL_INTERNAL_TREE_H
#include <stl_tree.h>
#endif
#include <algobase.h>
#include <alloc.h>
#include <stl_multimap.h>
```

```
#ifdef __STL_USE_NAMESPACES
using __STD::rb_tree;
using __STD::multimap;
#endif /* __STL_USE_NAMESPACES */
#endif /* __SGI_STL_MULTIMAP_H */
```

Для использования контейнера multimap необходима инструкция:

```
#include < multimap.h>;
```

multiset – соответствнно замечанию Скотта Майерса, контейнер multiset это модифицированный контейнер set. Он также не содержит требования уникальности ключа, что, в свою очередь, приводит к возможности хранения дубликатов значений. Тем не менее, как объясняет Олег Ремизов, существует возможность быстрого нахождения значений по ключу в случае, если в процессе разработки был определен свой класс. Поскольку все значения в контейнерах map и set хранятся в отсортированном виде, то получается, что в них можно быстро отыскать необходимое значение по ключу. Однако при этом, операция вставки нового элемента T, по выражению Олега Ремизова, “будет стоить нам несколько дороже, чем например в vector”.

Ниже приведён исходный код контейнера multiset.

```
#ifndef __SGI_STL_MULTISET_H
#define __SGI_STL_MULTISET_H

#ifndef __SGI_STL_INTERNAL_TREE_H
#include <stl_tree.h>
#endif
#include <algobase.h>
#include <alloc.h>
#include <stl_multiset.h>
#ifdef __STL_USE_NAMESPACES
using __STD::rb_tree;
using __STD::multiset;
#endif /* __STL_USE_NAMESPACES */
#endif /* __SGI_STL_MULTISET_H */
```

17-ЛЕКЦИЯ. СТРОКИ И STL.

План:

1. Строки и STL.

Ключевые слова: *Функции для перебора всех членов коллекции, функции для сортировки членов коллекции, Функции для выполнения определенных арифметических действий над членами коллекции.*

Наверно каждый программист C/C++, как, впрочем, и представители других языковых средств программирования, включали в проектируемые программы модули для обработки строк. Не существует библиотек, которые не содержат класс для представления строк или даже несколько подобных классов. Библиотека STL в этом смысле также не исключение и строки в STL поддерживают как формат ASCII, так и формат Unicode. Говоря о программировании обработки строк в STL, Скотт Майерс предостерегает от неверных последствий при использовании динамической памяти при обработке строк: "…Каждый раз, когда вы готовы прибегнуть к динамическому выделению памяти под массив (собираетесь включить в программу строку вида ≪new T[...]≫), подумайте, нельзя ли вместо этого воспользоваться контейнером vector или string. Как правило, контейнер string используется в том случае, если T является символьным типом, а контейнер vector — во всех остальных случаях.

Контейнеры vector и string избавляют программиста от хлопот, о которых говорилось выше, поскольку они самостоятельно управляют своей памятью. Занимаемая ими память расширяется по мере добавления новых элементов, а при уничтожении контейнера vector или string деструктор автоматически уничтожает элементы контейнера и освобождает память, в которой они находятся.

Кроме того, контейнеры vector и string входят в семейство последовательных контейнеров STL, поэтому в вашем распоряжении оказывается весь арсенал алгоритмов STL, работающих с этими контейнерами. Впрочем, алгоритмы STL могут использоваться и с массивами, однако у массивов отсутствуют удобные функции begin, end, size и т. п., а также вложенные

определения типов (iterator, reverseiterator, value_type и т. д.), а указатели char* вряд ли могут сравниться со специализированными функциями контейнера string. Работа с библиотекой STL приводит к исключению практики применения встроенных массивов.

Контейнер string – представляет собой коллекцию, хранящую символы char в формате ASCII.

Исходный код контейнера контенера string приведен в Приложении. Для использования контейнера string необходима инструкция:

#include <string>;

wstring - это контейнер, храннящий двухбайтные символы wchar_t, используемые для представления символов в формате Unicode. Относительно строковых контейнеров Скотт Майерс указывает, что: "…Все, что говорится о контейнере string, в равной степени относится и к wstring, его аналогу с расширенной кодировкой символов. Соответственно, любые упоминания о связи между string и char или char* относятся и к связи между wstring и wchar_t или wchar_t*. Иначе говоря, отсутствие специальных упоминаний о строках с расширенной кодировкой символов не означает, что в STL они не поддерживаются. Контейнеры string и wstring являются специализациями одного шаблона basic_string".

В таблице представлены имена используемых в STL функций (методов).

Таблица

empty	определяет, не пустой ли контейнер
size	определяет размер контейнера
begin	возвращает прямой итератор, указывающий на начало контейнера
end	возвращает прямой итератор, указывающий на конец контейнера
rbegin	возвращает обратный итератор, указывающий на начало контейнера
rend	возвращает обратный итератор, указывающий на конец контейнера
clear	удаляет все элементы контейнера
erase	удаляет элемент или несколько элементов из контейнера
capacity	определяет размер буфера контейнера

Алгоритмы STL представлены в виде функций, которые можно разделить на три группы:

“…Функции для перебора всех членов коллекции и выполнения определенных действий над каждым из них: count, count_if, find, find_if, adjacent_find, for_each, mismatch, equal, search copy, copy_backward, swap, iter_swap, swap_ranges, fill, fill_n, generate, generate_n, replace, replace_if, transform, remove, remove_if, remove_copy, remove_copy_if, unique, unique_copy, reverse, reverse_copy, rotate, rotate_copy, random_shuffle, partition, stable_partition.

Функции для сортировки членов коллекции:

Sort, stable_sort, partial_sort, partial_sort_copy, nth_element, binary_search, lower_bound, upper_bound, equal_range, merge, inplace_merge, includes, set_union, set_intersection, set_difference, set_symmetric_difference, make_heap, push_heap, pop_heap, sort_heap, min, max, min_element, max_element, lexographical_compare, next_permutation, prev_permutation.

Функции для выполнения определенных арифметических действий над членами коллекции: Accumulate, inner_product, partial_sum, adjacent_difference”.

Остальной материал по библиотеке STL необходимо изучать по документации. Читая документацию, следует повторить все особенности элементов библиотеки STL, рассмотренные выше, и восполнить значительный, как по объёму, так и по значимости.

Вопросы по темам:

1. Что такое компонентная архитектура ПО?
2. С помощью какого языка программирования можно разрабатывать компонентную архитектуру и почему?
3. Что Вы знаете про STL?
4. Назовите другие библиотеки стандартных шаблонов.
5. Какие библиотеки разработки для компонентных архитектур Вы знаете?
6. Какое направление развивает Бьерн Страуструп в области разработки ПО?
7. Какое направление развивает Александр Степанов в области разработки ПО?

1.2. Темы практических занятий.

1-практическая занятия. Архитектура программного обеспечения. Проектирование архитектуры систем предметной области.

Ишнинг мақсади: Предмет соҳасининг тизим архитектурасини лойиҳалаштириш бўйича назарий маълумотларни ўрганиш ва берилган саволларга жавоб бериш.

Назарий маълумот.

Дастурий таъминот архитектураси лойиҳасининг таркибига қўйидагилар киради: элементлар тавсифи, яъни берилган тизим нимлардан туриши, бу элементлар орасидаги ўзаро таъсир схемаси, намуналарнинг (patterns) ҳужжатлари, улар асосида уларнинг жойлашиши бажарилади, ҳамда бу намуналарга тегишли бўлган рўйхат ва таркибга бўлган чеклашлар.

Фуқоралик ва саноатга ойд қурилишда лойиҳанинг тавсифи архитектуралик –қурилиш чизмалари ва ҳажмли моделлар, ҳамда қурилаётган объектлар ва унинг қурилиш технологияларининг текстли тавсифи ҳисобланади. Дастурий таъминот характеристекасини иллюстрацияланган воситалар сифатида ифодалаш, архитектуравий лойиҳалашда ҳар хил шартли ёзма белгилар – блок-схемалар, ER-диаграммасы, UML-диаграммасы, DFD-диаграммасы ҳамда макетлар қўлланилади.

Ҳар бир дастурий таъминот подсистемаси унинг компоненталари ва уларнинг ўзаро таъсирининг йиғинидан ташкил топган бўлиб, ушбу шартли белгиларни фойдаланиш орқали лойиҳанинг мос бўлагида батафсил тавсифланган бўлиши керак. Сабаби бундай подсистема каттароқ масштабдаги тизим учун таркибий элемент сифатида киритилиши мумкин. Дастурий таъминот архитектураси лойиҳасида шу воситалар ёрдамида тизимнинг катталаштирилган бўлакларининг батафсил тавсифларини ўз ичига олади.

Умуман лойиҳалаштириш ҳамда Дастурий таъминотни лойиҳалаштириш амалий фаволият тури хисобланади. Модомики ҳар хил вариантлар ичида, лойиҳалаштириш – бу табиятда йўқ бўлган нарсани яратиш саноати, Дастурий таъминот архитектори(лойиҳаловчиси) ўз фикрига ва лойиҳалаштириш саноатига эга бўлиши керак ва у лойиҳа қатнашувчилари ва буюртмачиларга зарур бўлаган дастурий таъминотни қуриш ва уни бошқаришга ва тизимнинг кейинги эволюциясини ишлатишда имкон беради.

Саволлар:

1. Дастурий таъминот архитектураси ҳақида тушинча беринг?

2. Қандай артефактлар предмет соҳасининг лойиҳалаштириш архитектурасининг мақсади бўлиб хисобланади?
3. Дастурий таъминотни яратишда архитектуранинг тўтган ўрни?
4. Дастурий таъминотни лойиҳалаштириш жараёни қандай кетма-кетликда бажарилади: Предмет соҳасининг тизим архитектурасини лойиҳалаштириш, моделини амалга ошириш режаси ва моделини амалга ошириш.
5. Дастурий таъминот яратишда архитекторнинг тўтган ўрни.

2-практическая занятия. Применение архитектурных образцов для проектирования программного обеспечение.

Ишнинг мақсади: Дастурий таъминотни лойиҳалаштириш учун архитектура намуналарини қўлланиш бўйича назарий маълумотларни ўрганиш ва берилган саволларга жавоб бериш.

Назарий маълумот.

Дастурий таъминот архитектураси терминининг 30 йиллик фойдаланиш тажрибасига қарамасдан дастурий таъминотни архитектуравий лойиҳалашни амалий жорий қилиш ҳозиргача давом этиб келмоқди ва саноатдаги дастурлашда искидай янги технологик йўналиш бўлиб ҳисобланиди. Лойиҳаланувчи дастурий тизимларнинг компонентали характерин ҳисобга олиш, архитектурали лойиҳалашга мўлжалланган адабиётларда кўпроқ диққат лойиҳалашнинг намуналарин(паттернлар) ишлаб чиқишга ва жорий қилишга қаратилган. Улардан бирида лойиҳалаш намуналарига қўйидагича аниқлама берилади “Ахборот тизимларни ишлаб чиқишда фойдаланиладиган ҳохлаган лойиҳалаш паттернни кўп учрайдиган лойиҳалаштириш масаласининг формаллашган туридаги таърифи, шу масаланинг мувофақиятли ечими, ҳамда шу ечимни ҳар хил ҳолатларда фойдаланишга тавсия бўлиб ҳисобланади”. Бундан бошқа лойиҳалаштириш барча фойдаланувчи номига эга бўлади. Тўғри тўзилган лойиҳалаштириш патернни бир марта мувофақиятли топилган ечимни қайта-

қайта фойдаланиш имкониятини беради. Шуни айтиб ўтиш керак, паттернлар билан ишлашда аҳамиятли бошланғич босқич кўрилаётган предмет соҳасини адекват(реалликга яқин) моделлаштириш бўлиб ҳисобланади.

К. Чарнецки ҳәм У. Айзенекер кўбчилик ҳолатларда лойиҳалаштиришда паттернларни(architectural patterns) янги намуна олиш учун периодли турда саралаб тўриш фойдали деб ҳисобланади. Бундай йўл билан олинган янги архитектуравий намуна белгили талаблар йиғинига мос бўлиши, секциялардан иборат ҳужжат кўринишида таърифга эга бўлиши керак ва у секцияларда номи, контексти,таъсирлар, ечимлар ва мисоллар бўлади.

Саволлар:

1. Кодни қайт фойдаланиш деганда нимани тушунасиз?
2. Паттеринни лойиҳалаштириш деганда нимани тушунасиз?
3. Кодни қайт фойдаланиш муоммолари бўлиши мукинми?

3-практическая занятия. Генеративное программирование

Ишнинг мақсади: Пайдо қилувчи дастурлаш бўйича назарий маълумотларни ўрганиш ва берилган саволларга жавоб бериш.

Назарий маълумот.

Пайдо қилувчи дастурлашнинг ғояси тизимлар ойласи учун пайдо қилувчи моделларни тўзиш ва лойиҳалаштиришга асосланган ва унинг мақсади шу моделлар бўйича аниқ тизимни генерациялаш бўлиб ҳисобланади.

Белгили предмет соҳасида(бу амалий дастурлаш соҳасига тегишли бўлиб) тизимлар ойласини ишлаб чиқишда амалий тажрибага эга бўлиш кўп марта фойладаниладиган дастурий таъминотни ишлаб чиқиш учун имкониятини беради.

Кўп марта фойдаланиладиган компоненталарни ишлаб чиқиш, тизимлар ойласи аъзоларининг умумийлигин аниқлаш билан бирга, ўзгариш хусусиятига эга аҳамиятли параметрларнида аниқлайди.

Предмет соҳасининг хусусиятларини лойиҳанинг характеристикаларини моделлаштириш имконияти билан аниқлаш, қандай характеристикалар ва ўзгарувчи параметрлерни тез амалга ошириш керек ва қай бирларини келажакка режалаштиришкераклигини аниқлашга имкон беради.

Генеративлик дастурлаш дастурий коднинг генераторларни фойдаланишни талаб этади ва улар олдин тўзилган код ва талапларнинг спецификациясини ҳисобга олиш асосида лойиҳанинг “пилотли” ва иш версиясини йиғишни автоматлаштиришга имкон беради. Генеративлик дастурлаш дастурий таъминотнинг архитекторларининг воситаси сифатида лойиҳанинг мантиғини, уни ишлаб чиқиш босқичидан бошлаб дастурий тизимнинг ҳаётий цикли давомида бошқаришга йўналтирилган.

Генеративлик дастурлашни архитектор фойдаланиши учун аниқ предмет соҳасида фойдаланишга яроқли CASE-тизимлар, дастурий таъминот генераторларива қайта фойдаланишга яроқли олдин тўзилган дастурий код керак бўлади.

Саволлар:

1. Генеративли дастурлаш деб нима айтамиз?
2. Дастурий таъминот генераторлари ҳақида нима биласиз?
3. Дастурий таъминотни лойиҳалаштириш учун архитектуравий намуналардан фойдаланиш бўйича тушунча.

4-практическая занятия. Автоматное программирование

Ишнинг мақсади: Пайдо қилувчи дастурлаш бўйича назарий маълумотларни ўрганиш ва берилган саволларга жавоб бериш.

Назарий маълумот.

Дастурий инженерияни автоматлаштириш соҳасида кўпроқ маълум бўлган Россияда ишлаб чиқилган лойиҳалаштириш ва ҳолатларни аниқ белгиловчи реактив объектли-йўналтирилган дастурларни амалга ошириш методи ҳисобланади. Методни қўллаш учун UniMod инструментал воситаси ишлаб чиқилган. Автоматли дастурлашни яратган Анатолий Абрамович Шалытонинг ишлари, ДТ ни генерациялаш учун бажарилувчи UML моделлаштириш тилини қўллаш ғояси билан боғлиқ. Метод автоматли дастурлашга(SWITCH технологияси) ва UML –нотацияларни (шартли ёзма белгиларни) қўллашга асосланган. UniMod инструментал дастурлаш воситаси шу методга асосланади ва Eclipse платформасига ўрнатилувчи модул бўлиб ҳисобланади.

Авторлар ўз ишларида, ҳозирги пайтда UML моделлаштириш тили моделлар тизимининг спецификация тили сифатида қўлланилаётганлигини таъкидлайди. Бунда бор бўлган UML –воситалари ҳар хил диаграммалар тузиш ва синфлар диаграммаси бўйича автомат тарзда дастурлаш тилларида коднинг “скелет” ини яратишга мумкинлик беради, бундан бошқа ҳолатлар диаграммаси бўйича дастур коди ҳаракатини генерациялашни автоматлаштириш имконини беради. Аммо маълум инструментлар ҳаракатлар модели билан коднинг “скелет” ни эффектив боғлашга тўла имкон бермайди, чунки уни ҳолатлар,хизматлар, кооперация ёки кетма-кетликлар диаграммалари ёрдамида таърифлаш мумкин.

Авторларинг фикри бўйича бир хил маъноли операцион семантиканинг йўқлиги, дастурни анъанавий тузишда коднинг ҳаракати таърифини моделда ва дастурда ҳар хил тасвирлашга олиб келади, яъна дастурчилар томонидан ҳаракат диаграммаларининг эркин интерпретацияланишига олиб келади, моделда код

ҳаракатини тарифлаш кўпинча формал эмас характерга эга. Бунда архитектор томонидан тўзиладиган моделнинг формал ҳаракати, дастурчилар дастлабки кодни ишлаб чиқишда ундан фойдаланмасдан ўзларининг тушунчасидан келиб чиқан ҳолда ишлаши мумкин.

UML тилида операцион семантиканинг пайдо бўлиши ДТ лойиҳалаштиришдаги қатнашчиларнинг диаграммаларни бир хил маъноликда тўшинишни ва бажарилувчи UML яратишга имкон беради, хусусан кодни генерациялаш тавсифланган моделни интерпретациялашда бевосита бажарилиши мумкин.

Саволлар:

1. Автоматли дастурлаш деб нима айтамиз?
2. Дастурий таъминот генераторлари ҳақида нима биласиз?
3. Дастурий таъминотни лойиҳалаштириш учун архитектуравий намуналардан фойдаланиш бўйича тушунча.

5-практическая занятия. Архитектура на базе моделей.

6-практическая занятия. Компонентная архитектура.

7-практическая занятия. Библиотека стандартных шаблонов STL

8-практическая занятия. Предварительное проектирование программного обеспечения

Цель работы:

- Проведение предварительного проектирования конкретной программы.
- Составить перечень требований и функциональных характеристик разрабатываемой программы.
- Разработка документа «Постановки задачи».

Порядок выполнения работы и отчетность.

Во время выполнения лабораторной работы необходимо определить потребность в программном изделии, его назначение и основные функциональные характеристики; составить перечень требований к нему.

Работа должна быть оформлена в виде документа «Постановка задачи».

Теоретические сведения.

Определение полного комплекса требований к программному изделию является первоначальной задачей его разработки. Некачественное определение требований приводит к созданию программного изделия, которое будет правильно решать неверно сформулированную задачу, а программный продукт не будет соответствовать истинным потребностям заказчика.

Поэтому при определении требований к программному изделию требуется соблюдать максимально возможную аккуратность и точность, чтобы затем эти требования можно было транслировать в разрабатываемый проект с минимальным числом ошибок. Требования задаются на естественном языке и должны быть очень точно сформулированы.

Требования оформляются в виде документа, в котором письменно излагается то, что будет, и что не будет сделано при выпуске программного изделия. В учебном заведении такой документ называется " Постановка задачи ".

Постановка задачи пишется на естественном языке в терминах понятных и пользователю и разработчику программного обеспечения и может содержать следующие разделы:

1. Заголовок к программе.

2. Условие задачи.

Формулируется условие задачи, краткое описание разрабатываемой программы, ее назначение и необходимые уточнения.

3. Начало/окончание работы.

Указывается месяц и год начала/окончания разработки программы.

4. Основание для разработки программы.

Основанием для разработки программы может быть заказ пользователя, задание администрации учебного заведения, контракт учебного заведения с другой организацией и пр.

5. Краткая характеристика объекта разработки.

Описывается объект разработки: как решается поставленная задача в настоящее время без разрабатываемой программы и какая часть ручной работы будет заменена программой.

6. Пользователь.

Указываются пользователи программы.

7. Цель и назначение разработки.

8. Основные требования.

Описываются требования пользователя к разрабатываемой программе.

Здесь же с точки зрения пользователя следует подробно перечислить функции программы.

9. Входная информация.

Перечисляются все входные данные программы с точки зрения их содержания и назначения - отчеты, файлы, записи, поля данных, таблицы... Их возможные носители и средства отображения информации и т.д.

10. Выходная информация.

Описываются выходные данные так же, как в пункте 9.

11. Требования к аппаратному и программному обеспечению.

Описывается конфигурация аппаратуры и программного обеспечения, в которых разрабатываемая программа может работать, другие программные продукты, от которых она зависит.

12. Внешние ограничения.

13. Эффективность.

Цели производительности, такие, как временные и объемные характеристики, пропускная способность, использование ресурсов и пр.

14. Безопасность данных от несанкционированного доступа.

15. Эргономические характеристики.

Эргономическими характеристиками изделия являются такие свойства, которые обеспечивают надежность, комфорт и продуктивность работы пользователей и операторов. Эргономика (греч.) - труд + закон - отрасль знания, изучающая трудовые процессы с целью создания наилучших условий труда.

16. Мобильность.

Описываются требования и цели обеспечения переноса программного продукта из одних рабочих условий в другие.

17. Окупаемость капиталовложений.

Определяется прибыль, которую даст создание программного продукта в понятиях, соответствующих целевому назначению организации.

18. Другие соглашения сторон.

19. Терминология.

Четко определяется вся терминология, которая может оказаться специфической для данной разработки.

Контрольные вопросы

1. Предмет "Технология программирования", основные понятия.
2. Предварительное проектирование (Системный анализ).
3. Постановка целей проекта и продукта.
4. Определение требований.

5. Документ "Постановка задачи". ("Соглашение о требованиях")

9-10-11-практические занятии. Применение CASE-технологий при разработке архитектуре программного обеспечения

Цель работы:

- Применение CASE-технологий при разработке архитектуре программного обеспечения.

Порядок выполнения работы и отчетность.

Автоматизация архитектурного проектирования программного обеспечения основывается на применении инструментальных программных средств, которые принято называть CASE (Computer- Aided Software/System Engineering). Несмотря на достаточное, на первый взгляд, количество существующих средств автоматизации проектирования архитектуры программного обеспечения, программные архитекторы по- прежнему нуждаются в расширении набора доступных средств автоматизации. По сравнению со своими коллегами в других областях инженерного творчества, уровень оснащения архитекторов CASE-средствами явно недостаточен и вот почему. Автоматизация структурных методологий, характерных для программирования, и возможность применения современных методов системной и программной инженерии должны позволять CASE-системам:

- улучшать качество создаваемого программного обеспечения за счет средств автоматического контроля проекта;
- создавать за короткое время прототип будущей системы с использованием генераторов программного кода, что позволяет на ранних этапах оценить ожидаемый результат;
- ускорять процесс проектирования, разработки и внедрения;
- позволять разработчику сосредоточиться на творческой части разработки за счет сокращения рутинной работы;

- поддерживать развитие и сопровождение проекта;
- применять технологии повторного использования архитектурных образцов.

Большинство CASE-средств, стремящихся удовлетворять перечисленным требованиям, основано на парадигме программирования – методология→метод→нотация→средство, где:

- методология определяет оценку и выбор проекта разрабатываемого программного обеспечения, последовательность разработки и правила распределения и назначения методов;
- метод представляет собой способ генерации описаний компонентов программного обеспечения;
- нотации это средства описания проектной логики, в том числе: диаграммы, графы, формальные и естественные языки, а также таблицы и блок-схемы, которые предназначены для описания структуры системы, описания элементов данных и назначение этапов обработки;
- средства (инструменты) для поддержки и усиления методов предназначены для участия пользователей при создании и редактировании графического проекта в интерактивном режиме. Средства способствуют организации проекта в виде иерархии уровней абстракции и выполняют роль проверки соответствия компонентов.

Автоматизация проектирования архитектуры программного обеспечения с применением CASE-средств не может основываться на выборе отдельно взятой CASE-системы, поскольку универсальной CASE-системы, отвечающей перечисленным требованиям и достаточной для производства ПО для любой предметной области пока не существует. С появлением новых CASE-систем, а также с развитием генераторов программного кода и с внедрением систем для повторного применения ранее разработанного кода станет возможным создавать

на базе предприятий-разработчиков программного обеспечения, системы автоматизированного проектирования программного обеспечения.

Методология и средства анализа и проектирования многокомпонент -ных информационных систем, содержащиеся в большинстве CASE-систем, позволяют применять методологии создания информационных систем с компонентной архитектурой. Значительный вклад в развитие компонентной методологии внесли сотрудники фирмы IBM Rational Software (особенно Г. Буч, Д. Рамбо и И. Якобсон). Анализ и проектирование информационных систем с компонентной архитектурой. основываются на использовании унифицированного языка моделирования UML, и поддерживаются целым спектром инструментальных программных средств визуального моделирования. В CASE-системах поддерживаются основные языки программирования C++, Java, Visual Basic, SmallTalk и т.д., а также популярные среды разработки MS Visual Studio, Delphi, PowerBuilder, средства автоматизированного тестирования и документирования, охватывающих жизненный цикл создания программных систем.

Наиболее известной CASE-системой объектно-ориентированного моделирования является Rational Rose компании IBM Rational Software.
Все продукты Rational Rose поддерживают язык Unified Modeling Language (UML). Тем не менее, эти продукты различаются технологиями реализации, которые они поддерживают.

CASE-система IBM Rational Software – Rational Rose позволяет автоматизировать этапы анализа и проектирования разрабатываемого программного обеспечения, а также предоставляет возможность генерации кода ПО на различных языках программирования для формирования макетов систем и позволяет автоматизировать выпуск проектной документации.

Rational Rose позволяет разрабатывать проектную документацию в виде диаграмм и спецификаций, а также производить генерацию программного кода

на различных языках программирования (C++, Smalltalk, PowerBuilder, Ada, SQLWindows и ObjectPro). В составе инструментальных программных средств CASE-системы Rational Rose, также содержатся средства реинжиниринга программного обеспечения. Такая возможность, предназначена для повторного использование программных компонент в новых проектах.

Методической основой применения CASE-системы Rational Rose является автоматизация процесса построения диаграмм классов, состояний, сценариев, модулей и процессов, а также формализации спецификаций логической и физической структуры модели, и описания статических и динамических аспектов разрабатываемого программного обеспечения.

Уникальность CASE-системы Rational Rose заключается в обеспечении архитектора ПО (проектировщика ПО) достаточными средствами проектирования, в том числе: репозиторий, графический интерфейс, средства просмотра проекта, средства контроля проекта, средства сбора статистики и генератор документов, генератор и анализатор программного кода и средства реинжиниринга.

Репозиторий CASE-системы Rational Rose обеспечивают "навигацию" по проекту (включая перемещение по иерархиям классов и подсистем, переключение от одного вида диаграмм к другому, средства контроля и сбора статистики, генератор отчётов и др.) позволяют моделировать проект ПО и сопровождать результат разработки в течение всего жизненного цикла программной системы.

Создаваемый встроенным генератором CASE-системы Rational Rose скелет кода программы на языке программирования C++ предназначается для его доработки традиционным методом прямого программирования на языке C++. При генерации программного кода в CASE-системе Rational Rose используется информация из логической и физической моделей проекта ПО. В результате “прогона” генератора формируются заголовки и описания классов и объектов.

Анализатор исходного кода C++ позволяет создавать модули проектов и осуществляет контроль правильности исходных текстов и диагностику ошибок. Получаемая модель проекта пригодна для её использования в качестве повторно применяемого кода.

CASE-система Rational Rose позволяет формировать такие проектные документы:

•диаграммы классов;

•диаграммы состояний;

•диаграммы сценариев;

•диаграммы модулей;

•диаграммы процессов;

•диаграммы компонентов;

•спецификации классов, объектов, атрибутов и операций;

•заготовки текстов программ,

а также модель разрабатываемой программной системы в текстовом формате (.mdl-файл).

12-практическая занятия. Разработка программного обеспечения

Цель работы:

- Определение этапов разработки конкретной программы.
- Разработка календарного плана создания конкретной программы.

Порядок выполнения работы и отчетность.

Во время выполнения лабораторной работы необходимо подробно проанализировать этапы разработки конкретной программы (ее жизненный цикл), начиная от возникновения потребности в ней до полного прекращения ее использования вследствие ее морального старения или потери необходимости решения соответствующих задач.

Работа должна быть оформлена в виде календарного плана разработки программы по форме:

№	Наименование этапа разработки программы	Срок исполнения		Примечания
		Начало	Окончание	

Теоретические сведения.

Обобщенная модель жизненного цикла программы может выглядеть так:

1. Системный анализ (предварительное проектирование ПИ)

а) исследование

б) осуществимость

- эксплуатационная
- экономическая
- коммерческая

2. Проектирование программы

а) конструирование программы

- функциональная декомпозиция задачи
- разработка архитектуры системы
- внешнее проектирование программы
- разработка архитектуры программы
- проектирование базы данных

б) программирование

- внутреннее проектирование форм и модулей
- определение свойств объектов и кодирование
- отладка форм и модулей
- компоновка форм и модулей в программу

г) отладка программы в целом

3. **Оценка (испытания) программы**

4. **Использование программного изделия**

Контрольные вопросы

1. Технология разработки программного обеспечения.

2. Требования, предъявляемые к «идеальной» технологии разработки программного обеспечения..

3. Объектно - ориентированные технологии разработки ПО.

4. Программное обеспечение как изделие.

5. Проектирование ПО. Обзор этапов проектирования ПО (жизненный цикл)

13-14-практические занятии. Построение функциональной схемы системы ПО

Цель работы:
- проведение функциональной декомпозиции решаемой задачи;
- построение функциональной схемы;

Порядок выполнения работы и отчетность.

Во время выполнения лабораторной работы необходимо провести функциональную декомпозицию решаемой задачи, построить соответствующую схему.

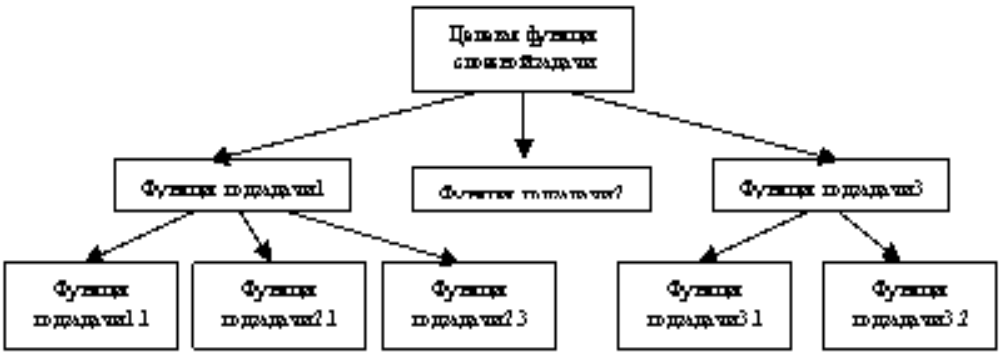
Работа должна быть оформлена в виде спецификации, содержащей функциональную схему решаемой задачи.

Теоретические сведения.

Проектирование программного обеспечения часто начинается с функциональной декомпозиции решаемой задачи.

Функциональная декомпозиция задачи представляет собой иерархическое разбиение сложной задачи на ряд проще решаемых небольших задач, которые, в свою очередь, разделяются на подзадачи до тех пор, пока каждая необходимая деталь в ней не будет определена достаточно ясно.

Концепция иерархической декомпозиции настолько естественна, что мы не всегда в состоянии осознать, как часто нам приходится использовать ее на практике. Она вытекает из человеческой потребности иметь дело с поддающимся управлению вполне определенным числом дискретных источников информации и производить «отсечение» информации до тех пор, пока число дискретных источников не станет приблизительно равно семи.



Строгая иерархическая декомпозиция подчиняется правилам:

1. На каждом уровне иерархии задача должна иметь законченный вид на данном уровне детализации;

2. На любом уровне иерархии каждое разбиение полностью охватывает отдельную задачу (функцию), соответствующую данному уровню детализации.

Контрольные вопросы

1. Методы проектирования программного обеспечения.
2. Восходящее и нисходящее проектирование программного обеспечения.
3. Объектно–ориентированное проектирование программного обеспечения.

4. Функциональная декомпозиция системы программного обеспечения.
5. Надежность программного обеспечения.

15-практическая занятия. Внешнее проектирование программного обеспечения

Цель работы:

- проведение внешнего проектирования конкретной программы;
- разработка взаимодействия разрабатываемой программы с пользователем: сценарий, экранные формы, набор подсказок, и пр.

Порядок выполнения работы и отчетность.

Во время выполнения лабораторной работы необходимо описать ожидаемое поведение разрабатываемой программы с точки зрения внешнего по отношению к нему наблюдателя (обычно - пользователя), то есть осуществить "конструирование" внешних взаимодействий будущей программы продукта с пользователем без конкретизации его внутреннего устройства.

Работа должна быть оформлена в виде внешней спецификации.

Теоретические сведения.

Внешнее проектирование мало, чем связано (если связано вообще) с программированием; более непосредственно оно касается понимания обстановки, проблем и нужд пользователя, психологии общения человека с машиной. Эта сторона внешнего проектирования становится все более значительной по мере того, как применение ЭВМ все больше начинает затрагивать пользователей, незнакомых с программированием.

Результаты внешнего проектирования программы отражаются во внешней спецификации, в которой может быть представлено описание следующих внешних аспектов программы:

- организация диалога программы с пользователем;
- состав меню, подменю ...;

- описание действий функциональных клавиш;
- все экранные формы или протокольные экранные сообщения;
- сообщения, выдаваемые пользователю во время проведения сеанса работы программы и выдаваемые пользователем на них ответы;
- сообщения об ошибках;
- подсказки пользователю, организация "помощи";
- структура и организация баз данных;
- описание и подготовка входных данных;
- выходные печатные формы;
- другие внешние сопряжения программы.

Внешняя спецификация должна быть написана на понятном пользователю и разработчику языке для уменьшения вероятности возможных недоразумений. Причем, проверку корректности и полноты спецификации необходимо проводить еще до начала программирования.

Основные правила организации диалога программы с пользователем.

1. Согласовывайте способ взаимодействия программы с пользователем, с его подготовкой и уровнем, с ограничениями, в условиях которых он работает.

2. Выходные данные должны выдаваться программой в требуемой форме и обязательно с комментариями. Нельзя, например, выдавать их в виде числа, а тем более - в виде набора чисел.

3. Обеспечьте концептуальную целостность для разных типов вводимых / выводимых сообщений. Например, все сообщения выдачи на экран, отчеты должны иметь одинаковые форматы, стиль и сокращения.

4. Старайтесь, чтобы пользователь вводил данные с клавиатуры как можно меньше. Будет лучше, если ему будет дана возможность выбора вводимых данных в виде меню, что исключит ошибки ввода пользователем. Сообщения, вводимые пользователем, должны быть как можно короче, но не настолько, чтобы исчезла их осмысленность.

5. Обеспечьте средства "помощи" - специальный набор функций (подсказки) по оказанию пользователю помощи, если тот запутался или забудет какое-либо правило взаимодействия.

6. Старайтесь, чтобы программа не рассердила пользователя. Избегайте оскорбительных сообщений. Общайтесь с пользователем на его языке, а не на тарабарском жаргоне программистов.

7. Помните о дизайне экрана. С эстетично оформленным экраном приятней работать. Экранная форма может быть разнообразной.

8. Старайтесь на каждое входное сообщение выдавать какое-либо уведомление. Программа должна принимать любые вводимые данные. Если данные не являются тем, что программа считает допустимым, то она должна информировать об этом пользователя.

9. Спроектируйте программу так, чтобы пользователь в любой момент работы с ней мог закончить эту работу или перейти в предыдущее состояние. Предполагается, что в первом случае программа успешно завершит свою работу (закроет открытые файлы, очистит переменные памяти и т.д.)

10. Ошибки пользователя должны обнаруживаться немедленно.

11. Не стремитесь исправлять входное сообщение пользователя.

Например, в медицинской информационной системе пользователь случайно нажимает на лишнюю клавишу, вследствие чего входное сообщение принимает вид "Рэтиловый спирт" вместо сообщения "Этиловый спирт". Система исправляет это сообщение на "Метиловый спирт". Известно, что этиловый спирт опьяняет, а метиловый спирт убивает.

12. Любые действия пользователя, как правильные, так и неправильные, должны контролироваться программой. В качестве отрицательного примера можно привести программу, которая может вдруг аварийно, преждевременно закончить свою работу.

Контрольные вопросы

1. Методическая, технологическая, инструментальная и организационная поддержка процесса проектирования ПО.
2. Конструирование программного обеспечения.
3. Внешнее проектирование ПО.
4. Правила организации диалога ПО с пользователем.
5. Создание интерфейса приложения.

16-17-практическая занятия. Разработка архитектуры программного обеспечения

Цель работы:

- разработать архитектуру программного изделия.

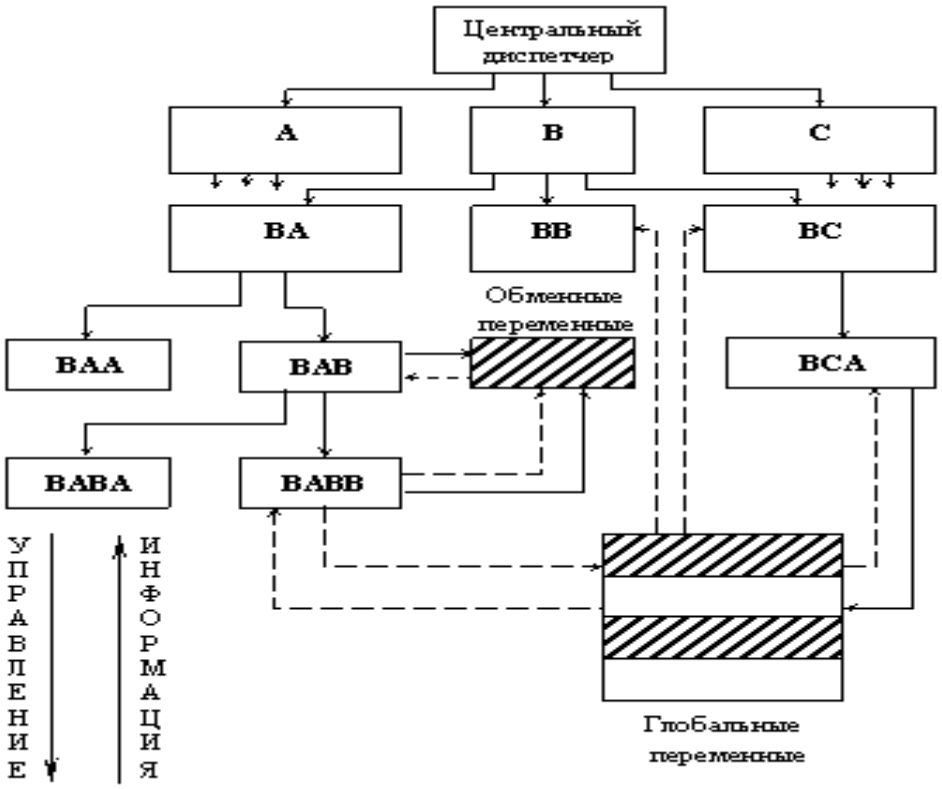
Порядок выполнения работы и отчетность.

Во время выполнения лабораторной работы необходимо разработать архитектуру разрабатываемой программы: спроектировать структуру всех его компонент, его объектно - модульно - иерархическое построение.

Работа должна быть оформлена в виде спецификации, содержащей архитектуру разрабатываемой программы.

Теоретические сведения.

Типовая архитектура программы может иметь вид:



1. Уровень -центральный диспетчер

2. Уровень - местные диспетчеры

3. Уровень - функциональные программы

4. Уровень – функциональные программы

5. Уровень - стандартные программы, библиотеки программ

Общие правила структурного построения программных модулей.

1. Каждый модуль характеризуется функциональной законченностью, автономностью и независимостью в оформлении от модулей, которые его используют и которые он вызывает. Высокую степень независимости модулей можно достичь с помощью двух методов оптимизации:

- усилением внутренних связей в каждом модуле, т.е. реализовать отдельные функции отдельными модулями (высокая прочность модуля).

- ослаблением взаимосвязи между модулями, применяя формальный механизм передачи параметров (слабое сцепление модулей).

2. Применяются стандартные правила организации связей по управлению и информации с другими модулями (смотри далее).

3. Комплексы программ разрабатываются в виде совокупности небольших по количеству (до 100) программных модулей, связанных иерархическим образом, что дает возможность полностью и относительно просто уяснить функцию и правила работы отдельных частей и комплекса программ в целом.

4. Как правило, модуль содержит от 10 до 1000 выполняемых операторов языка высокого уровня. Размеры модуля влияют на степень независимости программы, легкость ее чтения и тестирования.

5. Модуль прочный. Прочность модуля измеряется его внутренними связями. Модуль - это замкнутая программа, которая выполняет одну или несколько функций, обладает некоторой логикой.

6. Модуль предсказуемый, т.е. модуль, работа которого не зависит от предыстории его использования. Модули не должны сохранять никаких "воспоминаний" о предыдущем вызове.

7. Определена структура принятия решений. Желательно, чтобы те модули, на которые прямо влияет принятое решение, были подчиненными (вызываемыми) по отношению к принимающему решение модулю.

8. Объем данных, на которые модуль может ссылаться, должен быть сведен к минимуму.

9. Внутренняя процедура (или подпрограмма) - это замкнутая программа, физически расположенная в вызывающем ее модуле. Их следует избегать, т.к. их трудно изолировать для автономного тестирования и они не могут быть вызваны из модулей, отличных от тех, которые их физически содержат. Когда возникает потребность во внутренней процедуре, проектировщик должен рассмотреть возможность оформления ее в виде отдельного модуля.

10. В параметры процедуры следует включать только те переменные, через которые идет обмен информацией с другими программными единицами. Другие

переменные - это внутреннее дело процедуры. Процедуры, которые выдают в качестве результата только одно значение, оформляются как функции. Функция удобнее в использовании, так как ее результат непосредственно можно использовать в арифметическом и/или в логическом выражениях.

Правила связи программных модулей по управлению.

1. Передача управления вызываемому модулю всегда осуществляется через его начало, т.е. через первый оператор.

2. Выход из вызываемого модуля всегда происходит через его естественное окончание, т.е. после нормального его завершения.

3. По окончании исполнения вызываемого модуля управление передается в вызывающий модуль на оператор, следующий непосредственно за оператором вызова.

4. Модули низших уровней или одного уровня иерархии могут вызываться для исполнения только модулями высших уровней, т.е. модули низших уровней не могут вызывать модули высших уровней, а модули одного уровня - вызывать друг друга.

5. Если все же необходимо исполнить модуль с некоторой внутренней точки, то вызов все равно осуществляется стандартным образом (через его первый оператор), а точка начала задается в виде параметра. При этом в начале вызываемого модуля должен стоять переключатель, который обеспечивает передачу управления программой к его внутренним точкам по параметру, указанному при обращении к модулю.

6. В каждом модуле должна быть предусмотрена возможность подключения контрольных и отладочных средств; операторы, реализующие эти средства, обычно сосредотачиваются в конце модуля.

Правила связи программных модулей по информации.

1. Информация зон глобальных переменных доступна для использования любым модулям, входящим в комплекс программ или в группу программ в

соответствии с областью действия зоны глобальных переменных, т.е. глобальные переменные, могут быть доступны не для всего комплекса программ, а лишь для указанной в описании группы модулей.

2. Локальные переменные доступны лишь в пределах того модуля, в котором они определены или объявлены.

3. Для взаимодействия вызываемых и вызывающих модулей создаются зоны обменных переменных, информация из которых доступна лишь модулям, непосредственно связанным по управлению.

4. После окончания работы вызываемого модуля считается, что соответствующие регистры не содержат информации, являющейся результатом его работы. Запрещается их использовать в вызывающем модуле.

5. Информация, находящаяся в регистрах вызывающего модуля, при вызове должна быть сохранена на период выполнения вызываемого модуля и восстановлена при возврате управления в вызывающий модуль. Сохранение регистров может осуществлять как вызывающий, так и вызываемый модуль.

Контрольные вопросы

1. Конструирование архитектуры ПО.
2. Типовая архитектура программного обеспечения.
3. Общие правила структурного построения ПО.
4. Правила связи программных модулей по управлению.
5. Правила связи программных модулей по информации.

1.3. Темы самостоятельных работ.

1. Архитектура информационной системы.
2. Создание архитектуры.
3. Архитектурно-экономический цикл.
4. Документирование программной архитектуры.
5. Проектирование архитектуры.
6. Реконструкция программной архитектуры.
7. Повторное использование архитектурных средств.
8. Использование CASE-технологии.
9. Будущее программной архитектуры.
10. Анализ архитектуры.
11. Другие взгляды на архитектуру.
12. Архитектурные образцы, этолонные модели и эталонные варианты архитектуры.

1.4. Глоссарий

“Архитектура” и “инженерия”, как виды человеческой деятельности, существовали задолго до появления компьютерных технологий. Прежде всего, эти виды деятельности связывал процесс создания проекта — прототипа, прообраза предполагаемого или возможного объекта.

Архитектура ПО — это артефакт, представляющий собой результат процесса разработки программного обеспечения.

В состав архитектурного проекта ПО входят: описание элементов, из которых состоит данная система, схемы взаимодействий между этими элементами, документация образцов (patterns), на основе которых осуществляется их компоновка, а также список и содержание ограничений (требований), характерных для этих образцов.

В качестве иллюстративных средств выражения характеристик ПО, в архитектурном проекте используются различные нотации – блок-схемы (схемы алгоритмов), ER-диаграммы, UML- диаграммы, DFD-диаграммы, а также макеты.

План реализации, составленный архитектором ПО, регламентирует способы получения конкретных систем из общей архитектуры и компонентов.

При отсутствии однозначной методики по применению ранее разработанного кода, кроме опыта и интуиции разработчиков, имеется наука такого программирования, являющаяся частью программной инженерии. Она называется ≪Порождающее программирование (Generative programming)≫.

Идеи порождающего программирования основываются на проектировании и построении порождающих моделей для семейств систем с целью генерирования по этим моделям конкретной системы.
Программная среда, которая осуществляет сборку готового к применению программного продукта, в том числе: программной системы, компонента, класса, процедуры и тому подобных частей системы на основе высокоуровневой спецификации, называется “генератор”.

Генераторы представляют собой множество различных технологий, включая препроцессоры, метафункции, компиляторы, генерирующие классы и процедуры, генераторы кода CASE-систем (инструментов автоматизированного проектирования и создания программ), трансформационные компоненты и многое другое.

Ярким и пока единственным примером реализаций таких систем является система IP, разработанная Чарльзом (Каролем) Симони (Charles (Karoly) Simonyi) в период его работы в корпорации Microsoft. Работа в среде метапрограммирования IP (Intentional Programming) иногда называется ментальное или интенциальное программирование.

Язык UML является формальным языком спецификаций и отличается тем самым от синтаксиса традиционных формально-логических языков и языков программирования.

Концепцией MDA является описание представления алгоритмов на языке моделирования с последующим автоматическим преобразованием моделей в компьютерный код, причем программирование на базе моделей предполагает, что проектировщики ПО прежде всего создают наиболее подходящую модель, не "привязываясь" к платформе, на которой система будет реализована.

Автоматизация архитектурного проектирования программного обеспечения основывается на применении инструментальных программных средств, которые принято называть CASE (Computer-Aided Software/System Engineering).
Формируемая платформенно-независимая модель создается на языке унифицированного моделирования UML.

После создания модели PIM, создаются одна или несколько платформенно-зависимых моделей, так называемые PSM (Platform Specific Model), назначение которых обеспечение интеграции PIM с одной или несколькими технологиями разработки программных продуктов.

Автоматизация архитектурного проектирования программного обеспечения основывается на применении инструментальных программных средств, которые принято называть CASE (Computer-Aided Software/System Engineering).

Библиотека STL состоит из пяти основных видов компонентов: Алгоритм (Algorithm), который определяет вычислительную процедуру. Контейнер (Container), назначение которого управлять набором объектов в памяти. Итератор (Iterator), который обеспечивает средство доступа к содержимому контейнера для алгоритма. Функциональный объект (Function object), который инкапсулирует функцию в объекте для её использования другими компонентами. Адаптер (Adaptor), который настраивает компонент для обеспечения различного интерфейса.

2.1. Учебная программа.

2.2. Рабочая программа.

МИНИСТЕРСТВО ВЫСШЕГО И СРЕДНЕГО СПЕЦИАЛЬНОГО ОБРАЗОВАНИЯ РЕСПУБЛИКИ УЗБЕКИСТАН

МИНИСТЕРСТВО ПО РАЗВИТИЮ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И КОММУНИКАЦИЙ РЕСПУБЛИКИ УЗБЕКИСТАН

НУКУССКИЙ ФИЛИАЛ ТОШКЕНТСКОГО УНИВЕРСИТЕТА ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ ИМЕНИ МУХАММАДА АЛ-ХОРАЗМИЙ

Зарегистрирован

№_____

«УТВЕРЖДАЮ»
Зам. директор по учебной и научной работе
_____ Х.Сейткамалов
«_____» _____ 2018 г.

Р А Б О Ч А Я П Р О Г Р А М М А

дисциплины

“Архитектура программного обеспечение. Построение и кодирование программного обеспечение”

Область знаний:	300 000 – Услуг техническая производства
Область образования:	350 000 – Связь, информатизация и телекоммуникационные технологии
Направления образования:	5330600 – Программный инжиниринг

НУКУС-2018

Рабочая программа составлена на основании типовой программы “Архитектура программного обеспечение. Построение и кодирование программного обеспечение”.

Составитель:

Юлдашев К – ассистент кафедры “Праграммный инжиниринг” Нукусский филиал ТУИТ имени Мухаммада ал-Хорезмий.

Рецензия:

–д.т.н. кафедры “” Нукусский филиал ТУИТ имени Мухаммада ал-Хорезмий.

Рабочая программа дисциплины «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» одобрена на собраний кафедры (“___”от “___” – августа 2018 года), и передан на совет факультета.

Зав. кафедры: _____ д.т.н. Отениязов Р.

Рабочая программа дисциплины «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» утверждена на совете факультета “Компьютерный инжиниринг” (протокол “___” от “__” – августа 2018 г.).

Председатель совета факультета: _____ .

Согласовано:

Начальник учебно-методического отдела: _____ Ш.Ядгаров

Введение.

Рабочая программа курса «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» описывает цели и задачи учебной дисциплины в соответствии с требованиями Государственного образовательного стандарта.

Цель изучения предмета и его задачи

Основные цели преподавания дисциплины «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» – является расширение и углубление знаний бакалавров в области программного обеспечения, разработки, построения и методы кодирование, техники и технологий.

Требования по предмету к знаниям и методикам студентов

В процессе обучения данной дисциплины на основе предъявляемых требований, бакалавры должны выполнить лабораторные занятия, готовить тексты по лекциям, обладать способностью к самостоятельному мышлению, использовать полученные навыки для написания курсовых работи и диссертационных работ.

Студенты должны уметь применять полученные знания на практических и лабораторных занятиях, а в дальнейшем для решения задач в различных отраслях производства.

После обучения предметы “Архитектура программного обеспечение. Построение и кодирование программного обеспечение” студенты должны знать:

- архитектура программного обеспечение;
- атрибуты качество программного обеспечение;
- место жизненного цикла при разработке архитектуры программного обеспечение;
- требование программного обеспечение;
- проектирование архитектура программного обеспечение;
- разработка и тестирование архитектура программного обеспечение;
- оценка архитектура программного обеспечение;
- иметь представление об экономического роста и анализировать архитектура программного обеспечение.

Взаимная связь с другими предметами в учебнам плане.

Реализация программы основана на знаниях, приобретенных при обучении в бакалавриатуре по направлению «Программный инжиниринг» знании общих дисциплин «Программирование на C++», «Проектирование и моделирование компьютерных систем», «Принципы программирование», «Операционные системы и лаборатория», «Введение в программный инжиниринг», «Объектно-ориентированная программирования».

Новые педагогические технологии, используемые при изучении предмета

При освоении студентами курса «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» имеет особое значение применение новых информационно-педагогических технологий и современных методов обучения. Используются учебники, учебно-методические пособия, тексты лекций, раздаточные материалы, электронные материалы, лицензионные компьютерные программы, образцы информационных систем и технологий. Считается целесообразным проведение проблемных и открытых лекций,

использование компьютерной техники, интерактивных методов проведения занятий и игр. При проведении лекций и лабораторных занятий применяются передовые педагогические технологии.

Ниже приведены факторы, связанные с процессом обучения и влияющие на качество образования: высокий научно-педагогический уровень преподавания; проведение проблемных лекций; организация урока в виде вопросов-ответов; использование новых педагогических и мультимедийных технологий; постановка перед студентами задач, вызывающих интерес и стремление к знаниям.

При планировании курса «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» были применены следующие концептуальные основы:

Человеко-ориентированное образование. Такое образование, в соответствии с содержанием, контролирует полноценное развитие всех участников образовательного процесса. При планировании образовательного процесса, он был ориентирован не только на единичных слушателей курса, имеется в виду подход на основе связанных с образовательными целями будущими специальностями.

Системный подход. Целостность и взаимосвязанность всех элементов образовательного процесса в технологии образования.

Подход, ориентированный на работоспособность. Улучшение качеств слушателей, активизация и интенсификация работоспособности слушателей и направление всех способностей, возможностей слушателей на процесс обучения.

Диалогический подход. Этот подход показывает необходимость появления учебной вовлеченности. В итоге у слушателей усиливается творческая активность и самовыражение.

Появление совместного образования. Означает необходимость совместной работы над оценкой достигнутых результатов со стороны слушателей и преподавателя, на основе принципов демократии и равенств.

Проблемное обучение. Один из методов активизации способностей слушателей методом проблемного представления содержания предмета. При этом обеспечивается понимание методов научного познания через объективное противоречие и его решение, возникновение диалектического доказательства и его развитие, их творческое применение на практике.

Использование современных средств представления информации – применение в процессе обучения компьютеров и различных информационных технологий.

Формы обучения: диалог, полилог, участие, фронтал основанный на совместной работе и самостоятельном обучении, коллектив и группа.

Инструменты обучения: традиционные формы обучения (лекции, тестовые лекции) а также компьютер и информационные технологии.

Инструменты коммуникации: прямое взаимодействие на основе оперативной обратной связи со студентами

Методы и инструменты обратной связи: наблюдение, блиц-вопрос, диагностика обучения на основе анализа рубежных, текущих и итоговых контрольных.

Методы и инструменты управления: планирование учебных занятий в виде технологической карты, отмечающей этапы занятия, совместные действия преподавателя и студента в достижении поставленных целей, занятия в аудитории, контроль за внеклассной работой.

Мониторинг и контроль: планирование наблюдения в процессе занятий и всего курса. В конце курса – проведение тестов или письменных работ и на их основе оценка знаний студентов.

Распределения занятия дисциплины
«Архитектура программного обеспечение. Построение и кодирование программного обеспечение» по лекционным и часовым:

№	Наименование темы	Лекция	Практическое занятия	Самостоятельная работа
1	Архитектура программного обеспечения	2	2	6
2	Составление плана реализации модели предметной области программного обеспечения	2	2	6
3	Генеративное, интенциональное и автоматное программирование	2	2	8
4	Генеративное программирование	2	2	6
5	Генераторы программного обеспечения.	2	2	8
6	Применение архитектурных образцов для проектирования программного обеспечения	2	2	6
7	Интенциальное программирование	2	2	8
8	Автоматное программирование	2	2	6
9	Автоматизация архитектурного проектирования программного обеспечения. Архитектура на базе моделей.	2	2	6
10	Автоматизация архитектурного проектирования программного обеспечения. Архитектура на базе моделей.	2	2	8
11	Автоматизация архитектурного проектирования программного обеспечения. Преобразование моделей PIM PSM.	2	2	8
12	Автоматизация архитектурного проектирования программного обеспечения. Много платформенные модели	2	2	10
13	Применение CASE-технологий при проектирования программного обеспечения	2	2	10
14	Применение CASE-технологий при проектирования программного обеспечения	2	2	6
15	Компонентная архитектура	2	2	6
16	Стандартная библиотека шаблонов STL	2	2	6
17	Строки и STL.	2	2	6
ВСЕГО:		**34**	**34**	**120**

Основная часть:

Содержание теоретических занятий предмета.

В основной части (лекция) приводятся темы предмета в логической последовательности. Суть каждой темы раскрывается с помощью основных понятий и тезисов. По теме студенты должны получить необходимые знания и навыки согласно государственному стандарту.

В основной части рекомендуется раскрывать актуальность темы, ее соответствие требованиям работодателей и производителей, соответствие социально-политическим и демократическим направлениям.

Лекционные занятия

1. Архитектура программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О3; О4; О5; Д2; Д3; Д4.

2. Составление плана реализации модели предметной области программного обеспечения

Применяющие образовательные технологии: *диалогик ёндошув, муаммоли таълим*.

Литературы: О3; О4; Д2.

3. Генеративное, интенциональное и автоматное программирование

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О3; Д2; Д3.

4. Генеративное программирование

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О3; Д1; Д2; Д3.

5. Генераторы программного обеспечения.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим. Б/Б/Б жадвали, мунозара, Венн диаграммаси, Т-схема, ўз-ўзини назорат*

Литературы: О3; О4; О5; Д2; Д3; Д4.

6. Применение архитектурных образцов для проектирования программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим. Ажурали арра, бумеранг, усули, мунозара, ўз-ўзини назорат.*

Литературы: О3; О4; О5; Д2; Д3; Д4.

7. Интенциальное программирование

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим. Блитс, усули, мунозара, ўз-ўзини назорат.*

Литературы: О3; О4; О5; Д1; Д2; Д3; Д4.

8. Автоматное программирование

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О3; О4; Д2; Д3.

9. Автоматизация архитектурного проектирования программного обеспечения. Архитектура на базе моделей.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим. Маъруза, намойиш этиш, блиц-сўров, гуруҳларда ишлаш методи.*

Литературы: О3; О4; О5; Д2; Д3; Д4.

10. Автоматизация архитектурного проектирования программного обеспечения. Архитектура на базе моделей.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим. Маъруза, намойиш этиш, "Блиц-сўров" методлари.*

Литературы: О4; О5; Д2; Д3; Д4.

11. Автоматизация архитектурного проектирования программного обеспечения. Преобразование моделей PIM PSM.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим. Маъруза, намойиш этиш, "Блиц-сўров" методлари.*

Литературы: О3; О4; Д2; Д3.

12. Автоматизация архитектурного проектирования программного обеспечения. Много платформенные модели

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О3; О4; О5; Д2; Д3; Д4.

13. Применение CASE-технологий при проектирования программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О3; О4; О5; Д2; Д3; Д4.

14. Применение CASE-технологий при проектирования программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О3; О4; О5; Д2; Д3; Д4.

15. Компонентная архитектура

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О4; О5; Д2; Д3; Д4.

16. Стандартная библиотека шаблонов STL

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О3; О4; О5; Д2; Д3; Д4.

17. Строки и стандартная библиотека шаблонов STL.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О4; О5; Д2; Д3; Д4.

Календарно-тематичесий план лекционных занятий по предмету «Архитектура программного обеспечение. Построение и кодирование программного обеспечение»

П/н	**Темы лекционных занятий**	**Кол. часов**
1	Архитектура программного обеспечения	2
2	Составление плана реализации модели предметной области программного обеспечения	2
3	Генеративное, интенциональное и автоматное программирование	2
4	Генеративное программирование	2
5	Генераторы программного обеспечения.	2
6	Применение архитектурных образцов для проектирования программного обеспечения	2
7	Интенциальное программирование	2
8	Автоматное программирование	2
9	Автоматизация архитектурного проектирования программного обеспечения. Архитектура на базе моделей.	2
10	Автоматизация архитектурного проектирования программного обеспечения. Архитектура на базе моделей.	2
11	Автоматизация архитектурного проектирования программного обеспечения. Преобразование моделей PIM PSM.	2
12	Автоматизация архитектурного проектирования программного обеспечения. Много платформенные модели	2
13	Применение CASE-технологий при проектирования программного обеспечения	2
14	Применение CASE-технологий при проектирования программного обеспечения	2
15	Компонентная архитектура	2
16	Стандартная библиотека шаблонов STL	2
17	Строки и стандартная библиотека шаблонов STL.	2
Всего:		**34**

Рекомендуемые темы практических занятий.

На практических занятиях студенты изучают общее устройство компьютера, его основные и дополнительные устройства и принципы их работы.

В процессе обучения студенты получают знания об устройстве современных компьютерных архитектур, о современных процессорах и архитектурах шин, о центральной процессоре и последовательности процессорных команд, архитектуре ввода-вывода, шинах ввода-вывода, об устройстве видеопамяти и телекоммуникации, архитектурах параллельных компьютеров, о структуре мультипроцессоров и мультикомпьютеров, о разных этапах построения компьютерной архитектуры, этапы микроархитектуры, архитектура набора команд, а также основные понятия о прикладной системе и языке ассемблер.

При подготовке практических заданий со стороны преподавателей кафедры вносятся предложения. В том числе, рекомендуется на основе учебников и учебных пособий повышать знания студентов и широко использовать раздаточные материалы.

Применяющие образовательные технологии: *мозговой штурм, групповое обучение.*

Литературы: О3; О4; О5; Д1; Д2; Д3.

1. Архитектура программного обеспечения. Проектирование архитектуры систем предметной области.

Применяющие образовательные технологии: *диалогик ёндошув, муаммоли таълим.*

Литературы: О1; О2; Д2; Д3.

2. Применение архитектурных образцов для проектирования программного обеспечения.

Применяющие образовательные технологии: *диалогик ёндошув, муаммоли таълим*.

Литературы: О2; О3; Д2; Д3.

3. Генеративное программирование

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим*.

Литературы: О2; О3; Д2; Д3.

4. Автоматное программирование

Применяющие образовательные технологии**:** *ББББ, Инсерт, ўз-ўзини назорат.*

Литературы: О1; О2; О3; Д2; Д3.

5. Архитектура на базе моделей.

Применяющие образовательные технологии**:** *Маъруза, намойиш этиш, блиц-сўров.*

Литературы: О2; О3; О4; Д1; Д2; Д3; Д4.

6. Компонентная архитектура.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О2; О3; О4; Д1; Д2; Д3; Д4;

7. Библиотека стандартных шаблонов STL

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О2; О3; О4; О5; Д2; Д3; Д4.

8. Предварительное проектирование программного обеспечение.

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О1; О2; О4; О5; Д1; Д2; Д3.

9. Применение CASE-технологий при разработке архитектуре программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*

Литературы: О1; О2; О4; О5; Д1; Д2; Д3.

10. Применение CASE-технологий при разработке архитектуре программного обеспечения

Применяющие образовательные технологии: *диалогик ёндошув, муаммоли таълим*.

Литературы: О2; О3; Д2; Д3.

11. Применение CASE-технологий при разработке архитектуре программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О2; О3; Д2; Д3.

12. Разработка программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О1; О2; О3; Д2; Д3.

13. Проектирования функциональных схем программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О2; О3; О4; Д1; Д2; Д3; Д4.

14. Проектирования функциональных схем программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О2; О3; О4; Д1; Д2; Д3; Д4;

15. Внещная проектирования программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О2; О3; О4; О5; Д2; Д3; Д4.

16. Разработка архитектуры программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О1; О2; О4; О5; Д1; Д2; Д3.

17. Разработка архитектуры программного обеспечения

Применяющие образовательные технологии**:** *диалогик ёндошув, муаммоли таълим.*
Литературы: О1; О2; О4; О5; Д1; Д2; Д3.

№	Наименование темы практических занятий	Кол. часов
1.	Архитектура программного обеспечения. Проектирование архитектуры систем предметной области.	2
2.	Применение архитектурных образцов для проектирования программного обеспечение.	2
3.	Генеративное программирование	2
4.	Автоматное программирование	2
5.	Архитектура на базе моделей.	2
6.	Компонентная архитектура.	2
7.	Библиотека стандартных шаблонов STL	2
8.	Предварительное проектирование программного обеспечение.	2
9.	Применение CASE-технологий при разработке архитектуре программного обеспечения	2
10.	Применение CASE-технологий при разработке архитектуре программного обеспечения	2
11.	Применение CASE-технологий при разработке архитектуре программного обеспечения	2
12.	Разработка программного обеспечения	2
13.	Проектирования функциональных схем программного обеспечения	2
14.	Проектирования функциональных схем программного обеспечения	2
15.	Внещная проектирования программного обеспечения	2
16.	Разработка архитектуры программного обеспечения	2

17.	Разработка архитектуры программного обеспечения	2
ВСЕГО:		**34**

Форма и содержания организации самостоятельных работ.

Основная цель выполнения самостоятельных работ – организация непрерывного процесса обучения студентов под прямым руководством преподавателя, укрепление полученных знаний и навыков, подготовка к следующим занятиям, организация культуры умственного труда и самостоятельного получения новых знаний.

Содержание и объём самостоятельных работ студентов.

№	Темы самостоятельной работы	Задание	Срок исп.	Объём часов
1.	Архитектура информационной системы.	Технические литературы.. Индивидуальные задачи	1,2-недели	10
2.	Создание архитектуры.	Технические литературы.. Индивидуальные задачи	3,4-недели	10
3.	Архитектурно-экономический цикл.	Технические литературы. Изучение материаллов найденные в Интернете. Индивидуальные задачи	5,6-недели	10
4.	Документирование программной архитектуры.	Технические литературы. Индивидуальные задачи	7,8-недели	10
5.	Проектирование архитектуры.	Технические литературы. Индивидуальные задачи	9-недели	10
6.	Реконструкция программной архитектуры.	Технические литературы. Изучение материаллов найденные в Интернете. Индивидуальные задачи	10-недели	10
7.	Повторное использование архитектурных средств.	Технические литературы. Индивидуальные задачи	11-недели	10
8.	Использование CASE-технологии.	Технические литературы. Изучение материаллов найденные в Интернете. Индивидуальные задачи	12-недели	10
9.	Будущее программной архитектуры.	Изучение материаллов найденные в Интернете. Индивидуальные задачи	13,14-недели	10
10.	Анализ архитектуры.	Технические литературы. Индивидуальные задачи	15,16-недели	10
11.	Другие взгляды на архитектуру.	Технические литературы. Индивидуальные задачи	17-недели	10
12.	Архитектурные образцы, этолонные модели и эталонные варианты архитектуры.	Технические литературы. Индивидуальные задачи	18-недели	10
ВСЕГО:				**120**

Информационно-методическое обеспечение

В процессе преподавания дисциплины применялись современные методы образования, педагогические и информационно-коммуникационные технологии:

- на всех лекциях применяются презентации с помощью современных компьютерных и электронно-дидактических технологий;
- во время практических занятий широко применяются новейшие компьютерные устройства;
- во время практических занятий широко применяются мультимедийные технологии, во время всех практических занятий применяются педагогические технологии «мозговой штурм», «групповое рассуждение».

Критерии оценки знаний студентов на основе рейтинговой системы по предмету «Архитектура программного обеспечение. Построение и кодирование программного обеспечение»

Во время первого занятия по предмету студентам объявляется информация по рейтинговой таблице предмета, способам и формам оценки, количество и максимальный балл каждой контрольной, в том числе по баллам текущих и рубежных контрольных.

Для соответствия знаний и успеваемости студентов Государственным стандартам проводятся следующие виды контроля:

Текущий контроль (ТК) – способ определения и оценки уровня знаний студента по темам предмета и практическим навыкам. Текущий контроль может проводиться, исходя из качеств изучаемого предмета, в виде устного опроса, тестирования, диспута, контрольной работы, коллоквиума, проверки домашних заданий, а также в других подобных формах.

Рубежный контроль (РК) – способ определения и оценки уровня теоретических знаний и практических навыков студента, проводимый после прохождения соответствующего раздела (включающего в себя несколько тем) учебной программы. Рубежный контроль проводится дважды за семестр, а его форма (письменно, устно, тестирование, и т.д.) определяется исходя из объема выделенных часов на данный предмет.

Итоговый контроль (ИК) – способ оценки уровня усвоения студентами теоретических знаний и практических навыков по точным предметам по итогам семестра. Итоговый контроль проводится в форме письменной работы, основанной на опорных понятиях.

Итоговый контроль проводится при непосредственном участии комиссии, составленной заведующим кафедрой. При нарушении порядка проведения итогового контроля, результаты итогового контроля могут быть признаны недействительными. В этом случае итоговый контроль проводится заново.

Итоговый контроль проводится под руководством комиссии, составленной на основании приказа руководителя высшего учебного заведения. При нарушении порядка проведения итогового контроля, результаты итогового контроля могут быть признаны недействительными. В этом случае итоговый контроль проводится заново.

Рейтинговая система оценки знаний и навыков студентов основывается на успеваемости студентов, выраженной в баллах.

Показатель успеваемости студентов по курсу «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» за семестр оценивается по 100-балльной системе.

Эти 100 баллов распределяются по способам оценки следующим образом:

ИК – 30 баллов, остальные 70 баллов: **ТК** – 36 баллов, **РК** – 34 балла.

Балл	**Оценка**	**Уровень знаний студентов**
86-100	Отлично	Вывод и принятие решение. Творческое мышление. Самостоятельно делать вывод. Умение пользоваться знаниями на практике. Объяснение сути. Знать и объяснить. Иметь представление.
71-85	Хорошо	Самостоятельно делать вывод. Умение пользоваться знаниями на практике. Объяснение сути. Знать и объяснить. Иметь представление.
55-70	Удов.	Объяснение сути. Знать и объяснить. Иметь представление.
0-54	Неудов.	Незнание предмета.

Проходной балл по предмету составляет 55 баллов. Успеваемость ниже проходного балла не записывается в рейтинговую книжку.

Самостоятельные работы студентов по изучаемому предмету оцениваются в процессе текущих, рубежных и итоговых контрольных исходя из выделенных часов, в соответствии с выполнением порученных заданий.

Рейтинг студента по предмету определяется по следующей формуле:

$$R = \frac{V \bullet O'}{100}$$

где V – общий объем нагрузки, выделенной за семестр на данный предмет (в часах);

O' - успеваемость по предмету (в баллах).

Проходной балл по текущим и рубежным контрольным составляет 55 баллов. Студенты, набравшие баллы меньше проходного балла, не допускаются к итоговым контрольным.

Студенты, набравшие во время текущих контрольных (ТК) и рубежных контрольных (РК) больше 55 баллов, считаются успевающими по предмету и в дальнейшем допускаются до итогового контроля.

Общий балл, набранный студентом за семестр, в соответствии с правилами, высчитывается как сумма каждой контрольной.

Рубежный и итоговый контроли, в соответствии с календарным тематическим планом, проводится на основании рейтинговой контрольной таблицы, разработанной деканатом. Итоговый контроль проводится в последние 2 недели семестра.

Студентам, во время ТК и РК набравших баллы ниже проходных, а также студентам, которые по уважительным причинам не приняли участие в контрольных работах, дается срок до следующей ТК либо РК для повторной сдачи контрольной.

Студенты, в течение семестра набравшие менее 55 баллов по ТК и РК, либо в течение семестра набравшие по РК и ИК менее 55 баллов, считаются академическими должниками.

В случае несогласия с результатами контрольных, студент в течение одного дня после объявления результатов контрольной по предмету может обратиться в деканат с заявлением. В данном случае, на основании предложения декана факультета, по приказу ректора организовывается апелляционная комиссия, состоящая не менее чем из 3 (трех) человек.

Апелляционная комиссия, рассмотрев заявления студентов, в этот же день выносит решение.

Своевременное проведение контрольных работ согласно установленным правилам и их документирование должно проходить под контролем декана факультета, заведующего кафедрой, учебно-методического отдела и отдела внутреннего контроля и мониторинга.

Образцы критериев начисления баллов РК

№	Показатель	Баллы РК		
		Макс	**1-РК**	**2-РК**
1.	Посещаемость занятий. Активность на лекционных занятиях, наличие конспектов и их полнота.	4	0-2	0-2
2.	Ответы на тестовые или письменные вопросы	22	0-10	0-12
3.	Своевременное и качественное выполнение самостоятельных работ. Выполнение домашних работ по теме, успеваемость	8	0-4	0-4
Итого баллов по РК		**34**	**0-16**	**0-18**

Образцы критериев начисления баллов ТК

№	Показатель	Баллы ТК		
		макс	**1-ТК**	**2-ТК**
1.	Посещаемость и успеваемость по занятиям. Активность на лабораторных занятиях, наличие тетради для лабораторных работ и их состояние	4	0-2	0-2
2.	Своевременное и качественное выполнение лабораторных работ	24	0-12	0-12
3.	Своевременное и качественное выполнение самостоятельных работ. Выполнение домашних работ по теме, успеваемость	8	0-4	0-4
Итого по ТК		**36**	**0-18**	**0-18**

В случае, если итоговый контроль проводится в виде письменной работы, итоговый контроль проводится в виде 30-балльной письменной работы с вариантами.

В случае, если итоговый контроль по предмету проводится в виде письменной работы, итоговый контроль проводится согласно приведенной ниже таблице:

№	Показатель	Баллы ИК	
		Максимальный	Диапазон изменения
1.	Письменная контрольная работа по предмету	30	0-30
Итого		**30**	**0-30**

Критерии оценки письменных работ на итоговых контрольных

В случае, если итоговая контрольная работа проводится в виде письменной работы, проверка проводится методом множества вариантов. Каждый вариант состоит из 3 теоретических и 2 практических заданий. Теоретические вопросы включают в себя ключевые слова и словосочетания по предмету, и охватывают собой все темы предмета.

На каждый теоретический вопрос дается оценка по 6-балльной шкале (от 0 до 6 баллов). Практические задания оцениваются от 0 до 6 баллов. Студент может набрать максимум 30 баллов.

Для определения показателя успеваемости по письменной работе, набранные баллы по каждому ответу на поставленный в варианте вопрос складываются, и сумма считается оценкой за итоговую контрольную работу.

СПИСОК РЕКОМЕНДОВАННЫХ ЛИТЕРАТУР

Основная литература и учебные пособия:

1. “Software Architecture in Practice. Third Edition (Len Bass Paul Clements Rick Kazman) 2013y
2. Software Modeling And Design (Hassan Gomaa) 2011y
3. Генельт А.Е., «Автоматизированные методы разработки архитектуры программного обеспечения». Санкт-Петербург 2007.
4. Чарнецки К., Айзенекер У. Порождающее программирование: методы, инструменты, применение. СПб.: Питер. 2005 .
5. Брукс Ф. Мифический человеко-месяц, или Как создаются программные системы, СПб.: Символ-Плюс, 2001.
6. Ксензов М. Рефакторинг архитектуры программного обеспечения: выделение слоев. Труды Института Системного Программирования РАН, 2004 г.

Дополнительные литературы:

1. Architecting Software Intensive Systems: A Practitioner’s Guide, by Anthony J. Lattanze, Taylor and Francis/Auerbach 2008
2. Архитектура корпоративных программных приложений, Мартин Фаулер , 2006, Вильям.
3. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, 1995.
4. Метлис Я. Архитектура на базе моделей //Computerworld. 2006. № 30.
5. Грибачев К. Г. Delphi и Model Driven Architecture. Разработка приложений баз данных. СПб.: Питер, 2004.

Интернет сайты:

1. http://www.tuit.uz
2. http://www.atdt.uz
3. http://www.ziyonet.uz
4. http://askubuntu.com
5. http://opensource.com
6. http://www.wikipedia.org
7. http://www.intuit.ru

2.4. Тесты и варианты для контроля знания студентов.

1-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Что такое архитектура программного обеспечения?	3 баллов
2.	Какие Вы знаете генераторы кода?	3 баллов
3.	Что такое модельная архитектура MDA?	3 баллов
4.	Что такое PSM и PIM?	3 баллов
2-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Какие артефакты являются целью проектирования архитектуры программного обеспечения?	3 баллов
2.	Дайте объяснение сокращению IP.	3 баллов
3.	Что такое CIM?	3 баллов
4.	Что Вы знаете про STL?	3 баллов
3-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Какова роль архитектора при создании программного обеспечения?	3 баллов
2.	Что такое автоматное программирование?	3 баллов
3.	Что такое PIM?	3 баллов
4.	Назовите другие библиотеки стандартных шаблонов.	3 баллов
4-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	В каком порядке (очередности) выполняются процессы проектирования программного обеспечения: проектирование архитектуры систем предметной области,	3 баллов

	составление плана реализации модели и реализация модели?	
2.	В чем заключается подход генетического программирования?	3 баллов
3.	Что такое PSM?	3 баллов
4.	Какие библиотеки разработки для компонентных архитектур Вы знаете?	3 баллов
5-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Назовите состав архитектурной части проекта разработки программного обеспечения.	3 баллов
2.	Назовите отличия генеративного программирования от компонентно-ориентированного программирования?	3 баллов
3.	Чем отличаются платформенно-независимая модель от вычислительно-независимой модели?	3 баллов
4.	Какое направление развивает Бьерн Страуструп в области разработки ПО?	3 баллов
6-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Какова роль архитектора при создании программного обеспечения?	3 баллов
2.	Что такое повторное использование кода?	3 баллов
3.	Что Вы знаете о CASE-технологиях и о CASE-системах?	3 баллов
4.	Какое направление развивает Александр Степанов в области разработки ПО?	3 баллов
7-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Дайте определение Generative Programming.	3 баллов
2.	Что такое паттерн проектирования?	3 баллов
3.	Назовите CASE-систему, использование которой, на Ваш взгляд, является предпочтительным для разработки программного обеспечения?	3 баллов

4.	Дайте объяснение понятию управление на основе модели.	3 баллов
8-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Перечислите основные понятия порождающего программирования.	3 баллов
2.	Что было создано в результате работ Чарльза Симони?	3 баллов
3.	Что объединяет CASE-технологии и почему?	3 баллов
4.	Что такое модельная архитектура MDA?	3 баллов
9-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	В чем отличие от просто генерации кода?	3 баллов
2.	Какое направление развивает Сергей Дмитриев в области разработки программного обеспечения?	3 баллов
3.	Что такое компонентная архитектура ПО?	3 баллов
4.	Что такое архитектура программного обеспечения?	3 баллов
10-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Существуют ли проблемы при повторном использовании кода?	3 баллов
2.	Какое направление развивает Анатолий Шалыто в области разработки программного обеспечения?	3 баллов
3.	С помощью какого языка программирования можно разрабатывать компонентную архитектуру и почему?	3 баллов
4.	Что такое архитектура программного обеспечения?	3 баллов
11-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Назовите CASE-систему, использование которой, на Ваш взгляд, является предпочтительным для разработки программного обеспечения?	3 баллов

2.	Что такое паттерн проектирования?	3 баллов
3.	Дайте определение Generative Programming.	3 баллов
4.	Дайте объяснение понятию управление на основе модели.	3 баллов
12-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Что такое CIM?	3 баллов
2.	Какие Вы знаете генераторы кода?	3 баллов
3.	Что Вы знаете про STL?	3 баллов
4.	Что такое PSM и PIM?	3 баллов
13-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Дайте объяснение сокращению IP.	3 баллов
2.	Какие артефакты являются целью проектирования архитектуры программного обеспечения?	3 баллов
3.	Что такое архитектура программного обеспечения?	3 баллов
4.	Что такое модельная архитектура MDA?	3 баллов
14-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	В чем заключается подход генетического программирования?	3 баллов
2.	Что такое автоматное программирование?	3 баллов
3.	Что такое PIM?	3 баллов
4.	Какова роль архитектора при создании программного обеспечения?	3 баллов
15-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Назовите другие библиотеки стандартных шаблонов.	3 баллов

2.	В каком порядке (очередности) выполняются процессы проектирования программного обеспечения: проектирование архитектуры систем предметной области, составление плана реализации модели и реализация модели?	3 баллов
3.	Что такое PSM?	3 баллов
4.	Какие библиотеки разработки для компонентных архитектур Вы знаете?	3 баллов
16-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Назовите отличия генеративного программирования от компонентно-ориентированного программирования?	3 баллов
2.	Что Вы знаете о CASE-технологиях и о CASE-системах?	3 баллов
3.	Чем отличаются платформенно-независимая модель от вычислительно-независимой модели?	3 баллов
4.	Какое направление развивает Бьерн Страуструп в области разработки ПО?	3 баллов
17-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Назовите состав архитектурной части проекта разработки программного обеспечения.	3 баллов
2.	Что такое повторное использование кода?	3 баллов
3.	Какова роль архитектора при создании программного обеспечения?	3 баллов
4.	Какое направление развивает Александр Степанов в области разработки ПО?	3 баллов
18-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Что было создано в результате работ Чарльза Симони?	3 баллов
2.	Перечислите основные понятия порождающего программирования.	3 баллов
3.	Назовите CASE-систему, использование которой, на Ваш взгляд, является предпочтительным для разработки программного обеспечения?	3 баллов

4.	Дайте объяснение понятию управление на основе модели.	3 баллов
19-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Что такое паттерн проектирования?	3 баллов
2.	Дайте определение Generative Programming.	3 баллов
3.	Что объединяет CASE-технологии и почему?	3 баллов
4.	Что такое модельная архитектура MDA?	3 баллов
20-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	С помощью какого языка программирования можно разрабатывать компонентную архитектуру и почему?	3 баллов
2.	Какое направление развивает Сергей Дмитриев в области разработки программного обеспечения?	3 баллов
3.	В чем отличие от просто генерации кода?	3 баллов
4.	Что такое архитектура программного обеспечения?	3 баллов
21-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Существуют ли проблемы при повторном использовании кода?	3 баллов
2.	Какое направление развивает Анатолий Шалыто в области разработки программного обеспечения?	3 баллов
3.	Что такое компонентная архитектура ПО?	3 баллов
4.	Дайте определение Generative Programming.	3 баллов
22-В	**АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ. ПОСТРОЕНИЕ И КОДИРОВАНИЕ ПРОГРАММНОГО ОБЕСПЕЧЕНИЕ**	**Всего: 12 баллов**
1.	Назовите CASE-систему, использование которой, на Ваш взгляд, является предпочтительным для разработки программного обеспечения?	3 баллов
2.	Что такое паттерн проектирования?	3 баллов
3.	Что такое архитектура программного обеспечения?	3 баллов
4.	Дайте объяснение понятию управление на основе модели.	3 баллов

2.5. Методические указания по применению критериев оценок.

Критерии оценки знаний студентов на основе рейтинговой системы по предмету «Архитектура программного обеспечение. Построение и кодирование программного обеспечение»

Во время первого занятия по предмету студентам объявляется информация по рейтинговой таблице предмета, способам и формам оценки, количество и максимальный балл каждой контрольной, в том числе по баллам текущих и рубежных контрольных.

Для соответствия знаний и успеваемости студентов Государственным стандартам проводятся следующие виды контроля:

Текущий контроль (ТК) – способ определения и оценки уровня знаний студента по темам предмета и практическим навыкам. Текущий контроль может проводиться, исходя из качеств изучаемого предмета, в виде устного опроса, тестирования, диспута, контрольной работы, коллоквиума, проверки домашних заданий, а также в других подобных формах.

Рубежный контроль (РК) – способ определения и оценки уровня теоретических знаний и практических навыков студента, проводимый после прохождения соответствующего раздела (включающего в себя несколько тем) учебной программы. Рубежный контроль проводится дважды за семестр, а его форма (письменно, устно, тестирование, и т.д.) определяется исходя из объема выделенных часов на данный предмет.

Итоговый контроль (ИК) – способ оценки уровня усвоения студентами теоретических знаний и практических навыков по точным предметам по итогам семестра. Итоговый контроль проводится в форме письменной работы, основанной на опорных понятиях.

Итоговый контроль проводится при непосредственном участии комиссии, составленной заведующим кафедрой. При нарушении порядка проведения итогового контроля, результаты итогового контроля могут быть признаны недействительными. В этом случае итоговый контроль проводится заново.

Итоговый контроль проводится под руководством комиссии, составленной на основании приказа руководителя высшего учебного заведения. При нарушении порядка проведения итогового контроля, результаты итогового контроля могут быть признаны недействительными. В этом случае итоговый контроль проводится заново.

Рейтинговая система оценки знаний и навыков студентов основывается на успеваемости студентов, выраженной в баллах.

Показатель успеваемости студентов по курсу «Архитектура программного обеспечение. Построение и кодирование программного обеспечение» за семестр оценивается по 100-балльной системе.

Эти 100 баллов распределяются по способам оценки следующим образом:

ИК – 30 баллов, остальные 70 баллов: **ТК** – 36 баллов, **РК** – 34 балла.

Балл	Оценка	Уровень знаний студентов
86-100	Отлично	Вывод и принятие решение. Творческое мышление. Самостоятельно делать вывод. Умение пользоваться знаниями на практике. Объяснение сути. Знать и объяснить. Иметь представление.
71-85	Хорошо	Самостоятельно делать вывод. Умение пользоваться знаниями на практике. Объяснение сути. Знать и объяснить. Иметь представление.
55-70	Удов.	Объяснение сути. Знать и объяснить. Иметь представление.

0-54	Неудов.	Незнание предмета.

Проходной балл по предмету составляет 55 баллов. Успеваемость ниже проходного балла не записывается в рейтинговую книжку.

Самостоятельные работы студентов по изучаемому предмету оцениваются в процессе текущих, рубежных и итоговых контрольных исходя из выделенных часов, в соответствии с выполнением порученных заданий.

Рейтинг студента по предмету определяется по следующей формуле:

$$R = \frac{V \bullet O'}{100}$$

где V – общий объем нагрузки, выделенной за семестр на данный предмет (в часах);

O' - успеваемость по предмету (в баллах).

Проходной балл по текущим и рубежным контрольным составляет 55 баллов. Студенты, набравшие баллы меньше проходного балла, не допускаются к итоговым контрольным.

Студенты, набравшие во время текущих контрольных (ТК) и рубежных контрольных (РК) больше 55 баллов, считаются успевающими по предмету и в дальнейшем допускаются до итогового контроля.

Общий балл, набранный студентом за семестр, в соответствии с правилами, высчитывается как сумма каждой контрольной.

Рубежный и итоговый контроли, в соответствии с календарным тематическим планом, проводится на основании рейтинговой контрольной таблицы, разработанной деканатом. Итоговый контроль проводится в последние 2 недели семестра.

Студентам, во время ТК и РК набравших баллы ниже проходных, а также студентам, которые по уважительным причинам не приняли участие в контрольных работах, дается срок до следующей ТК либо РК для повторной сдачи контрольной.

Студенты, в течение семестра набравшие менее 55 баллов по ТК и РК, либо в течение семестра набравшие по РК и ИК менее 55 баллов, считаются академическими должниками.

В случае несогласия с результатами контрольных, студент в течение одного дня после объявления результатов контрольной по предмету может обратиться в деканат с заявлением. В данном случае, на основании предложения декана факультета, по приказу ректора организовывается апелляционная комиссия, состоящая не менее чем из 3 (трех) человек.

Апелляционная комиссия, рассмотрев заявления студентов, в этот же день выносит решение.

Своевременное проведение контрольных работ согласно установленным правилам и их документирование должно проходить под контролем декана факультета, заведующего кафедрой, учебно-методического отдела и отдела внутреннего контроля и мониторинга.

Образцы критериев начисления баллов РК

№	Показатель	Баллы РК		
		Макс	**1-РК**	**2-РК**
1.	Посещаемость занятий. Активность на лекционных занятиях, наличие конспектов и их полнота.	4	0-2	0-2
2.	Ответы на тестовые или письменные вопросы	22	0-10	0-12
3.	Своевременное и качественное выполнение самостоятельных работ. Выполнение домашних работ по теме, успеваемость	8	0-4	0-4
Итого баллов по РК		**34**	**0-16**	**0-18**

Образцы критериев начисления баллов ТК

№	Показатель	Баллы ТК		
		макс	1-ТК	2-ТК
1.	Посещаемость и успеваемость по занятиям. Активность на лабораторных занятиях, наличие тетради для лабораторных работ и их состояние	4	0-2	0-2
2.	Своевременное и качественное выполнение лабораторных работ	24	0-12	0-12
3.	Своевременное и качественное выполнение самостоятельных работ. Выполнение домашних работ по теме, успеваемость	8	0-4	0-4
Итого по ТК		**36**	**0-18**	**0-18**

В случае, если итоговый контроль проводится в виде письменной работы, итоговый контроль проводится в виде 30-балльной письменной работы с вариантами.

В случае, если итоговый контроль по предмету проводится в виде письменной работы, итоговый контроль проводится согласно приведенной ниже таблице:

№	Показатель	Баллы ИК	
		Максимальный	Диапазон изменения
1.	Письменная контрольная работа по предмету	30	0-30
Итого		**30**	**0-30**

Критерии оценки письменных работ на итоговых контрольных

В случае, если итоговая контрольная работа проводится в виде письменной работы, проверка проводится методом множества вариантов. Каждый вариант состоит из 3 теоретических и 2 практических заданий. Теоретические вопросы включают в себя ключевые слова и словосочетания по предмету, и охватывают собой все темы предмета.

На каждый теоретический вопрос дается оценка по 6-балльной шкале (от 0 до 6 баллов). Практические задания оцениваются от 0 до 6 баллов. Студент может набрать максимум 30 баллов.

Для определения показателя успеваемости по письменной работе, набранные баллы по каждому ответу на поставленный в варианте вопрос складываются, и сумма считается оценкой за итоговую контрольную работу.