Учреждение образования
«Гомельский государственный университет
имени Франциска Скорины»

Факультет физики и информационных технологий
Кафедра автоматизированных систем обработки информации

СОГЛАСОВАНО
Заведующий кафедрой
автоматизированных систем
обработки информации
А.В.Воруев
__ _____ 2023 г.

СОГЛАСОВАНО
Декан
факультета физики и
информационных технологий
А.Л.Самофалов
__ _____ 2023 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ

ПРОЕКТИРОВАНИЕ СИСТЕМ ОБРАБОТКИ ДАННЫХ

для специальности
1-53 01 02 Автоматизированные системы обработки информации

составители: старший преподаватель Сердюкова М.А.
профессор кафедры Зыкунов В.А.
старший преподаватель Пугачева Е.Е.

Рассмотрено и утверждено
на заседании кафедры АСОИ
__17 октября___ 2023 г., протокол № _3_

Рассмотрено и утверждено
на заседании научно-методического
совета университета
__28 ноября___ 2023 г., протокол № _4_

Гомель 2023

1 ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Электронный учебно-методический комплекс (ЭУМК) по дисциплине «Проектирование систем обработки данных» представляет собой комплекс систематизированных учебных, методических и вспомогательных материалов, предназначенных для использования в образовательном процессе специальности 1-53 01 02 – Автоматизированные системы обработки информации.

ЭУМК разработан в соответствии со следующими нормативными документами:

1. Положением об учебно-методическом комплексе на уровне высшего образования, утвержденном постановлением Министерства образования Республики Беларусь от 26.07.2011 №167.

2. Учебного плана ГГУ имени Ф.Скорины регистрационный № I 53-1-13, дата утверждения 25.08.2013.

3. Учебной программой по учебной дисциплине «Проектирование систем обработки данных» для специальности 1-53 01 02 Автоматизированные системы обработки информации, утвержденной 01.06.2016, регистрационный номер УД-37-2016-150/уч.

Цель создания ЭУМК – получить специальные знания, умения и навыки, необходимые инженеру по информационным технологиям в процессе проектирования автоматизированных систем обработки данных.

ЭУМК направлен на всестороннюю подготовку студентов теоретическим и прикладным аспектам проектирования систем обработки данных, ключевым понятиям архитектуры программного обеспечения предприятия.

ЭУМК способствует успешному осуществлению учебной деятельности, дает возможность планировать и осуществлять самостоятельную управляемую работу студентов, обеспечивает рациональное распределение учебного времени по темам учебной дисциплины и совершенствование методики проведения занятий.

ЭУМК состоит из теоретического, практического и вспомогательного разделов. Теоретический раздел содержит тексты лекций. Практический раздел содержит методические рекомендации к лабораторным работам, тестовые задания и вопросы для самоконтроля. Вспомогательный раздел содержит учебную программу и список литературы.

Теоретический раздел содержит лекционный материал по всем темам учебной программы, включая и темы, вынесенные на самостоятельное изучение.

Практический раздел включает в себя темы лабораторных занятий и задания с краткими методическими указаниями по выполнению лабораторных работ. В разделе так же приводятся некоторый набор тестовых заданий и к каждой теми указаны вопросы для самоконтроля.

Вспомогательный раздел содержит необходимые элементы учебно-программной документации по дисциплине с указанием рекомендуемой литературы (основной, дополнительной, вспомогательной).

Все разделы ЭУМК в полной мере соответствуют содержанию учебной программы и объему учебного плана.

Дисциплина компонента учреждения высшего образования «Проектирование систем обработки данных» изучается студентами 4 курса дневной формы обучения специальности 1-53 01 02 – «Автоматизированные системы обработки информации»; студентами 5 курса заочной формы обучения специальности 1-53 01 02 – «Автоматизированные системы обработки информации»; студентами 4 курса заочной интегрированной со средним специальным образованием формы обучения специальности 1-53 01 02 – «Автоматизированные системы обработки информации».

Дневная форма обучения: всего часов по плану - 134, аудиторное количество часов – 64; из них: лекционных занятий – 32 (в том числе УСР – 12), практических занятий – 32.

Форма отчётности – экзамен в 7 семестре.

Заочная форма обучения: всего часов по плану - 134, аудиторное количество часов – 16, из них: лекционных занятий – 12, практических занятий – 4.

Форма отчётности – экзамен в 9 семестре.

Заочная форма обучения (интегрированная на основе среднего специального образования): всего часов по плану - 66, аудиторное количество часов – 8, из них: лекционных занятий – 6, практических занятий – 2.

Форма отчётности – экзамен в 7 семестре.

2 ТЕКСТЫ ЛЕКЦИЙ

1. Понятие жизненного цикла системы обработки данных

1.1 Определение системы обработки данных

Системы обработки данных, комплекс взаимоувязанных методов и средств сбора и обработки данных, необходимых для организации управления объектами. С. о. д. основываются на применении ЭВМ и других современных средств информационной техники, поэтому их также называют автоматизированными системами обработки данных (АСОД). Без ЭВМ построение С. о. д. возможно только на небольших объектах. Применение ЭВМ означает выполнение не отдельных информационно-вычислительных работ, а совокупности работ, связанных в единый комплекс и реализуемых на основе единого технологического процесса.

С. о. д. следует отличать от автоматизированных систем управления (АСУ). В функции АСУ включается прежде всего выполнение расчётов, связанных с решением задач управления, с выбором оптимальных вариантов планов на основе экономико-математических методов и моделей и т. п. Их прямое назначение — повышение эффективности управления. Функции же С. о. д. — сбор, хранение, поиск, обработка необходимых для выполнения этих расчётов данных с наименьшими затратами. При создании АСОД ставится задача отобрать и автоматизировать трудоёмкие, регулярно повторяющиеся рутинные операции над большими массивами данных. С. о. д. — это обычно часть и первая ступень развития АСУ. Однако С. о. д. функционируют и как независимые системы. В ряде случаев более эффективно объединять в рамках одной системы обработку однородных данных для большого числа задач управления, решаемых в разных АСУ; создавать С. о. д. коллективного пользования.

Первые С. о. д. начали создаваться в США в 50-х гг. 20 в., когда выяснилась нецелесообразность использования ЭВМ для решения отдельных задач, например расчёта заработной платы, учёта товарно-материальных ценностей и т. п., и необходимость комплексной обработки данных, вводимых в ЭВМ.

1.2 Проблемы при разработке программного обеспечения предприятия

В современных условиях вопрос качества поставляемого на рынок программного обеспечения становится все более актуальным — с ростом количества компаний-разработчиков, а также независимых коллективов программистов, готовых предоставить потребителям готовый продукт, заинтересованность в скорейшем выводе в продажу коммерческого решения возрастает. Часто в погоне за прибылью ключевые проблемы разработки ПО уходят на второй план — сроки одерживают верх в борьбе за высокое качество программных решений. Отметим основные факторы, оказывающие негативное влияние на качественные показатели выпускаемых на рынок программ.

Достаточно крупные коллективы разработчиков, планируя создание нового программного обеспечения, как правило, совершают первую ошибку уже на этапе формирования технического задания. Вопреки отработанной технологии разработки качественного ПО, заказчики не редко ограничивают сроки реализации проекта, вынуждая команду разработчиков исключать из перечня задач процесс отладки написанного программистами кода. Вместо оптимизации каждого компонента программного продукта, задача по “вылавливанию багов” возлагается на тестировщиков,

берущихся за работу на самом последнем этапе разработки ПО, когда все модули программы уже собраны воедино.

Чуть менее значимой ошибкой, ведущей к появлению проблем при разработке программного обеспечения, является игнорирование весьма нужной процедуры анализа формируемых заказчиком и исполнителем требований к будущей программе. Нечеткое понимание целей, преследуемых заказчиком, а также несогласованность бизнес-деталей, подлежащих реализации в продукте, ведет к сдаче низкокачественной программы, имеющей существенное количество недоработок. Особенно остро эта проблема стоит в сфере разработки мобильных приложений, где ключевым фактором для заказчиков становится как можно более ранний вывод решения на рынок, чтобы успеть заработать деньги от продажи программы потребителям.

Пути решения. Чтобы избежать упомянутых проблем при разработке программного обеспечения, и заказчики, и исполнители, будь то крупная компания или небольшой коллектив программистов, должны придерживаться методики обеспечения качества ПО, состоящей всего из нескольких пунктов.

1. Анализ требований. Еще на этапе формирования технического задания по разработке ПО требуется согласовать ключевые вопросы, связанные с механикой работы и составом ключевых компонентов программы. Также стороны должны прийти ко взаимному пониманию в вопросе функциональности создаваемого программного продукта, что позволит добиться принятия работы сразу после демонстрации рабочего образца программы.

2. Анализ и сквозной контроль кода. Контроль работоспособности программного кода, наличие ошибок и корректность их обработки, должны осуществляться постоянно, в течение всего процесса разработки ПО. Перекладывать необходимость поиска проблемных участков кода на плечи тестировщиков, занимающихся проверкой выполнения основных функций ПО уже после выполнения основных этапов разработки, категорически нельзя.

3. Сессионное тестирование. Методика сессионного тестирования, предложенная одним из ведущих специалистов в области программирования Джеймсом Бахом, позволяет осуществлять качественную проверку работоспособности созданного решения. В отличие от технологии поиска “точечных” недоработок кода, при сессионном тестировании тестировщик получает свободу действий, пытаясь выявить необычные дефекты, фактически моделируя поведение предполагаемого пользователя.

Наиболее эффективным путем решения проблем при разработке ПО является обращение к профессиональным “аутсорсерам”, предоставляющим услуги IT-аутсорсинга в сегменте создания программного обеспечения. Ключевыми моментами становятся правильно составленное техническое задание, отражающее требования и потребности обеих сторон соглашения, а также установка наиболее оптимальных сроков исполнения заказа. При этом разработчики обязаны доказать обоснованность продления сроков реализации проекта в случае необходимости, чтобы достичь высокого качества конечного продукта. Только в этом случае качество одержит победу над сроками.

1.3 Сущность жизненного цикла

У каждого продукта, который выходит на рынок, всегда есть определенная продолжительность жизни. Он в какой-то момент устаревает и на смену ему приходит более современный и удобный аналог. Этот процесс называют жизненным циклом товара. В статье мы рассмотрим концепцию ЖЦТ, ее основные стадии и типы, а также разберем преимущества и недостатки использования этой концепции.

Жизненный цикл товара — это период времени, на протяжении которого продукт находится на рынке: внедрение товара, насыщение рынка, конец реализации.

Другими словами, ЖЦТ — это период существования продукта на рынке.

Суть концепции: всякий продукт рано или поздно вытеснят с рынка другие, более совершенные товары. Эти условия применимы к виду, типу и классу продукта, а также к конкретной модели и торговой марке.

Жизненный цикл у всех продуктов разный — это зависит от типа товара и рынка, где его размещают. В современном мире можно заметить такую тенденцию ускорения продолжительности жизни продукта. На рынок каждый день поступают новые товары, которые дешевле или лучше своего предшественника.

Простой пример — это смартфоны, технические характеристики, которых улучшаются каждый год.

Классическая кривая стадий развития товара выглядит так: снизу — линия времени жизни продукта, слева — объем продаж. На этом графике два уровня — продажи и прибыль, а также их соотношение на каждом этапе. В следующих разделах мы разберем каждую стадию подробнее.

2. Процессы жизненного цикла

2.1 Основные процессы жизненного цикла

Одним из базовых понятий методологии проектирования ИС является понятие жизненного цикла ее программного обеспечения (ЖЦ ПО). ЖЦ ПО - это непрерывный процесс, который начинается с момента принятия решения о необходимости его создания и заканчивается в момент его полного изъятия из эксплуатации.

Основным нормативным документом, регламентирующим ЖЦ ПО, является международный стандарт ISO/IEC 12207 [5] (ISO - International Organization of Standardization - Международная организация по стандартизации, IEC - International Electrotechnical Commission - Международная комиссия по электротехнике). Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, которые должны быть выполнены во время создания ПО.

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

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

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

Эксплуатация включает в себя работы по внедрению компонентов ПО в эксплуатацию, в том числе конфигурирование базы данных и рабочих мест пользователей, обеспечение эксплуатационной документацией, проведение обучения персонала и т.д., и непосредственно эксплуатацию, в том числе локализацию проблем и устранение причин их возникновения, модификацию ПО в рамках установленного регламента, подготовку предложений по совершенствованию, развитию и модернизации системы.

Управление проектом связано с вопросами планирования и организации работ, создания коллективов разработчиков и контроля за сроками и качеством выполняемых работ. Техническое и организационное обеспечение проекта включает выбор методов и

инструментальных средств для реализации проекта, определение методов описания промежуточных состояний разработки, разработку методов и средств испытаний ПО, обучение персонала и т.п. Обеспечение качества проекта связано с проблемами верификации, проверки и тестирования ПО. Верификация - это процесс определения того, отвечает ли текущее состояние разработки, достигнутое на данном этапе, требованиям этого этапа. Проверка позволяет оценить соответствие параметров разработки с исходными требованиями. Проверка частично совпадает с тестированием, которое связано с идентификацией различий между действительными и ожидаемыми результатами и оценкой соответствия характеристик ПО исходным требованиям. В процессе реализации проекта важное место занимают вопросы идентификации, описания и контроля конфигурации отдельных компонентов и всей системы в целом.

Управление конфигурацией является одним из вспомогательных процессов, поддерживающих основные процессы жизненного цикла ПО, прежде всего процессы разработки и сопровождения ПО. При создании проектов сложных ИС, состоящих из многих компонентов, каждый из которых может иметь разновидности или версии, возникает проблема учета их связей и функций, создания унифицированной структуры и обеспечения развития всей системы. Управление конфигурацией позволяет организовать, систематически учитывать и контролировать внесение изменений в ПО на всех стадиях ЖЦ. Общие принципы и рекомендации конфигурационного учета, планирования и управления конфигурациями ПО отражены в проекте стандарта ISO 12207-2 [5].

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

2.2 Вспомогательные процессы жизненного цикла

Вспомогательные процессы жизненного цикла состоят из восьми процессов. Вспомогательный процесс является целенаправленной составной частью другого процесса, обеспечивающей успешную реализацию и качество выполнения программного проекта. Вспомогательный процесс, при необходимости, инициируется и используется другим процессом. Вспомогательными процессами являются:

1. Процесс документирования. Определяет работы по описанию информации, выдаваемой в процессе жизненного цикла.
2. Процесс управления конфигурацией. Определяет работы по управлению конфигурацией.
3. Процесс обеспечения качества. Определяет работы по объективному обеспечению того, чтобы программные продукты и процессы соответствовали требованиям, установленным для них, и реализовывались в рамках утвержденных планов. Совместные анализы, аудиторские проверки, верификация и аттестация могут использоваться в качестве методов обеспечения качества.
4. Процесс верификации. Определяет работы (заказчика, поставщика или независимой стороны) по верификации программных продуктов по мере реализации программного проекта.
5. Процесс аттестации. Определяет работы (заказчика, поставщика или независимой стороны) по аттестации программных продуктов программного проекта.
6. Процесс совместного анализа. Определяет работы по оценке состояния и результатов какой-либо работы. Данный процесс может использоваться двумя любыми

сторонами, когда одна из сторон (проверяющая) проверяет другую сторону (проверяемую) на совместном совещании.

7. Процесс аудита. Определяет работы по определению соответствия требованиям, планам и договору. Данный процесс может использоваться двумя сторонами, когда одна из сторон (проверяющая) контролирует программные продукты или работы другой стороны (проверяемой).

8. Процесс решения проблемы. Определяет процесс анализа и устранения проблем (включая несоответствия), независимо от их характера и источника, которые были обнаружены во время осуществления разработки, эксплуатации, сопровождения или других процессов.

2.3 Организационные процессы жизненного цикла

Организационные процессы жизненного цикла состоят из четырех процессов. Они применяются в какой-либо организации для создания и реализации основной структуры, охватывающей взаимосвязанные процессы жизненного цикла и соответствующий персонал, а также для постоянного совершенствования данной структуры и процессов. Эти процессы, как правило, являются типовыми, независимо от области реализации конкретных проектов и договоров; однако уроки, извлеченные из таких проектов и договоров, способствуют совершенствованию организационных вопросов. Организационными процессами являются:

1. Процесс управления. Определяет основные работы по управлению, включая управление проектом, при реализации процессов жизненного цикла.

2. Процесс создания инфраструктуры. Определяет основные работы по созданию основной структуры процесса жизненного цикла.

3. Процесс усовершенствования . Определяет основные работы, которые организация (заказчика, поставщика, разработчика, оператора, персонала сопровождения или администратора другого процесса) выполняет при создании, оценке, контроле и усовершенствовании выбранных процессов жизненного цикла.

4. Процесс обучения. Определяет работы по соответствующему обучению персонала.

3. Обзор методов разработки программного обеспечения

3.1 Интуитивный подход

Интуитивный подход базируется на интуиции, на ощущении того, что выбор сделан правильно. Интуиция (озарение) — способность постижения истины без обоснования с помощью логики. Интуитивный подход применяется для решения относительно несложных проблем, когда решение зависит от соответствия появившейся проблемной ситуации прошлым ситуациям. При интуитивном подходе используются личный опыт, проницательность в большей степени, чем последовательная логика или четкие умозаключения. Интуиция базируется на многолетней практике и здравом смысле, зачастую хранящихся в подсознании. Обращаясь к своей интуиции, основанной на многолетнем опыте решения проблем, менеджеры гораздо быстрее осознают, что в организации возникли проблемы и при этом интуитивно предчувствуют варианты решения проблем, что значительно сокращает время их решения (принятия решений). При этом чаще всего интуитивно реагируют на «проблемы — угрозы», чем на «проблемы — возможности» для организации.

3.2 Каскадный процесс

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

Положительные стороны применения каскадного подхода заключаются в следующем [2]:

- на каждом этапе формируется законченный набор проектной документации, отвечающий критериям полноты и согласованности;
- выполняемые в логичной последовательности этапы работ позволяют планировать сроки завершения всех работ и соответствующие затраты.

Каскадный подход хорошо зарекомендовал себя при построении ИС, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования, с тем чтобы предоставить разработчикам свободу реализовать их как можно лучше с технической точки зрения. В эту категорию попадают сложные расчетные системы, системы реального времени и другие подобные задачи. Однако, в процессе использования этого подхода обнаружился ряд его недостатков, вызванных прежде всего тем, что реальный процесс создания ПО никогда полностью не укладывался в такую жесткую схему. В процессе создания ПО постоянно возникала потребность в возврате к предыдущим этапам и уточнении или пересмотре ранее принятых решений. В результате реальный процесс создания ПО принимал следующий вид (рис. 1.1):

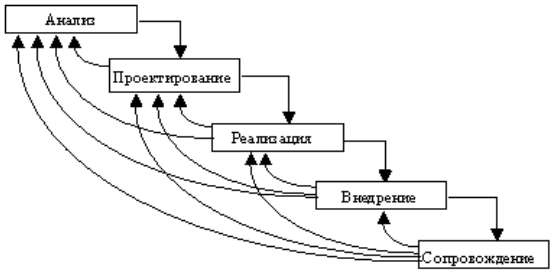


Рис. 1.1. Реальный процесс разработки ПО по каскадной схеме

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

3.3 Рациональный унифицированный процесс

Рациональный унифицированный процесс разработки программного обеспечения (*RUP – Rational Unified Process*) является частным случаем *унифицированного процесса* (*UP – Unified Process*). В основу рационального унифицированного процесса положена итеративная разработка программного обеспечения. В рамках RUP разработка выполняется в виде нескольких краткосрочных итераций продолжительностью от 2 до 6 недель. Итерация по существу является мини-проектом фиксированной длительности, в результате которой расширяется и дополняется функциональность разрабатываемой системы. Поэтому унифицированный процесс разработки иногда называют итеративной и инкрементальной разработкой**.**

В результате каждой итерации получается работающая, но не полнофункциональная система, которая еще не является коммерческой и не подлежит распространению. Продолжительность создания коммерческой версии программной системы составляет 10 – 15 итераций.

Но результат каждой итерации нельзя рассматривать и в виде прототипа системы. Правильнее сказать, что в результате каждой итерации создается окончательная версия некоторой части всех системы.

Следуют так же заметить, что, не смотря на то, что, как правило, на каждой итерации определяются и реализуются новые требования к системе, некоторые итерации могут быть целиком посвящены усовершенствованию существующей программы, например с целью повышения ее производительности.

Унифицированный процесс допускает внесение изменений требований пользователей к создаваемой программной системе. Таким образом, он является адаптивным процессом. Это достигается за счет итеративному процессу разработки и наличию ранней обратной связи. Благодаря обратной связи заказчик может оценить часть системы и высказать некоторые предложения по внесению изменений в ее функциональность. Здесь речь не идет о том, что функциональность совершенно не устраивает заказчика или пользователей, просто могут возникнуть идеи об ее улучшении или же возникнуть новая ситуация под которую необходимо адаптировать создаваемую систему. Таким образом, реализуется эволюционный процесс, в результате которого разрабатываемая система постоянно улучшается и все больше удовлетворяет требованиям пользователей.

3.4 Экстремальное программирование и разработка по функциям

Экстремальное программирование (XP) – это упрощенная методология организации разработки программ для небольших и средних по размеру команд разработчиков, занимающихся созданием программного продукта в условиях неясных или быстро меняющихся требований.

Цели XP. Основными целями XP являются повышение доверия заказчика к программному продукту путем предоставления реальных доказательств успешности развития процесса разработки и резкое сокращение сроков разработки продукта. При этом XP сосредоточено на минимизации ошибок на ранних стадиях разработки. Это позволяет добиться максимальной скорости выпуска готового продукта и даёт возможность говорить о прогнозируемости работы. Практически все приемы XP направлены на повышение качества программного продукта.

Принципы XP. Основными принципами являются:

Итеративность. Разработка ведется короткими итерациями при наличии активной взаимосвязи с заказчиком. Итерации как таковые предлагается делать короткими,

рекомендуемая длительность – 2-3 недели и не более 1 месяца. За одну итерацию группа программистов обязана реализовать несколько свойств системы, каждое из которых описывается в пользовательской истории. Пользовательские истории (ПИ) в данном случае являются начальной информацией, на основании которой создается модуль. Они отличаются от вариантов использования (ВИ). Описание ПИ короткое – 1-2 абзаца, тогда как ВИ обычно описываются достаточно подробно, с основным и альтернативными потоками, и дополняются моделью. ПИ пишутся самими пользователями, которые в XP являются частью команды, в отличие от ВИ, которые описывает системный аналитик. Отсутствие формализации описания входных данных проекта в XP стремятся компенсировать за счет активного включения в процесс разработки заказчика как полноправного члена команды.

Простота решений. Принимается первое простейшее рабочее решение. Экстремальность метода связана с высокой степенью риска решения, обусловленного поверхностностью анализа и жестким временным графиком. Реализуется минимальный набор главных функций системы на первой и каждой последующей итерации; функциональность расширяется на каждой итерации.

Интенсивная разработка малыми группами (не больше 10 человек) и парное программирование (когда два программиста вместе создают код на одном общем рабочем месте), активное общение в группе и между группами. Все это нацелено на как можно более раннее обнаружение проблем (как ошибок, так и срыва сроков). Парное программирование направлено на решение задачи стабилизации проекта. При применении XP методологии высок риск потери кода по причине ухода программиста, не выдержавшего интенсивного графика работы. В этом случае второй программист из пары играет роль «наследника» кода. Немаловажно и то, как именно распределены группы в рабочем пространстве – в XP используется открытое рабочее пространство, которое предполагает быстрый и свободный доступ всех ко всем.

Обратная связь с заказчиком, представитель которого фактически вовлечен в процесс разработки.

Достаточная степень смелости и желание идти на риск.

4. Структурный подход к проектированию программного обеспечения

4.1 Сущность структурного подхода

Сущность структурного подхода к разработке ИС заключается в её декомпозиции (разбиении) на автоматизируемые функции. Система разбивается на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи и так далее. Процесс разбиения продолжается вплоть до конкретных процедур. При этом автоматизируемая система сохраняет целостное представление, в котором все составляющие компоненты взаимоувязаны. При разработке системы “снизу-вверх” от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов.

В методологии структурного подхода используют принципы:

“разделяй и властвуй” – решение сложных проблем путём их разбиения на множество меньших независимых задач, легких для понимания и решения;

иерархического упорядочивания – организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне;

абстрагирования – выделение существенных аспектов системы и отвлечения от несущественных;

формализации – необходимость строгого методического подхода к решению проблемы;

непротиворечивости – обоснование и согласованность элементов;

структурирования данных – данные должны быть структурированы и иерархически организованы.

На стадии проектирования ИС модели расширяются, уточняются и дополняются диаграммами, отражающими структуру ПО: архитектуру ПО, структурные схемы программ и диаграммы экранных форм. Перечисленные модели в совокупности дают полное описание ИС независимо от того, является ли она существующей или вновь разрабатываемой. Состав диаграмм в каждом конкретном случае зависит от необходимой полноты описания системы.

Структурное (системное) проектирование – это метод определения подсистем, компонентов и способов их соединения, задающий ограничения, при которых система должна функционировать, выбирающий наиболее эффективное сочетание людей, машин и программного обеспечения для реализации системы.

В качестве примера рассмотрим одну из широко используемых систем такого проектирования – SADT (Structured Analysis and Design Technique – технология структурного анализа и проектирования) – это графические обозначения и подход к описанию систем.

SADT создана для описания системы и её среды до определения требований к программному обеспечению и др. Она облегчает описание и понимание искусственных систем средней сложности. В SADT используется графический язык и набор процедур анализа для понимания системы прежде, чем можно представить себе её воплощение. Она, как правило, применяется на ранних этапах процесса создания системы, который часто называют “жизненным циклом системы”.

SADT-модель – иерархически организованная совокупность диаграмм. Диаграммы обычно состоят из трёх-шести блоков, каждый из которых потенциально может быть детализирован на другой диаграмме. Каждая диаграмма представляет некоторую законченную часть всей модели.

Каждый блок может пониматься как отдельный тщательно определённый объект. Разделение такого объекта на его структурные части (блоки и дуги, составляющие диаграмму) называется декомпозицией.

Декомпозиция формирует границы, и каждый блок в SADT рассматривается как формальная граница некоторой части целой системы, которая описывается. Блок и касающиеся его дуги определяют точную границу диаграммы, представляющей декомпозицию этого блока. Эта диаграмма, называемая диаграммой с потомком, описывает всё, связанное с этим блоком и его дугами, и не описывает ничего вне этой границы. Декомпозируемый блок называется родительским блоком, а содержащая его диаграмма – соответственно родительской диаграммой.

Блок изображает границу системы: всё, лежащее внутри него, является частью описываемой системы, а всё, лежащее вне него образует среду системы.

Процесс моделирования в SADT включает сбор информации об исследуемой области, документирование полученной информации, представление её в виде модели и уточнение модели посредством итеративного рецензирования. Кроме того, этот процесс подсказывает вполне определённый путь выполнения согласованной и достоверной структурной декомпозиции, что является ключевым моментом в квалифицированном анализе системы.

Модель в SADT редко создается одним автором. Над различными частями модели могут совместно работать множество разработчиков, так как каждый функциональный блок модели представляет отдельный субъект, который может быть независимо проанализирован и декомпозирован.

4.2 Метод функционального моделирования

Метод SADT разработан Дугласом Россом в 1973 г. Данный метод успешно использовался в военных, промышленных и коммерческих организациях США для решения широкого круга задач, таких, как долгосрочное и стратегическое планирование, автоматизированное производство и проектирование, разработка ПО для оборонных систем, управление финансами и материально-техническим снабжением и др. Метод SADT представляет собой совокупность правил и процедур, предназначенных для построения функциональной модели объекта какой-либо предметной области. Функциональная модель SADT отображает функциональную структуру объекта, т.е. производимые им действия и связи между этими действиями. Основные элементы этого метода основываются на следующих концепциях:

· графическое представление блочного моделирования. Графика блоков и дуг SADT-диаграммы отображает функцию в виде блока, а интерфейсы входа-выхода представляются дугами, соответственно входящими в блок и выходящими из него. Взаимодействие блоков друг с другом описывается посредством интерфейсных дуг, выражающих «ограничения», которые, в свою очередь, определяют, когда и каким образом функции выполняются и управляются;

· строгость и точность. Выполнение правил SADT требует достаточной строгости и точности, не накладывая в то же время чрезмерных ограничений на действия аналитика. Правила SADT включают: ограничение количества блоков на каждом уровне декомпозиции (правило 3-6 блоков), связность диаграмм (номера блоков), уникальность меток и наименований (отсутствие повторяющихся имен), синтаксические правила для графики (блоков и дуг), разделение входов и управлений (правило определения роли данных);

· отделение организации от функции, т.е. исключение влияния административной структуры организации на функциональную модель.

Метод SADT может использоваться для моделирования самых разнообразных систем и определения требований и функций с последующей разработкой информационной системы, удовлетворяющей этим требованиям и реализующей эти функции. В существующих системах метод SADT может применяться для анализа функций, выполняемых системой, и указания механизмов, посредством которых они осуществляются.

Результатом применения метода SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга. Диаграммы – главные компоненты модели, все функции организации и интерфейсы на них представлены как блоки и дуги соответственно. Место соединения дуги с блоком определяет тип интерфейса. Управляющая информация входит в блок сверху, в то время как входная информация, которая подвергается обработке, показана с левой стороны блока, а результаты (выход) показаны с правой стороны. Механизм (человек или автоматизированная система), который осуществляет операцию, представляется дугой, входящей в блок снизу.

Одной из наиболее важных особенностей метода SADT является постепенное введение все больших уровней детализации по мере создания диаграмм, отображающих модель.

4.3 Состав функциональной модели

Начало разработки диаграмм функционального моделирования относится к середине 1960-х годов, когда Дуглас Т. Росс предложил специальную технику моделирования, получившую название SADT (Structured Analysis & Design Technique). Военно-воздушные силы США использовали методику SADT в качестве части своей программы интеграции компьютерных и промышленных технологий (Integrated Computer Aided Manufacturing,

ICAM) и назвали ее IDEFO (Icam DEFinition). Целью программы ICAM было увеличение эффективности компьютерных технологий в сфере проектирования новых средств вооружений и ведения боевых действий. Одним из результатов этих исследований являлся вывод о том, что описательные языки не эффективны для документирования и моделирования процессов функционирования сложных систем. Подобные описания на естественном языке не обеспечивают требуемого уровня непротиворечивости и полноты, имеющих доминирующее значение при решении задач моделирования.

В рамках программы ICAM было разработано несколько графических языков моделирования, которые получили следующие названия:

Нотация IDEF0 - для документирования процессов производства и отображения информации об использовании ресурсов на каждом из этапов проектирования систем.

Нотация IDEF1 - для документирования информации о производственном окружении систем.

Нотация IDEF2 - для документирования поведения системы во времени.

Нотация IDEF3 - специально для моделирования бизнес-процессов.

Нотация IDEF2 никогда не была полностью реализована. Нотация IDEF1 в 1985 году была расширена и переименована в IDEF1X. Методология IDEF-SADT, нашла применение в правительственных и коммерческих организациях, поскольку на тот период времени вполне удовлетворяла различным требованиям, предъявляемым к моделированию широкого класса систем.

В начале 1990 года специально образованная группа пользователей IDEF (IDEF Users Group), в сотрудничестве с Национальным институтом по стандартизации и технологии США (National Institutes for Standards and Technology, NIST), предприняла попытку создания стандарта для IDEFO и IDEF1X. Эта попытка оказалась успешной и завершилась принятием в 1993 году стандарта правительства США, известного как FIPS для данных двух технологий IDEFO и IDEF1X. В течение последующих лет этот стандарт продолжал активно развиваться и послужил основой для реализации в некоторых первых CASE-средствах.

Методология IDEF-SADT представляет собой совокупность методов, правил и процедур, предназначенных для построения функциональной модели системы какой-либо предметной области. Функциональная модель SADT отображает структуру процессов функционирования системы и ее отдельных подсистем, т. е. выполняемые ими действия и связи между этими действиями. Для этой цели строятся специальные модели, которые позволяют в наглядной форме представить последовательность определенных действий. Исходными строительными блоками любой модели IDEFO процесса являются деятельность (activity) и стрелки (arrows).

Рассмотрим кратко эти основные понятия методологии IDEF-SADT, которые используются при построении диаграмм функционального моделирования.

Деятельность представляет собой некоторое действие или набор действий, которые имеют фиксированную цель и приводят к некоторому конечному результату. Иногда деятельность называют просто процессом. Модели IDEFO отслеживают различные виды деятельности системы, их описание и взаимодействие с другими процессами. На диаграммах деятельность или процесс изображается прямоугольником, который называется блоком. Стрелка служит для обозначения некоторого носителя или воздействия, которые обеспечивают перенос данных или объектов от одной деятельности к другой. Стрелки также необходимы для описания того, что именно производит деятельность и какие ресурсы она потребляет. Это так называемые роли стрелок - ICOM - сокращение первых букв от названий соответствующих стрелок IDEFO. При этом различают стрелки четырех видов:

I (Input) - вход, т. е. все, что поступает в процесс или потребляется процессом.

C (Control) - управление или ограничения на выполнение операций процесса.

O (Output) - выход или результат процесса.

M (Mechanism) - механизм, который используется для выполнения процесса.

Пример использования метода

Методология IDEFO однозначно определяет, каким образом изображаются на диаграммах стрелки каждого вида ICOM. Стрелка Вход (Input) выходит из левой стороны рамки рабочего поля и входит слева в прямоугольник процесса. Стрелка Управление (Control) входит и выходит сверху. Стрелка Выход (Output) выходит из правой стороны процесса и входит в правую сторону рамки. Стрелка Механизм (Mechanism) входит в прямоугольник процесса снизу. Таким образом, базовое представление процесса на диаграммах IDEFO имеет следующий вид (рис. 4.1).

Техника построения диаграмм представляет собой главную особенность методологии IDEF-SADT. Место соединения стрелки с блоком определяет тип интерфейса. При этом все функции моделируемой системы и интерфейсы на диаграммах представляются в виде соответствующих блоков процессов и стрелок ICOM. Управляющая информация входит в блок сверху, в то время как информация, которая подвергается обработке, изображается с левой стороны блока. Результаты процесса представляются как выходы процесса и показываются с правой стороны блока. В качестве механизма может выступать человек или автоматизированная система, которые реализуют данную операцию. Соответствующий механизм на диаграмме представляется стрелкой, которая входит в блок процесса снизу.

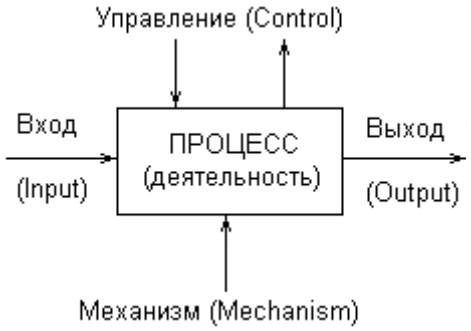


Рис. 4.1. Обозначение процесса и стрелок ICOM на диаграммах IDEF0

Одной из наиболее важных особенностей методологии IDEF-SADT является постепенное введение все более детальных представлений модели системы по мере разработки отдельных диаграмм. Построение модели IDEF-SADT начинается с представления всей системы в виде простейшей диаграммы, состоящей из одного блока процесса и стрелок ICOM, служащих для изображения основных видов взаимодействия с объектами вне системы. Поскольку исходный процесс представляет всю систему как единое целое, данное представление является наиболее общим и подлежит дальнейшей декомпозиции.

Для иллюстрации основных идей методологии IDEF-SADT рассмотрим следующий простой пример. В качестве процесса будем представлять деятельность по оформлению кредита в банке. Входом данного процесса является заявка от клиента на получение кредита, а выходом - соответствующий результат, т. е. непосредственно кредит. При этом управляющими факторами являются правила оформления кредита, которые

регламентируют условия получения соответствующих финансовых средств в кредит. Механизмом данного процесса является служащий банка, который уполномочен выполнить все операции по оформлению кредита. Пример исходного представления процесса оформления кредита в банке изображен на рис. 4.2.

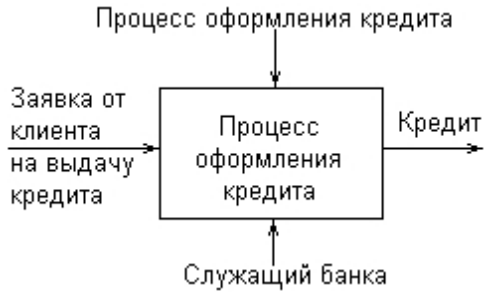


Рис. 4.2. Пример исходной диаграммы IDEF-SADT для процесса оформления кредита в банке

В конечном итоге модель IDEF-SADT представляет собой серию иерархически взаимосвязанных диаграмм с сопроводительной документацией, которая разбивает исходное представление сложной системы на отдельные составные части. Детали каждого из основных процессов представляются в виде более детальных процессов на других диаграммах. В этом случае каждая диаграмма нижнего уровня является декомпозицией некоторого процесса из более общей диаграммы. Поэтому на каждом шаге декомпозиции более общая диаграмма конкретизируется на ряд более детальных диаграмм.

В настоящее время диаграммы структурного системного анализа IDEF-SADT продолжают использоваться целым рядом организаций для построения и детального анализа функциональной модели существующих на предприятии бизнес-процессов, а также для разработки новых бизнес-процессов. Основной недостаток данной методологии связан с отсутствием явных средств для объектно-ориентированного представления моделей сложных систем. Хотя некоторые аналитики отмечают важность знания и применения нотации IDEF-SADT, ограниченные возможности этой методологии применительно к реализации соответствующих графических моделей в объектно-ориентированном программном коде существенно сужают диапазон решаемых с ее помощью задач.

5. Архитектура программного обеспечения

5.1 Определение программной архитектуры

Архитектура — это структура программы или вычислительной системы, определяющая ее работу на самом высоком концептуальном уровне, включая аппаратные и программные компоненты, видимые снаружи свойства этих компонентов, отношения между ними, а также документирование системы. Документирование архитектуры упрощает процесс взаимодействия между участниками проекта, позволяет зафиксировать принятые на ранних этапах проектирования решения о высокоуровневом дизайне системы и использовать элементы этого дизайна и шаблоны повторно в других проектах.

Для разработки архитектуры системы привлекаются специалисты со следующими ролями: системный архитектор (проектирует систему в целом, а также отдельные ее компоненты), архитектор базы данных (занимается проектированием БД и ее структуры), системный аналитик (участвует в проектировании, подготавливает

документацию), администраторы (участвуют в проектировании аппаратной части системы).

На архитекторов системы возлагается большая ответственность. Если разработанная архитектура не будет реализовывать поставленные заказчиком цели, то это может, например, увеличить сроки выполнения проекта (за счет того, что необходимо будет делать доработки по исправлению недостатков архитектуры), а следовательно, снизить прибыль за разработку.

5.2 Необходимость программной архитектуры

На уровне разработки архитектуры приложения должны решаться следующие основные задачи, важные для заказчика.

Улучшение и повышение продуктивности процессов. Типичными ожиданиями заказчика от внедрения приложения являются: уменьшение времени, затрачиваемого на выполнение различных действий; ускорение выполнения различных операций; автоматизация процессов; различные улучшения, получаемые за счет масштабируемости — способности системы, сети или процесса справляться с увеличением рабочей нагрузки (увеличивать свою производительность) при добавлении ресурсов, обычно аппаратных.

Уменьшение затрат. Одной из целей разработки может стать уменьшение затрат, необходимых при совершении каких-либо действий. Это может осуществляться как за счет повышения продуктивности процессов, так и за счет ускорения выполнения операций.

Улучшение операционной деятельности. Операционная деятельность обычно связана с выполнением рутинных типовых операций (например, работа кассира в магазине, прием коммунальных платежей и т.д.). Автоматизируя (упрощая, ускоряя) такую операционную деятельность, можно снижать затраты либо увеличивать производительность системы.

Повышение эффективности управления. Архитектурное решение может иметь целью повышение эффективности управления, например, за счет автоматизации документооборота на предприятии (переход от бумажных документов к электронным с отслеживанием истории изменения, уведомлениями и пр.).

Уменьшение рисков. Любая деятельность связана с определенными рисками. Одной из целей разработки приложения может быть их снижение. Например, правило двойной подписи для финансовых операций (когда финансовую операцию, созданную одним сотрудником, обязательно должен проверить другой сотрудник и поставить свою подпись).

Повышение эффективности IT-подразделений. Этот результат достигается за счет автоматизации различных процессов.

Повышение продуктивности работы пользователей. Под пользователями можно понимать как сотрудников самой компании (в этом случае повышение продуктивности можно отнести к целям, затрагивающим процессы), так и клиентов компании, которые будут пользоваться разработанным ПО (чем комфортней клиентам, тем меньше вероятность, что они перейдут к конкурентам).

Повышение возможности и прозрачности взаимодействия. На многих предприятиях используют по несколько систем, между которыми необходимо осуществлять обмен информацией. Разработка ПО может быть направлена на автоматизацию и упрощение данного обмена (создание его более «прозрачным», простым для конечных пользователей).

Уменьшение стоимости «поддержки» жизненного цикла. Процессы, связанные с ЖЦ ПП, также могут быть целью автоматизации, так как снижение затрат, осуществляемых в процессе ЖЦ ПП, приведет к дополнительной прибыли.

Улучшение характеристик безопасности. Безопасность приложений с каждым годом становится все более актуальной. Более «безопасные» приложения обладают большей конкурентоспособностью по сравнению с аналогами.

Повышение управляемости. Под управляемостью понимается влияние на различные процессы, происходящие в приложении без вмешательства разработчика.

6. Объектно-ориентированный подход к проектированию программного обеспечения

6.1 Сущность объектно-ориентированного подхода

Объектно-ориентированное программирование – это технология программирования, при которой программа рассматривается как набор дискретных объектов, содержащих, в свою очередь, наборы структур данных и процедур, взаимодействующих с другими объектами.

Поведение или функционирование объекта определяется как последовательная смена состояний. Состояние объекта, параметры которого не испытывают воздействий, остается неизменным. Внешняя среда для объекта или системы объектов определяется как множество существующих вне объекта (системы) элементов любой природы, оказывающих влияние на объект (систему) или находящихся под его (её) воздействием.

Если существует некоторое множество объектов, находящихся во взаимодействии друг с другом, то для каждого из них все прочие объекты из этого множества являются частью внешней среды. Объекты воздействуют друг на друга посредством обмена сообщениями.

Сущность объектно-ориентированного подхода

Суть объектно-ориентированного подхода (ООП) состоит в том, что проектируются не данные и программы в отдельности, а объекты, сочетающие в себе и данные, и программы, информационно и функционально характеризующие соответствующие сущности предметной области.

Основное преимущество ООП – возможность создавать классы и объекты визуальным способом, т.е. прорисовывать на экране основные элементы, определять цвет, местоположение элементов и т.д. При определённом навыке объекты можно быстро создавать, записывая в методы фрагменты программного кода, определяющие их поведение при наступлении определённых событий. В дальнейшем из визуальных элементов и этих программных фрагментов генерируется общая программа. Этим занимается сама система.

Подход полезен как с методической точки зрения (две разнородные характеристики предметной области – данные и программы – объединяются в объекты), так и с точки зрения техники проектирования и разработки программных систем (вместо двух технически не связанных, но логически переплетенных веток образуется один надёжный ствол).

На различных этапах анализа и синтеза систем возникают проблемы разбиения (декомпозиции) системы на подсистемы, задачи на подзадачи, программного обеспечения на отдельные программы и подпрограммы. При этом объекты каждого последующего уровня разбиения представляют собой абстрактные компоненты (объекты) системы предыдущего уровня, реализация которого зависит от конкретной рассматриваемой проблемы.

В объектно-ориентированных системах декомпозиция системы на объекты осуществляется с учётом удобства последующего детального анализа, разработки и внедрения системы. Одним из наиболее важных критериев выделения компонентов системы является минимизация числа аппаратно-зависимых её компонент. Это позволяет снизить затраты на адаптацию системы при переносе на другую аппаратную платформу, а также уменьшить количество неиспользуемых компонент при работе на конкретной

платформе. Решение этой проблемы осуществляется путём исследования существующих платформ, оценки направлений их развития, анализа возможностей использования принятых и (или) предложения новых стандартов взаимодействия системы с аппаратной платформой.

На основе декомпозиции системы:

выделяют задачи, подлежащие автоматизации;

определяют необходимое множество процедур реализации заданного множества функциональных задач и необходимой для этого информации;

осуществляют предварительную оценку уровня стандартизации используемых алгоритмов и интерфейсов.

Использование объектно-ориентированного подхода позволяет свести проектирование системы к оптимальному синтезу функционально независимых компонент (объектов), совместно выполняющих заданные функции системы. Таким образом, значительно снижаются затраты на разработку, внедрение и модификацию систем.

В пользу объектно-ориентированного подхода (ООП) говорит большое количество успешно реализованных систем различной природы, спроектированных по этому принципу. Он породил создание распределённой среды обработки данных, включающей системы обработки данных, информации и знаний.

Под распределённой обработкой данных понимают обработку приложений несколькими территориально разделёнными ЭВМ. При этом в приложениях, связанных с обработкой базы данных, собственно управление базой данных может выполняться централизованно.

6.2 Обзор языка моделирования UML

Отдельные языки объектно-ориентированного моделирования стали появляться в период между серединой 1970-х и концом 1980-х годов, когда различные исследователи и программисты предлагали свои подходы к ООАП. В период между 1989-1994 гг. общее число наиболее известных языков моделирования возросло с 10 до более чем 50. Многие пользователи испытывали серьезные затруднения при выборе языка ООАП (ООАП - объектно-ориентированный анализ и проектирование), поскольку ни один из них не удовлетворял всем требованиям, предъявляемым к построению моделей сложных систем. Принятие отдельных методик и графических нотаций в качестве стандартов (IDEF0, IDEF1X) не смогло изменить сложившуюся ситуацию непримиримой конкуренции между ними в начале 90-х годов, которая тоже получила название "войны методов".

К середине 1990-х некоторые из методов были существенно улучшены и приобрели самостоятельное значение при решении различных задач ООАП. Наиболее известными в этот период становятся:

Метод Гради Буча (Grady Booch), получивший условное название Booch или Booch'91, Booch Lite (позже - Booch'93).

Метод Джеймса Румбаха (James Rumbaugh), получивший название Object Modeling Technique - OMT (позже - OMT-2).

Метод Айвара Джекобсона (Ivar Jacobson), получивший название Object-Oriented Software Engineering - OOSE.

Каждый из этих методов был ориентирован на поддержку отдельных этапов ООАП.

История развития языка UML берет начало с октября 1994 года, когда в Rational Software Corporation начали работу по унификации методов Booch и OMT, в которую затем были включены и другие методы. В январе 1997 года был опубликован документ с описанием языка UML 1.0, как начальный вариант ответа на запрос предложений RTP.

В настоящее время все вопросы дальнейшей разработки языка UML сконцентрированы в рамках консорциума OMG. Соответствующая группа специалистов обеспечивает публикацию материалов, содержащих описание последующих версий языка UML и запросов предложений RFP по его стандартизации. Очередной этап развития данного языка закончился в марте 1999 года, когда консорциумом OMG было опубликовано описание языка UML 1.3 (alpha R5). Версией языка UML, рассматриваемой в данных лекциях, является UML 1.3 (июнь 1999 г.).

Статус языка UML определен как открытый для всех предложений по его доработке и совершенствованию. Сам язык UML не является чьей-либо собственностью и не запатентован кем-либо, хотя указанный выше документ защищен законом об авторском праве. В то же время аббревиатура UML, как и некоторые другие (OMG, CORBA, ORB), является торговой маркой их законных владельцев, о чем следует упомянуть в данном контексте.

Язык UML ориентирован для применения в качестве языка моделирования различными пользователями и научными сообществами для решения широкого класса задач ООАП. Многие специалисты по методологии, организации и поставщики инструментальных средств обязались использовать язык в своих разработках, причем для широкого класса систем: не только программного обеспечения, но и бизнес-процессов.

На основе технологии UML Microsoft, Rational Software и другие поставщики средств разработки программных систем разработали единую информационную модель, которая получила название UML Information Model. Предполагается, что эта модель даст возможность различным программам, поддерживающим идеологию UML, обмениваться между собой компонентами и описаниями. Все это позволит создать стандартный интерфейс между средствами разработки приложений и средствами визуального моделирования.

Уже в настоящее время разработаны средства визуального программирования на основе UML, обеспечивающие интеграцию, включая прямую и обратную генерацию кода программ, с наиболее распространенными языками и средами программирования, такими как MS Visual C++, Java, Object Pascal/Delphi, Power Builder, MS Visual Basic, Forte, Ada, Smalltalk.

Общая структура языка UML:

Формальное описание языка UML основывается на некоторой общей иерархической структуре модельных представлений, состоящей из четырех уровней:

Мета-метамодель

Метамодель

Модель

Объекты пользователя

Уровень мета-метамодели образует исходную основу для всех метамодельных представлений. Главное предназначение этого уровня состоит в том, чтобы определить язык для спецификации метамодели. Мета-метамодель определяет модель языка UML на самом высоком уровне абстракции и является наиболее компактным ее описанием. С другой стороны, мета-метамодель может специфицировать несколько метамоделей, чем достигается потенциальная гибкость включения дополнительных понятий. Примерами понятий этого уровня служат метакласс, метаатрибут, метаоперация.

Это важно не столько для программистов, сколько для разработчиков инструментальных средств

Метамодель является экземпляром или конкретизацией мета-метамодели. Главная задача этого уровня - определить язык для спецификации моделей. Данный уровень является более конструктивным, чем предыдущий, поскольку обладает более развитой семантикой базовых понятий. Все основные понятия языка UML - это понятия уровня метамодели. Примеры таких понятий - класс, атрибут, операция, компонент, ассоциация и многие другие. Метамодель UML является по своей сути скорее логической моделью,

опуская детали конкретной физической реализации. Метамодель языка UML имеет довольно сложную структуру, которая включает в себя порядка 90 метаклассов, более 100 метаассоциаций и почти 50 стереотипов, число которых возрастает с появлением новых версий языка. Чтобы справиться с этой сложностью языка UML, все его элементы организованы в логические пакеты. Поэтому рассмотрение языка UML на метамодельном уровне заключается в описании трех его наиболее общих логических блоков или пакетов: основные элементы, элементы поведения и общие механизмы.

Модель в контексте языка UML является экземпляром метамодели в том смысле, что любая конкретная модель системы должна использовать только понятия метамодели, конкретизировав их применительно к данной ситуации. Это уровень для описания информации о конкретной предметной области. Примерами понятий уровня модели могут служить, например, имена полей проектируемой базы данных, такие как имя и фамилия сотрудника, возраст, должность, адрес, телефон. При этом данные понятия используются лишь как имена соответствующих информационных атрибутов.

Объекты: Конкретизация понятий модели происходит на уровне объектов. В настоящем контексте объект является экземпляром модели, поскольку содержит конкретную информацию относительно того, чему в действительности соответствуют те или иные понятия модели. Примером объекта может служить следующая запись в проектируемой базе данных: "Илья Петров, 30 лет, иллюзионист, ул. Невидимая, 10-20, 100-0000".

6.3 Состав объектной модели

К основным понятиям объектно-ориентированного подхода (элементам объектной модели) относятся:

- объект;
- класс;
- атрибут;
- операция;
- полиморфизм (интерфейс);
- компонент;
- связи.

Объект может представлять собой абстракцию некоторой сущности предметной области (объект реального мира) или программной системы (архитектурный объект). Любой объект обладает состоянием (state), поведением (behavior) и индивидуальностью (identity).

Состояние объекта – одно из возможных условий, в которых он может существовать, оно изменяется со временем.

Состояние объекта характеризуется перечнем всех возможных (статических) свойств данного объекта и текущими значениями (динамическими) каждого из этих свойств. Состояние объекта определяется значениями его свойств (атрибутов) и связями с другими объектами.

Поведение определяет действия объекта и его реакцию на запросы от других объектов.

Поведение характеризует воздействие объекта на другие объекты, изменяющее их состояние. Иначе говоря, поведение объекта полностью определяется его действиями. Поведение представляется с помощью набора сообщений, воспринимаемых объектом (операций, которые может выполнять объект).

Каждый объект обладает уникальной индивидуальностью.

Индивидуальность – это свойства объекта, отличающие его от всех других объектов.

Структура и поведение схожих объектов определяют общий для них класс. Термины «экземпляр класса» и «объект» эквивалентны.

6.4 Прикладное программное обеспечение для проектирования автоматизированных систем обработки данных

Прикладное программное обеспечение (ППО) HMI (от англ. human-machine interface – человеко-машинный интерфейс) – программы, обеспечивающие взаимодействие человека-оператора с управляемыми им машинами.

Основные разновидности ППО HMI:

SCADA (от англ. Supervisory Control And Data Acquisition) – для технически сложных/крупных объектов;

Локальная панель оператора – для объектов локальной автоматизации, небольших объектов;

Web-диспетчеризация – для удаленного управления объектами через интернет.

При использовании SCADA взаимодействие с оператором реализуется через автоматизированные рабочие места.

При использовании панелей оператора – взаимодействие реализуется через специальные сенсорные панели.

Web - диспетчеризация – взаимодействие реализуется посредством облачных технологий. Это позволяет с любого устройства (ПК, планшет, смартфон) из любой точки мира видеть состояние всех систем через интернет-браузер. А также управлять оборудованием, получать уведомления по SMS и E-mail.

Функции ППО HMI:

- Визуализация работы исполнительных механизмов и датчиков объекта управления;
- Отображение мнемосхем технологических процессов;
- Отображение оперативных параметров;
- Ввод параметров процесса и оперативное управление с помощью функциональных кнопок и/или сенсорного экрана;
- Выполнение команд по расписанию;
- Сбор и архивирование данных с программируемых логических контроллеров, датчиков, исполнительных механизмов и другого оборудования;
- Отображение аварийных, предупреждающих, информирующих сообщений, а также подсказок для операторов;
- Отображение диаграмм и трендов по историческим данным;
- Формирование отчетов по работе системы;
- Защита и шифрование данных от несанкционированного доступа.

7. Анализ требований клиента

7.1 Идентификация исполнителей

Анализ требований — часть процесса разработки программного обеспечения, включающая в себя сбор требований к программному обеспечению (ПО), их систематизацию, выявление взаимосвязей, а также документирование. В англоязычной среде также говорят о дисциплине «инженерия требований» (англ. Requirements Engineering). В процессе сботребований важно принимать во внимание

возможные противоречия требований различных заинтересованных лиц, таких как заказчики,разработчики или пользователи.

Полнота и качество анализа требований играют ключевую роль в успехе всего проекта. Требования к ПО должны быть документируемые, выполнимые, тестируемые, с уровнем детализации, достаточным для проектирования системы. Требования могут быть функциональными и нефункциональными.

Анализ требований включает три типа деятельности:

- Сбор требований — общение с клиентами и пользователями, чтобы определить, каковы их требования; анализ предметной области.
- Анализ требований — определение, являются ли собранные требования неясными, неполными, неоднозначными или противоречащими; решение этих проблем; выявление взаимосвязи требований.
- Документирование требований — требования могут быть задокументированы в различных формах, таких как простое описание, сценарии использования, пользовательские истории, или спецификации процессов.

Анализ требований может быть длинным и трудным процессом, во время которого вовлечены много тонких психологических навыков. Новые системы изменяют окружающую среду и отношения между людьми, поэтому важно определить все заинтересованные лица, принять во внимание все их потребности и гарантировать, что они понимают значения новых систем.

Аналитики могут использовать несколько методов, чтобы выявить следующие требования от клиента: проведение интервью, использование фокус-групп или создание списков требований. Более современные методы включают создание прототипов и сценариев использования. Где необходимо, аналитик будет использовать комбинацию этих методов, чтобы выявить точные требования заинтересованных лиц так, чтобы была создана система, которая удовлетворяет деловые потребности.

Процесс анализа требований к информационной системе включает следующие фазы:

- Разработка требований
- Выявление требований
- Анализ требований
- Спецификация требований
- Проверка требований
- Управление требованиями

7.2 Выявление прецедентов

Диаграмма прецедентов (вариантов использования) является исходной концептуальной моделью системы в процессе ее проектирования и разработки. Разработка диаграммы прецедентов преследует цели:

- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Диаграммы прецедентов применяются для моделирования поведения системы с точки зрения внешнего наблюдателя. Поведение системы – это совокупность ее реакций в ответ на внешние события. Под реакцией понимается выполнение некоторого целостного

набора функций и формирование отклика, имеющего определенную ценность для некоторого субъекта. Под субъектом понимают кого-то или что-то (человека, устройство, программу и т.д.) так или иначе взаимодействующее с системой, а некоторый целостный набор функций, имеющих определенную ценность для субъекта, именуют прецедентом. Сущность концепции прецедентов подразумевает несколько важных пунктов.

1. Прецедент представляет собой завершенный фрагмент функциональных возможностей (включая основной поток логики управления, его любые вариации (подпотоки) и исключительные условия (альтернативные потоки)).

2. Фрагмент внешне наблюдаемых функций (отличных от внутренних функций).

3. Ортогональный фрагмент функциональных возможностей (прецеденты могут при выполнении совместно использовать объекты, но выполнение каждого прецедента независимо от других прецедентов)

4. Фрагмент функциональных возможностей, инициируемый субъектом. Будучи инициирован, прецедент может взаимодействовать с другими субъектами. При этом возможно, что субъект окажется только на принимающем конце прецедента, опосредованно инициированного другим субъектом.

5. Фрагмент функциональных возможностей, который предоставляет субъекту ощутимый полезный результат (и этот результат достигается в пределах одного прецедента).

Выявление прецедентов основано на анализе задач, выполняемых субъектами и целей субъектов применительно к системе.

Таким образом в любой системе существует некоторое множество субъектов. Каждому субъекту соответствует некоторый набор прецедентов с которыми данный субъект взаимодействует. Субъекты инициируют события приводящие к активизации того или иного прецедента. Результатом выполнения прецедента являются изменение состояния системы и/или отклик. Отклик направляется этому же или другому субъекту или может быть событием, активизирующим другой прецедент. Это означает, что в общем случае могут существовать прецеденты, которые не активизируются непосредственно ни одним субъектом. С другой стороны, субъекты, которым нельзя поставить в соответствие ни одного прецедента, смысла не имеют.

В общем случае можно выделить основные субъекты и второстепенные. Основными считаются субъекты непосредственно инициирующие хотя бы один прецедент. Второстепенными считаются субъекты либо инициирующие прецедент опосредованно (побуждающие другой субъект инициировать прецедент), либо являющиеся получателями отклика. Между субъектами возможны зависимости. Одни субъекты (независимые) активизируют некоторый прецедент исходя из своих внутренних потребностей, другие (зависимые) делают это только в случае «просьбы» со стороны другого субъекта или в качестве ответа на отклик.

По отношению к системе субъекты имеют двойственную природу. С одной стороны - это объекты внешней, по отношению к системе, среды. С другой стороны система должна хранить информацию о субъектах, чтобы «со знанием дела» взаимодействовать с ними, т.е. субъекты можно рассматривать как часть системы.

При инфологическом проектировании субъекты рассматриваются как супертип пользователя, а каждый отдельный субъект - как тип пользователя.

Между субъектами и прецедентами – основными компонентами диаграммы прецедентов – могут существовать различные отношения, которые описывают взаимодействие экземпляров одних субъектов и прецедентов с экземплярами других субъектов и прецедентов.

7.3 Диаграммы прецедентов, последовательностей, деятельности

Диаграммы прецедентов. Визуальное моделирование в UML можно представить как некоторый процесс поуровневого спуска от наиболее общей и абстрактной концептуальной модели исходной системы к логической, а затем и к физической модели соответствующей программной системы. Для достижения этих целей вначале строится модель в форме так называемой диаграммы вариантов использования (use case diagram), которая описывает функциональное назначение системы или, другими словами, то, что система будет делать в процессе своего функционирования.

Разработка диаграммы вариантов использования преследует цели:

- Определить общие границы и контекст моделируемой предметной области на начальных этапах проектирования системы.
- Сформулировать общие требования к функциональному поведению проектируемой системы.
- Разработать исходную концептуальную модель системы для ее последующей детализации в форме логических и физических моделей.
- Подготовить исходную документацию для взаимодействия разработчиков системы с ее заказчиками и пользователями.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером (actor) или действующим лицом называется любая сущность, взаимодействующая с системой извне. Это может быть человек, техническое устройство, программа или любая другая система, которая может служить источником воздействия на моделируемую систему так, как определит сам разработчик. В свою очередь, вариант использования (use case) служит для описания сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с актером. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие актеров с системой.

В самом языке UML пакет Варианты использования является подпакетом пакета Элементы поведения. Базовые элементы этого пакета - вариант использования и актер.

Вариант использования.

Стандартный элемент языка UML «вариант использования» применяется для спецификации общих особенностей поведения системы или любой другой сущности предметной области без рассмотрения внутренней структуры этой сущности. Каждый вариант использования определяет последовательность действий, которые должны быть выполнены проектируемой системой при взаимодействии ее с соответствующим актером. Диаграмма вариантов может дополняться пояснительным текстом, который раскрывает смысл или семантику составляющих ее компонентов. Такой пояснительный текст получил название примечания или сценария.

Отдельный вариант использования обозначается на диаграмме эллипсом, внутри которого содержится его краткое название или имя в форме глагола с пояснительными словами (рис. 7.1).

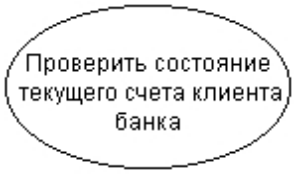


Рис. 7.1. Графическое обозначение варианта использования

Каждый вариант использования соответствует отдельному сервису. Сервис, который инициализируется по запросу пользователя, представляет собой законченную последовательность действий. Это означает, что после того как система закончит обработку запроса пользователя, она должна возвратиться в исходное состояние, в котором готова к выполнению следующих запросов.

Варианты использования описывают не только взаимодействия между пользователями и сущностью, но также реакции сущности на получение отдельных сообщений от пользователей и восприятие этих сообщений за пределами сущности. Варианты использования могут включать в себя описание особенностей способов реализации сервиса и различных исключительных ситуаций, таких как корректная обработка ошибок системы. Множество вариантов использования в целом должно определять все возможные стороны ожидаемого поведения системы и может рассматриваться как отдельный пакет.

Варианты использования неявно устанавливают требования, определяющие, как пользователи должны взаимодействовать с системой, чтобы иметь возможность корректно работать с предоставляемыми данной системой сервисами

Примерами вариантов использования могут являться следующие действия: проверка состояния текущего счета клиента, оформление заказа на покупку товара, получение дополнительной информации о кредитоспособности клиента, отображение графической формы на экране монитора и другие действия.

Актер представляет собой любую внешнюю по отношению к моделируемой системе сущность, которая взаимодействует с системой и использует ее функциональные возможности для достижения определенных целей или решения частных задач. Стандартным графическим обозначением актера на диаграммах является фигурка "человечка", под которой записывается конкретное имя актера (рис. 7.2).

Рис. 7.2. Графическое обозначение актера

В некоторых случаях актер может обозначаться в виде прямоугольника класса с ключевым словом "актер" и обычными составляющими элементами класса. Имена актеров должны записываться заглавными буквами и следовать рекомендациям использования имен для типов и классов модели. При этом символ отдельного актера связывает соответствующее описание актера с конкретным именем. Имена абстрактных актеров, как и других абстрактных элементов языка UML, рекомендуется обозначать курсивом.

Примечание

Имя актера должно быть достаточно информативным с точки зрения семантики. Вполне подходят для этой цели наименования должностей в компании (например, продавец, кассир, менеджер, президент).

Примерами актеров могут быть: клиент банка, банковский служащий, продавец магазина, менеджер отдела продаж, пассажир авиарейса, водитель автомобиля, администратор гостиницы, сотовый телефон и другие сущности, имеющие отношение к концептуальной модели соответствующей предметной области.

Актеры используются для моделирования внешних по отношению к проектируемой системе сущностей, которые взаимодействуют с системой и используют ее в качестве отдельных пользователей. В качестве актеров могут выступать другие системы, подсистемы проектируемой системы или отдельные классы. В каждый момент времени с системой взаимодействует вполне определенный пользователь, при этом он играет или выступает в одной из таких ролей. Наиболее наглядный пример актера - конкретный пользователь системы со своими собственными параметрами аутентификации.

Актеры взаимодействуют с системой посредством передачи и приема сообщений от вариантов использования. Сообщение представляет собой запрос актером сервиса от системы и получение этого сервиса. Это взаимодействие может быть выражено посредством ассоциаций между отдельными актерами и вариантами использования или

классами. Кроме этого, с актерами могут быть связаны интерфейсы, которые определяют, каким образом другие элементы модели взаимодействуют с этими актерами.

Два и более актера могут иметь общие свойства, т. е. взаимодействовать с одним и тем же множеством вариантов использования одинаковым образом. Такая общность свойств и поведения представляется в виде отношения обобщения с другим, возможно, абстрактным актером, который моделирует соответствующую общность ролей.

Диаграмма последовательности. В ходе проектирования ИС аналитик поэтапно спускается от общей концепции, через понимание ее логической структуры к наиболее детальным моделям, описывающим физическую реализацию.

С помощь диаграммы прецедентов (вариантов использования) выявляются основные пользователи системы и задачи, которые данная система должна решать. С помощью диаграммы деятельности мы описываем последовательность действий для каждого прецедента, необходимая для достижения поставленной цели.

Далее проектируется логическая структура системы с помощью диаграммы классов. На данном этапе выделяются классы, формирующие структуру БД Системы, а также классы реализующие некий набор операций, способствующий достижению целей в рамках выбранного прецедента. Для описания сложного поведения некоторых объектов (экземпляров класса) составляется диаграмма состояний.

Таким образом, аналитиками фиксируются такие поведенческие аспекты как алгоритм действий в рамках одного или нескольких прецедентов, необходимый для достижения определённого результата, а также изменение состояния объектов в ходе выполнения приведенных действий.

Зачастую на этапе спецификации требований необходимо показать не только алгоритм действий или изменение состояния объекта, но и обмен сообщениями между отдельными объектами Системы. Данную задачу решает диаграмма взаимодействия.

Диаграмма взаимодействия предназначена для моделирования отношений между объектами (ролями, классами, компонентами) Системы в рамках одного прецедента. Данный вид диаграмм отражает следующие аспекты проектируемой Системы:

- обмен сообщениями между объектами (в том числе в рамках обмена сообщениями со сторонними Системами)
- ограничения, накладываемые на взаимодействие объектов
- события, инициирующие взаимодействия объектов.

В отличие от диаграммы деятельности, которая показывает только последовательность (алгоритм) работы Системы, диаграммы взаимодействия акцентируют внимание разработчиков на сообщениях, инициирующих вызов определенных операций объекта (класса) или являющихся результатом выполнения операции.

Диаграмма последовательности является одной из разновидности диаграмм взаимодействия и предназначена для моделирования взаимодействия объектов Системы во времени, а также обмена сообщениями между ними.

Одним из основных принципов ООП является способ информационного обмена между элементами Системы, выражающийся в отправке и получении сообщений друг от друга. Таким образом, основные понятия диаграммы последовательности связаны с понятием Объект и Сообщение (рис.7.3).

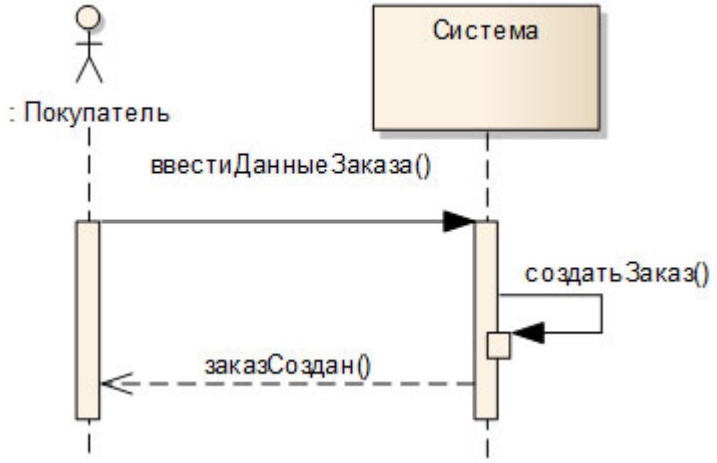


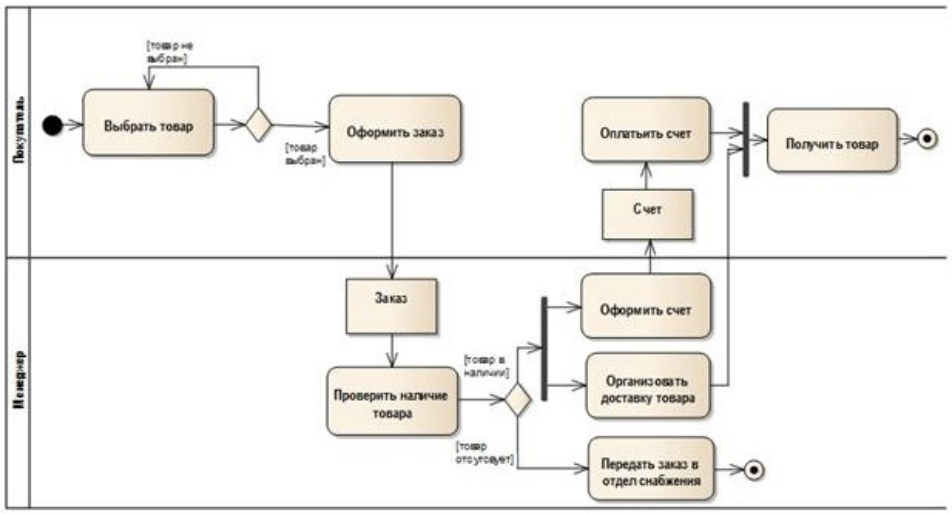
Рис. 7.3. диаграмма последовательности

На диаграмме последовательности объекты в основном представляю экземпляры класса или сущности, обладающие поведением. В качестве объектов могут выступать пользователи, инициирующие взаимодействие, классы, обладающие поведением в Системе или программные компоненты, а иногда и Системы в целом.

Объекты располагаются с лева на права таким образом, чтобы крайним с лева был тот объект, который инициирует взаимодействие. Неотъемлемой частью объекта на диаграмме последовательности является линия жизни объекта. Линия жизни показывает время, в течение которого объект существует в Системе. Периоды активности объекта в момент взаимодействия показываются с помощью фокуса управления. Временная шкала на диаграмме направлена сверху вниз.

Диаграмма деятельности. Создание Информационной Системы – сложный процесс, который можно представить как поэтапный спуск от общей концепции будущей ИС, через понимание ее логической структуры к наиболее детальным моделям, описывающим физическую реализацию. Диаграмма деятельности принадлежит к логической модели.

В качестве графического представления для выделения основных функций Системы мы применяем диаграмму вариантов использования (use case). Диаграмма вариантов использования дает нам представление ЧТО должна делать Система. На вопрос КАК мы можем ответить, используя диаграмму активности.



То есть если варианты использования ставят перед Системой цель, то диаграмма деятельности показывает последовательность действий, необходимых для ее достижения. Действия (action) это элементарные шаги, которые не предполагают дальнейшую декомпозицию.

Деятельность может содержать входящие и/или исходящие дуги деятельности, показывающие потоки управления и потоки данных. Если поток соединяет две деятельности, он является потоком управления. Если поток заканчивается объектом, он является потоком данных.

Деятельность выполняется, только тогда, когда готовы все его «входы», после выполнения, деятельность передает управление и(или) данные на свои «выходы». Саму диаграмму деятельности принято располагать таким образом, чтобы действия следовали слева направо или сверху вниз.

Чтобы указать, где именно находится процесс, используется абстрактная точка «маркер» (или «токен»). Визуально на диаграмме маркер не показывается, данное понятие вводится только для удобства описания динамического процесса.

Переход маркера осуществляется между узлами. Маркер может не содержать никакой дополнительной информации (пустой маркер) и тогда он называется маркером управления (control flow token) или же может содержать ссылку на объект или структуру данных, и тогда маркер называется маркером данных (data flow token).

8. Создание профиля UML

8.1 Анализ прецедентов

Прецеденты — описание "на высоком уровне" функций, предоставляемых системой, т.е. они иллюстрируют, как можно использовать систему.

Продолжим пример с системой бронирования авиабилетов. Кроме действующих лиц необходимо определить прецеденты. Безразлично, что будет специфицировано первым — более того, можно одновременно выявлять и действующих лиц, и прецеденты. Для описания прецедентов необходимо ответить на вопрос: "В чем особенности работы системы, имеющие важное значение для внешнего мира?" Из приведенного выше краткого описания системы ясно, что она обязана обеспечивать продажу билетов, изменение бронирования и отмену заказа на билет. Эти условия становятся хорошими кандидатами для прецедентов, поскольку каждое такое действие имеет значение для конечных пользователей системы. В анализ не включаются некоторые прецеденты из унаследованной системы, например "Получить информацию о рейсе". Это выходящая из зоны анализа логика, которая не заботит конечного пользователя. Поэтому нельзя рассматривать такие информационные элементы, как прецеденты. С другой стороны, действия "Продажа билета", "Изменение заказа" или "Отмена заказа" имеют значение для конечного пользователя с точки зрения описания системы "на высоком уровне", поэтому такие действия должны быть отражены в модели через прецеденты. В UML прецедент изображается в виде овала (см. рис.1).

Преимуществом описания системы через прецеденты является способность отделения реализации системы от побудительных причин для ее разработки. Можно сконцентрироваться на действительно важных аспектах — удовлетворении требований и ожиданий клиента, которые превалируют над деталями реализации. Анализ прецедентов позволит клиенту увидеть то, что он получит от системы, и согласовать границы действия системы в самом начале разработки проекта.

Применение прецедентов немного отличается от традиционных способов разработки. Разделение проекта по прецедентам ориентировано на процессы в системе, но не на их реализацию. В этом проявляется отличие данного метода от часто используемого декомпозиционного подхода. Хотя функциональная декомпозиция также предназначена

для последовательного деления проблемы на части, с которыми будет работать система, прецеденты акцентируют то, что ожидает от системы клиент.

В начале работы над проектом возникает вопрос: "Как выявить прецеденты?" Прецеденты не зависят от реализации и предоставляют высокоуровневое описание системы. Обсудим каждую часть этого определения.

Во-первых, прецеденты не зависят от реализации. Прецеденты заостряют внимание на том, что должна делать система, а не на том, как она будет это делать.

Во-вторых, прецеденты дают высокоуровневую картину системы. Набор прецедентов должен ясно и просто показать пользователю всю систему. Прецедентов не должно быть много, чтобы пользователю не приходилось знакомиться с большим объемом документации только для того, чтобы узнать, что делает эта система. В то же время набор прецедентов обязан предоставить полное описание системы. В типичных системах выявляется от 20 до 70 прецедентов (если же их будет 3000, то теряется преимущество простоты описания). Допустимо применять разные типы отношений, чтобы при необходимости подразделять сложные прецеденты. Можно и группировать прецеденты в пакеты, чтобы упорядочить структуру описания.

Наконец, прецеденты должны акцентировать то, что получит пользователь от системы. Каждый вариант использования должен представлять собой законченную транзакцию между пользователем и системой, причем результат транзакции должен иметь важное значение для пользователя. Прецеденты именуются по терминологии пользователей, а не техническими названиями, которые не всегда понятны работающим с системой людям. Не должно быть прецедента "Взаимодействие с кредитной системой банка для проверки номера кредитной карточки". Клиент пытается купить билет, поэтому разумно ввести прецедент "Покупка билета" или "Купить билет". Обычно название прецедента приводится в формате "<отглагольное существительное><дополнение> " или "<глагол><существительное>" и определяет то, что в результате получает клиент. Пользователю безразлично количество систем, с которыми взаимодействует его система, какие специфические операции выполняются внутри системы и сколько строк кода будет написано для проверки кредитной карточки Visa. Пользователю важен только покупаемый им билет, поэтому акцентируются результаты действия системы, но не операции, приводящие к такому результату.

Получив окончательный список прецедентов, следует задуматься о его полноте и попытаться ответить на вопрос: существует ли для любого функционального требования хотя бы один прецедент? Если для требования нет прецедента, то такое требование не будет реализовано в системе.

8.2 Типы объектов модели

С помощью моделей данных объекты реального мира можно представить как абстрактные информационные объекты. Информационный объект состоит из совокупности значений, каждое из которых описывает то или иное свойство моделируемого объекта.

Как правило, для упрощения работы с моделями данных каждому значению, описывающему свойство моделируемого объекта, присваивается имя – атрибут. Таким образом, с тем или иным свойством моделируемого объекта в модели связаны атрибут и его значение. В конструкторской базе данных хранятся значения, атрибуты, информационные объекты и модели.

Модель объекта как физическая, так и логическая может служить не только для построения его изображений, но и для других целей, например, описания структуры и динамики поведения, как самого объекта, так и различных систем, которые включают в себя объекты такого рода.

Модели могут описывать экономические процессы, и позволять находить численные значения их характеристик.

Для того чтобы модель некоторой системы позволяла строить изображения, в нее должны входить следующие данные:

- характеристики составных частей модели – объектов;
- базовые элементы и описания и форм;
- отношения, в которых находятся базовые элементы;
- описания внешних связей различных объектов;
- параметры описываемых объектов и допустимые области их изменения;
- алгоритмы для анализа значений параметров объектов;
- алгоритм оценки результатов моделирования.

Далее под термином «модель» будем понимать информационно насыщенное описание объектов и процессов, определяющее как структуру, так и поведение моделируемой системы. Если модель используется в конструкторском проектировании, в нее должны входить геометрические данные, обрабатывая которые, прикладная программа может построить графическое изображение. В виде графических изображений могут быть представлены: электрические схемы, топологические карты микросхем, чертежи механических деталей, строительные чертежи, географические карты и т.д.

Около 80 % решаемых при конструкторском проектировании задач связаны со сбором и обработкой данных. Остальные 20 % задач – вычислительного характера – их автоматизация требует моделирования.

Только стандартизация представления элементов моделей позволит обеспечить обмен моделями между различными системами, объединенными сетью связи. Основная цель такой метасистемы – объединить усилия специалистов различных специальностей, работающих над проектом, предоставляя им средства связи, как друг с другом, так и с базами данных конструкторских, производственных и управленческих подразделений.

В качестве исходных данных система визуализации использует составляющие модель информационные объекты, представленные структурами данных. В результате последовательных преобразований этих данных строится изображение, соответствующее модели. Построенное изображение может быть воспроизведено на графическом дисплее или фиксирующем графическом устройстве – графопостроителе, принтере.

Модели широко используются в научных исследованиях (с целью приобретения новых знаний об окружающем мире), в технике и практической деятельности людей.

Никакая модель не может с абсолютной точностью воспроизвести все свойства и поведение своего прототипа, и поэтому получаемые на основе модели числовые или иные результаты соответствуют реальности лишь приближенно, с определенной степенью точности. Иногда точность модели можно выразить в каких-то единицах (например, в процентах), иногда приходится ограничиваться «качественными» оценками или просто здравым смыслом.

Например, математические модели физических процессов, основанные на законах Ньютона, применимы лишь в определенном диапазоне плотностей, скоростей, температур. В земных условиях эти модели вполне удовлетворяют нас, однако многие процессы во Вселенной (для которых характерны чудовищные плотности, скорости, температуры) нельзя ни понять, ни описать на основе законов Ньютона. В этих условиях необходимо использовать другие, более точные модели физических процессов, например, специальную и общую теорию относительности Эйнштейна (хотя существуют и другие).

Создавая модель, человек, прежде всего, старается отобрать наиболее важные, существенные для объекта моделирования черты и свойства, пренебрегая при этом теми характеристиками объекта, которые не оказывают заметного влияния на поведение объекта в рамках поставленной задачи.

В зависимости от поставленной задачи, способа создания модели и предметной области различают множество типов моделей.

Существуют общепринятые и широко используемые типы:

- математическая (в первую очередь);
- физическая;
- информационная;
- численная;

- специальных типов:

- эвристическая;
- логическая;
- концептуальная;
- сетевая;
- реляционная и т.д.

В технике и быту термином «модель» обозначают некий эталон, образец, например: модель автомобиля или утюга, фотомодель, модель художника и т.д.

8.3 Диаграммы взаимодействий, классов, состояний, развертывания

Целью диаграмм взаимодействия является визуализация интерактивного поведения системы. Визуализация взаимодействия – сложная задача. Следовательно, решение состоит в том, чтобы использовать различные типы моделей, чтобы охватить различные аспекты взаимодействия.

Диаграммы последовательности и сотрудничества используются для отображения динамической природы, но под другим углом.

Целью диаграммы взаимодействия является –

- Чтобы зафиксировать динамическое поведение системы.
- Для описания потока сообщений в системе.
- Для описания структурной организации объектов.
- Для описания взаимодействия между объектами.

Чтобы зафиксировать динамическое поведение системы.

Для описания потока сообщений в системе.

Для описания структурной организации объектов.

Для описания взаимодействия между объектами.

Диаграмма классов.

Центральное место в ООАП занимает разработка логической модели системы в виде диаграммы классов. Нотация классов в языке UML проста и интуитивно понятна всем, кто когда-либо имел опыт работы с CASE-инструментариями. Схожая нотация применяется и для объектов - экземпляров класса, с тем различием, что к имени класса добавляется имя объекта и вся надпись подчеркивается.

Нотация UML предоставляет широкие возможности для отображения дополнительной информации (абстрактные операции и классы, стереотипы, общие и частные методы, детализированные интерфейсы, параметризованные классы). При этом возможно использование графических изображений для ассоциаций и их специфических свойств, таких как отношение агрегации, когда составными частями класса могут выступать другие классы.

Диаграмма классов (class diagram) служит для представления статической структуры модели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру и типы отношений. На данной диаграмме не указывается информация о

временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов представляет собой некоторый граф, вершинами которого являются элементы типа "классификатор", которые связаны различными типами структурных отношений. Следует заметить, что диаграмма классов может также содержать интерфейсы, пакеты, отношения и даже отдельные экземпляры, такие как объекты и связи. Когда говорят о данной диаграмме, имеют в виду статическую структурную модель проектируемой системы. Поэтому диаграмму классов принято считать графическим представленном таких структурных взаимосвязей логической модели системы, которые не зависят или инвариантны от времени.

Диаграмма классов состоит из множества элементов, которые в совокупности отражают декларативные знания о предметной области. Эти знания интерпретируются в базовых понятиях языка UML, таких как классы, интерфейсы и отношения между ними и их составляющими компонентами. При этом отдельные компоненты этой диаграммы могут образовывать пакеты для представления более общей модели системы. Если диаграмма классов является частью некоторого пакета, то ее компоненты должны соответствовать элементам этого пакета, включая возможные ссылки на элементы из других пакетов.

В общем случае пакет статической структурной модели может быть представлен в виде одной или нескольких диаграмм классов. Декомпозиция некоторого представления на отдельные диаграммы выполняется с целью удобства и графической визуализации структурных взаимосвязей предметной области. При этом компоненты диаграммы соответствуют элементам статической семантической модели. Модель системы, в свою очередь, должна быть согласована с внутренней структурой классов, которая описывается на языке UML.

Класс

Класс (class) в языке UML служит для обозначения множества объектов, которые обладают одинаковой структурой, поведением и отношениями с объектами из других классов. Графически класс изображается в виде прямоугольника, который дополнительно может быть разделен горизонтальными линиями на разделы или секции (рис. 8.1). В этих разделах могут указываться имя класса, атрибуты (переменные) и операции (методы).

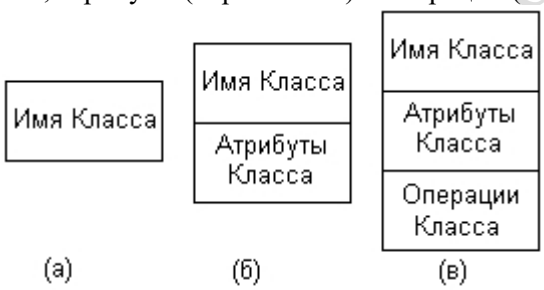


Рис. 8.1. Графическое изображение класса на диаграмме классов

Обязательным элементов обозначения класса является его имя. На начальных этапах разработки диаграммы отдельные классы могут обозначаться простым прямоугольником с указанием только имени соответствующего класса (рис. 5.1, а). По мере проработки отдельных компонентов диаграммы описания классов дополняются атрибутами (рис. 5.1, б) и операциями (рис. 5.1, в).

Предполагается, что окончательный вариант диаграммы содержит наиболее полное описание классов, которые состоят из трех разделов или секций. Иногда в обозначениях классов используется дополнительный четвертый раздел, в котором приводится семантическая информация справочного характера или явно указываются исключительные ситуации.

Даже если секция атрибутов и операций является пустой, в обозначении класса она выделяется горизонтальной линией, чтобы сразу отличить класс от других элементов языка UML. Примеры графического изображения классов на диаграмме классов приведены на рис. 8.2. В первом случае для класса "Прямоугольник" (рис. 8.2, а) указаны только его атрибуты - точки на координатной плоскости, которые определяют его расположение. Для класса "Окно" (рис. 8.2, б) указаны только его операции, секция атрибутов оставлена пустой. Для класса "Счет" (рис. 8.2, в) дополнительно изображена четвертая секция, в которой указано исключение - отказ от обработки просроченной кредитной карточки.

Рис.8.2. Примеры графического изображения классов на диаграмме

Имя класса.

Имя класса должно быть уникальным в пределах пакета, который описывается некоторой совокупностью диаграмм классов (возможно, одной диаграммой). Оно указывается в первой верхней секции прямоугольника. В дополнение к общему правилу наименования элементов языка UML, имя класса записывается по центру секции имени полужирным шрифтом и должно начинаться с заглавной буквы. Рекомендуется в качестве имен классов использовать существительные, записанные по практическим соображениям без пробелов. Необходимо помнить, что именно имена классов образуют словарь предметной области при ООАП.

В первой секции обозначения класса могут находиться ссылки на стандартные шаблоны или абстрактные классы, от которых образован данный класс и, соответственно, от которых он наследует свойства и методы. В этой секции может приводиться информация о разработчике данного класса и статус состояния разработки, а также могут записываться и другие общие свойства этого класса, имеющие отношение к другим классам диаграммы или стандартным элементам языка UML.

Примерами имен классов могут быть такие существительные, как "Сотрудник", "Компания", "Руководитель", "Клиент", "Продавец", "Менеджер", "Офис" и многие другие, имеющие непосредственное отношение к моделируемой предметной области и функциональному назначению проектируемой системы.

Класс может не иметь экземпляров или объектов. В этом случае он называется абстрактным классом, а для обозначения его имени используется наклонный шрифт (курсив). В языке UML принято общее соглашение о том, что любой текст, относящийся к абстрактному элементу, записывается курсивом. Данное обстоятельство является семантическим аспектом описания соответствующих элементов языка UML.

Примечание

В некоторых случаях необходимо явно указать, к какому пакету относится тот или иной класс. Для этой цели используется специальный символ разделитель - двойное двоеточие "::". Синтаксис строки имени класса в этом случае будет следующий <Имя_пакета>::<Имя_класса>. Другими словами, перед именем класса должно быть явно указано имя пакета, к которому его следует отнести. Например, если определен пакет с именем "Банк", то класс "Счет" в этом банке может быть записан в виде: "Банк::Счет".

9. Моделирование компонентов платформы JEE в UML

9.1 Назначение платформы JEE.

В современном быстро развивающемся мире электронного бизнеса, в который входят сложные распределенные приложения, важно, чтобы приложения предприятия выпускались на рынок как можно быстрее. Это означает, что команда разработчиков проекта не может позволить себе тратить время на разработку служб системного уровня, таких как службы удаленных соединений, присвоения имен, постоянных данных, защиты или управления транзакциями. Команда разработчиков проекта должна создать и внедрить переносимые многоразовые компоненты; заново изобретать испытанные, хорошо себя зарекомендовавшие архитектуры нерационально.

Платформа Java™ 2 Platform, Enterprise Edition (J2EE™) специально предназначена для указанной цели. Она предоставляет хорошо документированную стандартизованную среду для разработки и запуска распределенных, многоуровневых, основанных на компонентах приложений на Java. Эта среда автоматически выполняет большую часть низкоуровневой работы при создании приложения, например, настройку служб удаленных соединений, присвоения имен, постоянных данных, защиты и управления транзакциями, позволяя разработчикам сосредоточиться на бизнес-логике приложения.

Состав платформы J2EE:

- Набор стандартов для компонентов J2EE и для платформы J2EE, на которой работают эти компоненты.
- Эскиз проекта разработки приложения, подробно описывающий платформу J2EE и содержащий ценную информацию о том, как следует разрабатывать приложения J2EE.
- Справочная реализация платформы J2EE, предоставленная фирмой Sun Microsystems Inc. в качестве стандарта, с которым можно сравнивать создаваемые коммерческие продукты J2EE. Справочная реализация содержит полностью готовые к использованию примеры приложений.
- Набор тестов на совместимость, предназначенный для проверки и оценивания коммерческих реализаций J2EE относительно стандартов J2EE.

Платформа J2EE аналогична службам операционной системы компьютера. Используя инструменты программирования, операционная система предоставляет стандартные службы, на основе которых вы можете разрабатывать и выполнять приложения, не заботясь о настройке низкоуровневого управления дисками, памятью, видеовыводом, сетью и т.п. Вы имеете дело только с самим приложением, но не с базовой системой. Платформа J2EE предоставляет своеобразную *операционную систему* для приложений предприятия.

Применение платформы J2EE позволяет упростить процесс разработки, так что команда разработчиков проекта сможет сконцентрироваться на фактической бизнес-логике приложения, не отвлекаясь на вопросы системного уровня. Вероятность достичь успеха в том, чтобы своевременно создать свободную от ошибок систему, отвечающую требованиям пользователей, будет гораздо выше, если команда разработчиков может сконцентрироваться лишь на том, что должно делать приложение, и не заботиться о том, как предоставить приложению все необходимые ему базовые службы.

9.2 Сервлеты и компоненты JSP.

Java Server Pages (JSP) - это одна из технологий J2EE, которая представляет собой расширение технологии сервлетов для упрощения работы с Web-содержимым. Страницы JSP позволяет легко разделить Web-содержимое на статическую и динамическую часть, допускающую многократное использование ранее определенных компонентов.

Разработчики Java Server Pages могут использовать компоненты JavaBeans и создавать собственные библиотеки нестандартных тегов, которые инкапсулируют сложные динамические функциональные средства.

Спецификация Java Server Pages наследует и расширяет спецификацию сервлетов. Как и сервлеты, компоненты JSP относятся к компонентам Web и располагаются в Web-контейнере. Страницы JSP не зависят от конкретной реализации Web-контейнера, что обеспечивает возможность их повторного использования.

В дополнение к классам и интерфейсам для программирования сервлетов (пакеты javax.servlet и javax.servlet.http),в пакетах javax.servlet.jsp и javax.servlet.jsp.target содержатся классы и интерфейсы, относящиеся к программированию Java Server Pages.

Базовая архитектура Java Server Pages в самом общем виде представлена на рисунке (рис. 9.1). Платформа J2EE обеспечивает базу, на которой функционирует все приложение в целом и страницы JSP, в частности, в то время, как сеть Интернет предоставляет механизм транспортировки данных.

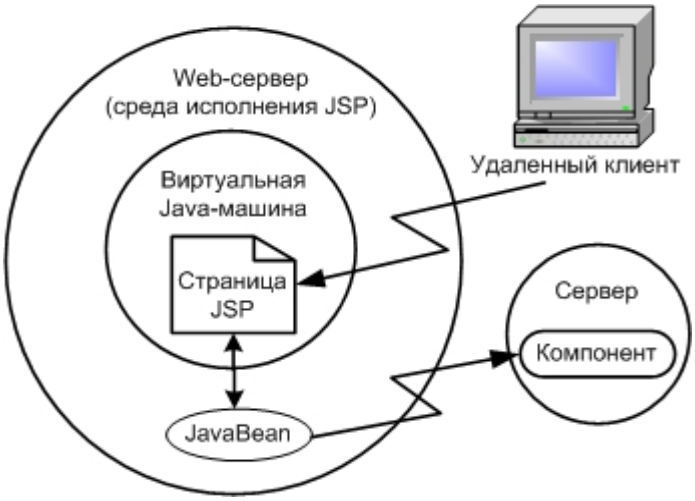


Рис.9.1. Базовая архитектура Java Server Pages

Страница JSP располагается на Web-сервере в среде виртуальной Java-машины. Доступ к страниц JSP, как и в случае сервлета, осуществляется через Web с использованием протокола HTTP.

Страница JSP функционирует под управлением JSP Engine (среды исполнения JSP). Страница JSP может взаимодействовать с программным окружением с помощью компонентов JavaBeans, получая и устанавливая его параметры, используя теги: <jsp:useBean>, <jsp:getProperty>, <jsp:setProperty>.

Компонент JavaBean сам может участвовать в других процессах, предоставляя результаты в виде своих параметров, доступных страницам JSP, участвующим в сеансе, а через них - всем пользователям, запрашивающим эти страницы JSP.

Использование межплатформенных компонентов JavaBeans и библиотек тегов значительно расширяет возможности JSP. Программный Java-код в странице JSP, в идеале, должен использоваться только для управления представлением информации.

9.3 Шаблон MVC.

Одно из понятий, с которым приходится часто встречаться в программировании - это понятие шаблона программирования или проектирования MVC.

MVC - это всего лишь один способов, как вы можете организовать свой код. Он описывает некоторые правила, как это нужно делать.

Нужно понимать, что это не единственный способ организации кода. Существуют и другие способы.

Задача MVC - сделать так, чтобы наш код проще воспринимался, проще читался и проще и быстрее обрабатывался компьютером.

Основная идея MVC - разделить внешний вид приложения от логики. Т.к. мы говорим о веб-приложениях, то под внешним видом у нас понимается HTML-разметка

документа и его CSS оформление. А под логикой мы понимаем различные скрипты, функции, классы и.т.д.

Шаблон MVC - это аббревиатура, которая состоит из первых букв тех составляющих, которые в нее входят: Модель и Контроллер (скрипты и логика) и Вид (HTML и CSS (шаблонизатор)),.

В логике не должно быть внешнего вида, а во внешнем виде не должно быть логики. Здесь важно понимать такой момент: не то, чтобы вы не можете нарушить это правило, вы это сделать можете и ваша программа будет работать. Идея в то, что разместив код таким образом, вы отклоняетесь от модели MVC и теряете все преимущества, которые эта модель предоставляет.

Модель MVC - это всего лишь набор рекомендаций, которым нужно придерживаться при написании кода. Эти рекомендации даны вам для вашего же блага, чтобы в дальнейшем вы потом легче читали код, легче его анализировали и.т.д.

Контроллер

В модели MVC самым главным элементом, с которого все начинается и на котором все, как правило, заканчивается - это контроллер.

Задача контроллера - принять запрос пользователя и решить, что делать с этим запросом далее, если требуется, то перенаправить запрос в модель, который обработает информацию и возвращает ее контроллеру.

После того, как информация обработана, контроллер решает, что с ней делать дальше, если пользователю достаточно предоставить просто набор каких-то данных, не в виде html-странице, а например, в формате json, контроллер этот набор данных ему выдает.

Если необходимо сформировать html-страницу, контроллер передает эти данные в вид и внутри вида шаблонизатор формирует каркас страницы, выдает ее назад контроллеру и контроллер уже выдает этот каркас пользователю в виде html-страницы.

Из 3 частей MVC модели контроллер является обязательной частью. Остальные части являются опциональными. Если пользователю достаточно только отдать какой-то набор данных, то можно обойтись без вида. Если не нужно обрабатывать данные, то можно обойтись без модели.

Контроллер - это некоторая функция. Может содержать ссылки на какие-то другие функции и возвращает какой-то результат.

Контроллер можно сравнить с начальником отдела на предприятии. Начальнику отдела приходит задание от директора предоставить какой-то отчет и далее контроллер, под которым мы имеем в виду начальника отдела, принимает решение. Или он сам сделает этот отчет и отдаст его директору или он может передать ее выполнение своим подчиненным. В качестве подчиненного можно представить "модель".

Модель

Модель - это "подчиненный", который делает какую-то рутинную работу, выполняет обработку данных. Он произвел эту обработку данных и выдал результат начальнику (контроллеру). Контроллер принял этот отчет.

После того, как данные обработаны, начальник (контроллер) относит этот отчет директору (т.е. человеку, который запросил эти данные).

Примерно на таком же принципе работает и модель MVC.

Вид

Вид - можно представить тоже в виде каких-то работников отдела, которые набирают уже подготовленные данные от модели на компьютере, например в текстовом редакторе Microsoft Word и распечатывают его. Вот, по сути, этом и заключается задача "вида".

Задача модели - это такие работники, которые производят расчеты, расчетчики какие-то и.т.д. Примерно такую аналогию можно произвести.

Модель - это работник или исполнитель. Содержит основные функции и классы (логику).

Не всегда удается и целесообразно придерживаться структуры Controller - Model. В принципе, можно какую-то программную логику и более целесообразно, если этой логики мало, написать эту логику внутри контроллера. Нет необходимости создавать дополнительные функции и отдельные классы, чтобы сделать всего несколько действий.

3 ВОПРОСЫ ДЛЯ САМОКОНТРОЛЯ

Понятие жизненного цикла

1. Определение системы обработки данных
2. Сущность жизненного цикла

Процессы жизненного цикла

3. Процессы жизненного цикла
4. Вспомогательные процессы жизненного цикла
5. Организационные процессы жизненного цикла

Методы разработки программного обеспечения

1. Интуитивный подход
2. Каскадный процесс
3. Рациональный унифицированный процесс
4. Экстремальное программирование и разработка по функциям

Структурный подход к проектированию программного обеспечения

1. Сущность структурного подхода.
2. Метод функционального моделирования
3. Метод функционального моделирования

Архитектура программного обеспечения

4. Определение программной архитектуры
5. Ключевые понятия архитектуры программного обеспечения предприятия

Объектно-ориентированный подход к проектированию программного обеспечения

1. Сущность объектно-ориентированного подхода.
2. Обзор языка моделирования UML
3. Состав объектной модели

Анализ требований клиента

1. Идентификация исполнителей
2. Выявление прецедентов
3. Диаграммы прецедентов
4. Диаграммы последовательностей
5. Диаграммы деятельности

Анализ требований клиента

1. Идентификация исполнителей
2. Выявление прецедентов

3. Диаграммы прецедентов, последовательностей, деятельности

Создание профиля UML

1. Анализ прецедентов
2. Типы объектов модели
3. Диаграммы взаимодействий
4. Диаграммы классов
5. Диаграммы состояний
6. Диаграммы развертывания

Моделирование компонентов платформы JEE в UML

1. Назначение платформы JEE
2. Составные части платформы JEE
3. Сервлеты и компоненты JSP
4. Шаблон MVC

Отношения между классами

1. Отношение зависимости
2. Отношение ассоциации
3. Отношение агрегации
4. Отношение обобщения

4 ЗАДАНИЯ К ЛАБОРАТОРНЫМ РАБОТАМ

Лабораторная работа №1 Создание функциональных моделей промышленного производства по методологии IDEF0

Лабораторная работа №2 Декомпозиция производственных процессов по методологии IDEF0

Лабораторная работа №3 Диаграмма дерева узлов и диаграммы FEO

Лабораторная работа №4 Стоимостный анализ

Лабораторная работа №5 Создание UML схемы паттерна Абстрактная фабрика, которая будет описывать кампании по производству телефонов и мониторов. В UML схеме описать название методов.

Лабораторная работа №6 Создание UML схемы паттерна **Builder**, которая будет описывать класс Person. В UML схеме описать название методов.

Лабораторная работа №7 Реализация паттерна проектирования **Singleton.** Класс должен содержать одно поле и метод который будет возвращать или создавать экземпляр класса.

Примеры реализации лабораторных работ:
Создание UML схемы паттерна Абстрактная фабрика
Для реализации паттерна была выбрана программа IntelliJ IDEA. Данная программа предназначена для интеллектуальной помощи в написании кода, надежных рефакторингов, быстрой навигации по коду, широкого набора встроенных инструментов разработчика, поддержки веб- и корпоративной разработки и многих других полезных возможностей. Обладает удобным интерфейсом, универсальностью и поддержкой большого количества возможностей.

Описание классов и методов в IntelliJ IDEA:

```java
package by.gsu.design.lesson3.factory;

import ...

5 usages  2 implementations
public interface AbstractFactory {
    1 usage  2 implementations
    AbstractPhone createPhone();
    1 usage  2 implementations
    AbsrtactMonitor createMonitor();
}
```

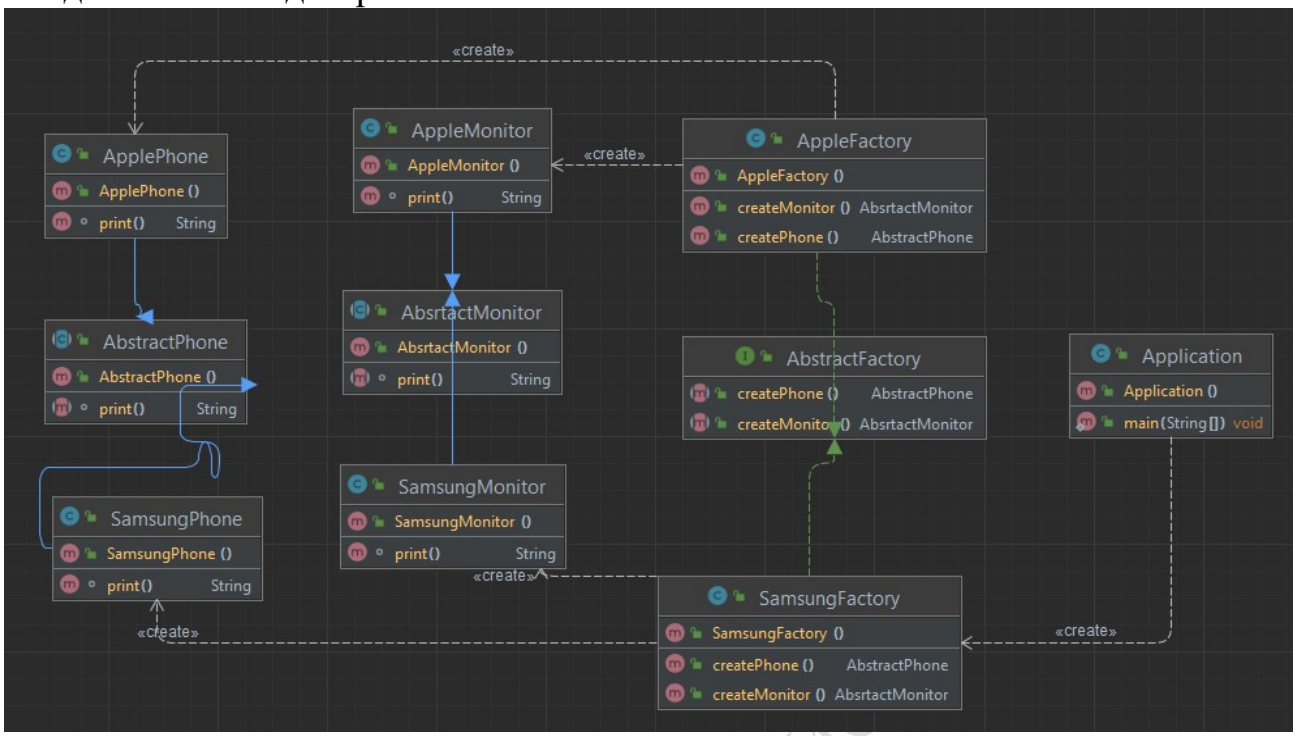
```java
package by.gsu.design.lesson3.factory;

import ...

public class AppleFactory implements AbstractFactory {

    1 usage
    @Override
    public AbstractPhone createPhone() { return new ApplePhone(); }

    1 usage
    @Override
    public AbsrtactMonitor createMonitor() { return new AppleMonitor(); }
}
```

```java
package by.gsu.design.lesson3;

public abstract class AbsrtactMonitor {
    1 usage  2 implementations
    abstract String print();
}
```

```java
package by.gsu.design.lesson3;

public class SamsungMonitor extends AbsrtactMonitor{

    1 usage
    @Override
    String print() { return "I'm SamsungMonitor"; }
}
```

```java
package by.gsu.design.lesson3;

import ...

public class Application {

    public static void main(String[] args) {
        AbstractFactory phoneFactory = new SamsungFactory();
        AbstractPhone phone = phoneFactory.createPhone();
        System.out.println(phone.print());


        AbstractFactory abstractFactory = new SamsungFactory();
        System.out.println(abstractFactory.createMonitor().print());

    }

}
```

Создание UML – диаграммы в IntelliJ IDEA:



5 ТЕСТОВЫЕ ЗАДАНИЯ (примеры)

1 Баллов: 1

Какие две фундаментальные концепции включает моделирование прецендентов:

Выберите один ответ.

- a. управление и прецедент
- b. исполнитель и процесс
- c. управление и процесс
- d. исполнитель и прецедент

2 Баллов: 1

Какой процесс обеспечивает соответствующие гарантии того, что ПО и процессы его ЖЦ соответствуют заданным требованиям и утвержденным планам:

Выберите один ответ.

- a. Процесс обеспечения качества
- b. Процесс разработки
- c. Процесс документирования
- d. Процесс управления

3 Баллов: 1

Что изображает диаграмма классов:

Выберите один ответ.

- a. статические отношения между различными структурными элементами
- b. взаимодействия между исполнителем и системой во времени
- c. физические взаимосвязи между программными и аппаратными компонентами системы
- d. структурные отношения между исполнителями и прецедентами

4 Баллов: 1

Какой из следующих процессов относится к организационным процессам жизненного цикла ПО:

Выберите один ответ.

- a. Процесс управления
- b. Процесс документирования
- c. Процесс приобретения
- d. Процесс совместной оценки

5 Баллов: 1

В чём суть компоненов JSP:

Выберите один ответ.

- a. возможность внедрения кода Java в структурированный документ HTML или XML, в результате чего код представления можно легко обрабатывать как обычный код HTML
- b. инкапсуляция постоянных данных в хранилище данных, которое обычно представляет собой полную или частичную запись базы данных
- c. облегчение операций взаимодействия с базами данных для разработчика
- d. сведение к минимуму связей между объектами системы путем их классификации по определенным наборам обязанностей

6 Баллов: 1

Какой подход называют двухуровневым:

Выберите один ответ.

- a. Подход, где логика представления отделяется от логики приложения, а логика приложения — от данных, на которых базируется программа
- b. Подход, где интерфейс программы привязан к логике приложения, а логика приложения зависит от конкретных структур данных
- c. Подход, где логика представления зависит от логики приложения, а логика приложения отделяется от данных, на которых базируется программа
- d. Подход, где аспекты представления и части логики приложения выделяются в отдельный уровень

7

Баллов: 1

Все процессы ЖЦ ПО разделены на 3 группы. Сколько процессов относится к группе «организационные»:

Выберите один ответ.

- a. 6
- b. 8
- c. 4
- d. 2

8

Баллов: 1

Предмет или явление, имеющие четко определяемое поведение - это:

Выберите один ответ.

- a. Функция
- b. Объект
- c. Класс
- d. Метод

9

Баллов: 1

Что изображает диаграмма последовательностей:

Выберите один ответ.

- a. взаимодействие между исполнителем и системой во времени
- b. физические взаимосвязи между программными и аппаратными компонентами системы
- c. поток передачи управления в прецеденте от одного действия к другому
- d. структурные отношения между исполнителями и прецедентами

10

Баллов: 1

Ранжированная или упорядоченная система абстракций, расположение их по уровням - это:

Выберите один ответ.

- a. Иерархия
- b. Наследование
- c. Модульность
- d. Абстрагирование

11

Баллов: 1

Множество объектов, связанных общностью структуры и поведения - это:

Выберите один ответ.

- a. Функция
- b. Класс
- c. Метод
- d. Переменная

12

Баллов: 1

Построение новых классов на основе существующих с возможностью добавления или переопределения данных и методов называется:

Выберите один ответ.

- a. Полиморфизм
- b. Иерархия
- c. Абстрагирование
- d. Наследование

13 Баллов: 1

Какой из следующих процессов не относится к основным процессам жизненного цикла ПО:

Выберите один ответ.

- a. Процесс эксплуатации
- b. Процесс разработки
- c. Процесс приобретения
- d. Процесс документирования

14 Баллов: 1

Программный объект, который выполняется на сервере и отвечает за управление определенными типами компонентов - это:

Выберите один ответ.

- a. компонент
- b. шаблон
- c. контейнер
- d. сервлет

15 Баллов: 1

Какой из следующих процессов не относится к вспомогательным процессам жизненного цикла ПО:

Выберите один ответ.

- a. Процесс разработки
- b. Процесс разрешения проблем
- c. Процесс совместной оценки
- d. Процесс обеспечения качества

18 Баллов: 1

Наиболее важным базовым принципом структурного подхода является:

Выберите один ответ.

- a. принцип структурирования данных
- b. принцип «разделяй и властвуй»
- c. принцип абстрагирования
- d. принцип непротиворечивости

19 Баллов: 1

Веб-компоненты, которые могут генерировать динамическое содержимое для веб-страниц - это:

Выберите один ответ.

- a. шаблоны
- b. сервлеты
- c. контейнеры
- d. контуры

20 Баллов: 1

Какой процесс охватывает действия и задачи, выполняемые поставщиком, который снабжает заказчика программным продуктом или услугой:

Выберите один ответ.

- a. Процесс поставки
- b. Процесс сопровождения
- c. Процесс приобретения
- d. Процесс управления

Учреждение образования
«Гомельский государственный университет имени Франциска Скорины»

УТВЕРЖДАЮ
Проректор по учебной работе
ГГУ имени Ф. Скорины

_____ И.В. Семченко
(подпись)

_____
(дата утверждения)

Регистрационный № УД-_____/уч.

ПРОЕКТИРОВАНИЕ СИСТЕМ ОБРАБОТКИ ДАННЫХ

Учебная программа учреждения высшего образования
по учебной дисциплине для специальности:
1 – 53 01 02 Автоматизированные системы обработки информации.

2016 г.

Учебная программа составлена на основе *образовательного стандарта ОСВО 1-53 01 02 2013 г. и учебного плана УВО, регистрационный № 1-53-01-13, дата утверждения 25.08.2013 г.*

СОСТАВИТЕЛЬ:

В.Д. Левчук, заведующий кафедрой АСОИ

РЕКОМЕНДОВАНА К УТВЕРЖДЕНИЮ:

Кафедрой автоматизированных систем обработки информации (протокол № *9* от *12.04.2016*);

Научно-методическим советом ГГУ имени Ф.Скорины (протокол № *7* от *01.06.2016*).

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Изучение дисциплины компонента учреждения высшего образования «Проектирование систем обработки данных» предусмотрено учебным планом подготовки специалистов специальности 1-53 01 02 – «Автоматизированные системы обработки информации».

Проектирование систем обработки данных является одним из основных направлений разработки программных систем. Такие системы представляют собой сложные программно-аппаратные комплексы, требующие от их разработчиков знаний не только языков программирования и аппаратного обеспечения, но и предметной области, для которой создается система. Командная разработка требует тщательного анализа задачи, причем, с помощью средств и методов, понятных и доступных всем вовлеченным в проект специалистам в разных областях знаний, а также подробного планирования и координирования всех работ. Кроме того, высокие требования к качеству программного продукта требуют подробного документирования проекта и применения средств автоматизации процессов, связанных с работой над проектом.

Цель изучения дисциплины – получить специальные знания, умения и навыки, необходимые инженеру по информационным технологиям в процессе проектирования автоматизированных систем обработки данных.

В результате изучения дисциплины студент должен:

знать:

– теоретические и прикладные аспекты проектирования систем обработки данных;

– ключевые понятия архитектуры программного обеспечения предприятия;

уметь:

– использовать технологию объектно-ориентированного проектирования информационных систем;

владеть:

– прикладным программным обеспечением для проектирования автоматизированных систем обработки данных.

Специалист должен быть компетентен в следующих видах деятельности: производственно-технологической, проектно-конструкторской, экспертно-консультационной, научно-исследовательской и образовательной, организационно-управленческой.

Специалист должен обладать следующими видами компетенций:

1 Требования к академическим компетенциям специалиста:

АК-1 Уметь применять базовые научно-теоретические знания для решения теоретических и практических задач.

АК-2. Владеть системным и сравнительным анализом.

АК-3. Владеть исследовательскими навыками.

АК-4. Уметь работать самостоятельно.

АК-5. Быть способным порождать новые идеи (обладать креативностью).

АК-12. Владеть основными методами, способами и средствами получения, хранения, переработки информации, наличием навыков работы с компьютером как средством управления информацией.

2 Требования к профессиональным компетенциям специалиста:

ПК-11. Проводить анализ эффективности функционирования систем обработки информации и выявлять узкие места по производительности и надежности.

ПК-12. Выявлять и устранять уязвимость систем обработки информации к угрозам безопасности.

ПК-13. Выполнять реконфигурацию баз данных и системного программного обеспечения под условия применения.

ПК-14. Выполнять обновление системного и прикладного программного обеспечения.

ПК-15. Выявлять актуальные проблемы развития и совершенствования информационных технологий.

ПК-17. Выделять области эффективного применения различных методов и средств реализации информационных технологий.

ПК-18. Консультировать потребителей по вопросам выбора эффективных методов решения задач, связанных с представлением, хранением, отображением, передачей и аналитической обработкой информации.

Дисциплина компонента учреждения высшего образования «Проектирование систем обработки данных» изучается студентами 4 курса дневной формы обучения специальности 1-53 01 02 – «Автоматизированные системы обработки информации»; студентами 5 курса заочной формы обучения специальности 1-53 01 02 – «Автоматизированные системы обработки информации»; студентами 4 курса заочной интегрированной со средним специальным образованием формы обучения специальности 1-53 01 02 – «Автоматизированные системы обработки информации».

Дневная форма обучения: всего часов по плану - 134, аудиторное количество часов – 64; из них: лекционных занятий – 32 (в том числе УСР – 12), практических занятий – 32.

Форма отчётности – экзамен в 7 семестре.

Заочная форма обучения: всего часов по плану - 134, аудиторное количество часов – 16, из них: лекционных занятий – 12, практических занятий – 4.

Форма отчётности – экзамен в 9 семестре.

Заочная форма обучения (интегрированная на основе среднего специального образования): всего часов по плану - 66, аудиторное количество часов – 8, из них: лекционных занятий – 6, практических занятий – 2.

Форма отчётности – экзамен в 7 семестре.

Содержание учебного материала

Тема 1. Понятие жизненного цикла системы обработки данных

Определение системы обработки данных. Проблемы при разработке программного обеспечения предприятия. Развитие программного обеспечения предприятия. Сущность жизненного цикла.

Тема 2. Процессы жизненного цикла

Основные процессы жизненного цикла. Вспомогательные процессы жизненного цикла. Организационные процессы жизненного цикла. Взаимосвязь между процессами.

Тема 3. Обзор методов разработки программного обеспечения

Интуитивный подход. Каскадный процесс. Итерационный процесс. Рациональный унифицированный процесс. Экстремальное программирование и разработка по функциям.

Тема 4. Структурный подход к проектированию программного обеспечения

Сущность структурного подхода. Метод функционального моделирования. Состав функциональной модели. Построение иерархии диаграмм. Пример использования метода.

Тема 5. Архитектура программного обеспечения

Определение программной архитектуры. Необходимость программной архитектуры. Ключевые понятия архитектуры программного обеспечения предприятия. Декомпозиция. Многоуровневое представление. Компоненты. Шаблоны. Фреймворки.

Тема 6. Объектно-ориентированный подход к проектированию программного обеспечения

Сущность объектно-ориентированного подхода. Обзор языка моделирования UML. Состав объектной модели. Прикладное программное обеспечение для проектирования автоматизированных систем обработки данных.

Тема 7. Анализ требований клиента

Моделирование прецедентов. Идентификация исполнителей. Выявление прецедентов. Отношения между прецедентами. Диаграммы прецедентов. Диаграммы последовательностей. Диаграммы деятельности.

Тема 8. Создание профиля UML

Анализ прецедентов. Граничные объекты. Объекты-сущности. Объекты управления. Диаграммы взаимодействий. Диаграммы классов. Диаграммы состояний. Диаграммы развертывания.

Тема 9. Моделирование компонентов платформы JEE в UML

Назначение платформы JEE. Составные части платформы JEE. Контейнеры. Сервлеты. Компоненты JSP. Компоненты EJB. Шаблон MVC. Программные интерфейсы платформы.

Тема 10. Практические исследования

Постановка задачи. Обоснование и допущения. Требования к проекту. Начальная стадия. Стадия развития. Стадия конструирования. Стадия перехода.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА
(Дневная форма обучения)

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	Понятие жизненного цикла системы обработки данных	2	2				2	
	Определение системы обработки данных. Проблемы при разработке программного обеспечения предприятия. Сущность жизненного цикла.							Дискуссия
2	Процессы жизненного цикла	2	2				2	
	Основные процессы жизненного цикла. Вспомогательные процессы жизненного цикла. Организационные процессы жизненного цикла.							Рефераты
3	Обзор методов разработки программного обеспечения	2	2				2	
	Интуитивный подход. Каскадный процесс. Рациональный унифицированный процесс. Экстремальное программирование и разработка по функциям.							Рефераты
4	Структурный подход к проектированию программного обеспечения	2	4				2	
	Сущность структурного подхода. Метод функционального моделирования. Метод функционального моделирования. Пример использования метода.							Практическая работа
5	Архитектура программного обеспечения	2	2				2	
	Определение программной архитектуры. Необходимость программной архитектуры. Ключевые понятия архитектуры программного обеспечения предприятия.							Дискуссия
6	Объектно-ориентированный подход к проектированию программного обеспечения	2	4				2	

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
	Сущность объектно-ориентированного подхода. Обзор языка моделирования UML. Состав объектной модели. Прикладное программное обеспечение для проектирования автоматизированных систем обработки данных.							Практическая работа
7	Анализ требований клиента	2	4					
	Идентификация исполнителей. Выявление прецедентов. Диаграммы прецедентов, последовательностей, деятельности.							Практическая работа
8	Создание профиля UML	2	4					
	Анализ прецедентов. Типы объектов модели. Диаграммы взаимодействий, классов, состояний, развертывания.							Практическая работа
9	Моделирование компонентов платформы JEE в UML	2	4					
	Назначение платформы JEE. Составные части платформы JEE. Сервлеты и компоненты JSP. Шаблон MVC.							Дискуссия
10	Практические исследования	2	4					
	Постановка задачи. Допущения и требования к проекту. Реализация модели проекта на различных стадиях.							Практическая работа
	Всего	20	32				12	

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА
(Заочная форма обучения)

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	Понятие жизненного цикла системы обработки данных	2						
	Определение системы обработки данных. Проблемы при разработке программного обеспечения предприятия. Сущность жизненного цикла.							Дискуссия
2	Обзор методов разработки программного обеспечения	2						
	Интуитивный подход. Каскадный процесс. Рациональный унифицированный процесс. Экстремальное программирование и разработка по функциям.							Рефераты
3	Структурный подход к проектированию программного обеспечения	2	2					
	Сущность структурного подхода. Метод функционального моделирования. Состав функциональной модели. Пример использования метода.							Практическая работа
4	Объектно-ориентированный подход к проектированию программного обеспечения	2						
	Сущность объектно-ориентированного подхода. Обзор языка моделирования UML. Состав объектной модели. Прикладное программное обеспечение для проектирования автоматизированных систем обработки данных.							Практическая работа
5	Анализ требований клиента	2						
	Идентификация исполнителей. Выявление прецедентов. Диаграммы прецедентов, последовательностей, деятельности.							Практическая работа
6	Создание профиля UML	2	2					
	Анализ прецедентов. Типы объектов модели. Диаграммы взаимодействий, классов, состояний, развертывания.							Практическая работа
	Всего	12	4					

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА
(Заочная интегрированная на основе среднего специального образования форма обучения)

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов					Количество часов УСР	Форма контроля знаний
		Лекции	Практические занятия	Семинарские занятия	Лабораторные занятия	Иное		
1	Обзор методов разработки программного обеспечения	2						
	Интуитивный подход. Каскадный процесс. Рациональный унифицированный процесс. Экстремальное программирование и разработка по функциям.							Рефераты
2	Структурный подход к проектированию программного обеспечения	2						
	Сущность структурного подхода. Метод функционального моделирования. Состав функциональной модели. Пример использования метода.							Практическая работа
3	Объектно-ориентированный подход к проектированию программного обеспечения	2	2					
	Сущность объектно-ориентированного подхода. Обзор языка моделирования UML. Состав объектной модели. Прикладное программное обеспечение для проектирования автоматизированных систем обработки данных.							Практическая работа
	Всего	6	2					

информационно - методическАЯ часть

Формы контроля знаний

– Дискуссия.
– Рефераты.
– Практическая работа.
– Экзамен.

Перечень программного обеспечения

– Star UML.
– ERwin.
– Eclipse.

Рекомендации по организации и выполнению УСР

Для углубленного самостоятельного изучения учебного материала в рамках УСР выделяются следующие темы дисциплины:

- Понятие жизненного цикла системы обработки данных
- Процессы жизненного цикла
- Обзор методов разработки программного обеспечения
- Структурный подход к проектированию программного обеспечения
- Архитектура программного обеспечения
- Объектно-ориентированный подход к проектированию программного обеспечения

Самостоятельное изучение материала данных тем преследует цель получения навыков работы с современными версиями программного обеспечения для проектирования систем обработки данных.

Форма выполнения заданий – индивидуальная.

Форма контроля выполнения заданий – дискуссия, реферат, практическая работа.

Рекомендуемая литература

Основная

1. Буч, Г. UML. Классика CS: справочник по языку UML/ Г. Буч, Д. Рамбо, А. Джекобсон – СПб.: Питер, 2006. – 736 с.

2. Буч, Г. Объектно-ориентированный анализ и проектирование с примерами приложений на C++, 2-е изд. – М.: Издательство Бином, СПб.: Невский диалект, 2015. – 560 с.

3. Вендров, А.М. Проектирование программного обеспечения экономических информационных сетей: Учебник. – М.: Финансы и статистика, 2012. – 352 с.

4. Маклафлин, Б. Объектно-ориентированный анализ и проектирование/ Б. Маклафлин, Г. Поллайс, Д. Уэст – СПб.: Питер, 2013. – 608 с.

6. Орлов, С.А. Технологии разработки программного обеспечения: Учебник, 2-е изд. – СПб.: Питер, 2014. – 528 с.

7. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка/ Дж. Рамбо, М. Блаха – СПб.: Питер, 2007. – 544 с.

Дополнительная

8. Единая система программной документации: [Каталог нормативных документов по стандартизации]: ГОСТ 19.001-77 - ГОСТ 19.701-90. - Минск: БелГИИС, 1999. - Т.2.

9. Стандарт ISO/IEC 12207:2005 ”Information Technology – Software Life Cycle Processes”.

ПРОТОКОЛ СОГЛАСОВАНИЯ УЧЕБНОЙ ПРОГРАММЫ
ПО ИЗУЧАЕМОЙ УЧЕБНОЙ ДИСЦИПЛИНЕ
С ДРУГИМИ ДИСЦИПЛИНАМИ СПЕЦИАЛЬНОСТИ

Название дисциплины, с которой требуется согласование	Название кафедры	Предложения об изменениях в содержании учебной программы по изучаемой учебной дисциплине	Решение, принятое кафедрой, разработавшей учебную программу (с указанием даты и номера протокола)
Объектно-ориентированное программирование и проектирование	АСОИ		Рекомендовать к утверждению учебную программу в представленном варианте протокол № ___ от ___.___.200__
Информационные технологии и психология управления	АСОИ		Рекомендовать к утверждению учебную программу в представленном варианте протокол № ___ от ___.___.200__

ДОПОЛНЕНИЯ И ИЗМЕНЕНИЯ К УЧЕБНОЙ ПРОГРАММЕ
ПО ИЗУЧАЕМОЙ УЧЕБНОЙ ДИСЦИПЛИНЕ
на _____/_____ учебный год

№№ пп	Дополнения и изменения	Основание

Учебная программа пересмотрена и одобрена на заседании кафедры АСОИ
(протокол № ____ от _____ 20___ г.)

Заведующий кафедрой АСОИ
к.т.н., доцент _____ В.Д. Левчук

УТВЕРЖДАЮ
Декан физического факультета УО «ГГУ им. Ф. Скорины»
к.ф.-м.н., доцент _____ Д.Л. Коваленко