АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ
ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ
«Международный Колледж Бизнеса и Дизайна»
(АНО ПОО «Международный Колледж Бизнеса и Дизайна»)

УТВЕРЖДАЮ
Директор АНО ПОО «МКБиД»
_____ Н.Н.Репин
«28» августа 2023 г.

КОНСПЕКТ ЛЕКЦИЙ

по учебной дисциплине МДК 02.01 Технологии разработки программного обеспечения

программы подготовки специалистов среднего звена

по специальности: 09.02.07 «Информационные системы и программирование»

2023 год

Темы:

1.Основные понятия и определения. Технология разработки ПО. Программное обеспечение.
2. Проект. Основные подходы к определению проекта. Системный проект.
3.Деятельный проект.

Цель: изучить основных понятий. Научиться отличать проект от простого задания. Запомнить основные характеристики ПО.

Термин «технология» – он подчеркивает аналогию между созданием программного продукта и промышленным производством.

Создание программного продукта это у нас творческий процесс, регламентируемый стандартами

Словно фиксирует ту точку зрения, что программирование, несмотря на интеллектуальность и творческий характер *этой деятельности, нуждается в организации и регламентировании, наборе соглашений и правил*, не говоря уже об инструментальном обеспечении.

Технология разработки программного обеспечения – это совокупность процессов и методов создания программного продукта.

Что вообще такое технология? Это совокупность каких либо методов.

Пример : забить гвоздь: технология - это как нужно правильно поставить под определенным углом гвоздь, как его нужно придерживать, и последний этап это какую силу удара нужно приложить, чтобы гвоздь забился.

А инструментами разработки будут уже молоток или что либо чем мы будем осуществлять этот удар.

Технология разработки программного обеспечения – это система инженерных принципов для создания экономичного ПО с заданными характеристиками качества.

Близким по смыслу к термину технология разработки ПО является термин *программная инженерия*.

Любая технология разработки ПО базируется на некоторой методологии или совокупности методологий.

Под **методологией** понимается система принципов и способов организации процесса разработки программных средств.

Методология-это различные принципы и способы разработки ПО.

Цель методологии разработки ПО – внедрение методов разработки ПС, обеспечивающих достижение соответствующих характеристик качества.

Целью методологии является внедрение определенные методы, принципы, которые позволят нам улучшить качество разработки или скорость разработки определенных методов.

базовых принципа разработки ПС:

1. модульный
2. объектно-ориентированный.

1. Разработка ***модульных*** ПС основывается на использовании *структурных методов проектирования*, целью которых является разбиение по некоторым правилам проектируемого программного средства на структурные компоненты.
2. **Объектно-ориентированная разработка** базируется на применении объектных методов, к которым относятся методологии объектно-ориентированного анализа, проектирования и программирования.

Идеологически ООП — подход к программированию как к моделированию информационных объектов. То как мы управляем объектами, каким свойствами их наделяем

программное обеспечение (software) – полный набор или часть программ, процедур, правил и связанной с ними документации системы обработки информации.

Это комплекс программ и ее документации

Программное средство – *ограниченная часть программного обеспечения* системы обработки информации, имеющая определенное функциональное назначение.

Проектировании программного средства сейчас обозначается как РАЗРАБОТКА

разработка = анализ + проектирование + программирование (кодирование) + тестирование + отладка Иногда сюда также включают “сопровождение”

Приступая к разработке новому проекту, главное — решить, с чего начать. Как правило, от него ждут многого, а время и ресурсы ограничены.

Я объясню, чем проекты отличаются от других видов деятельности, и расскажу вкратце о планировании, организации и управлении проектами.

Разберемся, что такое проект.

Проект (от англ. project — то, что задумывается и планируется)- это создание чего-либо к установленному сроку, он имеет планируемую дату завершения, после которой команда проектантов распускается.

В современной литературе по управлению проектами можно выделить

два основных подхода к определению проекта:

1. **Системный проект** — временное предприятие для создания уникальных продуктов, услуг или результатов.

Создание одноразового специфического проекта.

2. **Деятельностный проект**.- проект в самом широком смысле может пониматься как творческая, разумная, целеполагающая деятельность субъекта.

Что представил то и реализовал, самые креативные мысли и подходы

Проектируемая система, в результате чего мы получаем каркас для создания других проектов, аналогов.

Первый подход Системный подход определяет проект как систему временных действий, направленных на достижение неповторимого, но в то же время определенного результата.

Системный подход к определению проекта предопределяет основные его характеристики.

В системном походе Проекты могут быть :

1. разнообразными
2. многоплановыми

Однако все они имеют следующие **общие характеристики проектов**:

1. **разовость** — все проекты представляют собой разовое явление.

Они приходят и уходят, появляются и исчезают, оставляя после себя конкретные результаты, существенно отличаясь от наших повседневных обязанностей и деятельности;

2. **уникальность** — нет двух одинаковых проектов.

Каждый из них, независимо от его результатов, в своей основе имеет что-то неповторимое, характерное только для него;

3. **инновационность** — в процессе реализации проекта всегда создается нечто новое. Изменения могут быть большими или маленькими;

4. **результативность** — все проекты имеют вполне определенные результаты.

Это может быть новый дом, напечатанная книга, модифицированная структура компании, победа на выборах.

Все проекты нацелены на получение определенных результатов, иными словами, они направлены на достижение целей;

5. **временная локализация** — все проекты ограничены четкими временными рамками (дедлайнами).

Все перечисленные характеристики взаимосвязаны и задают определенные рамки проекта, три его измерения, критерии, по которым можно оценить любой проект

Критерии оценки проекта:

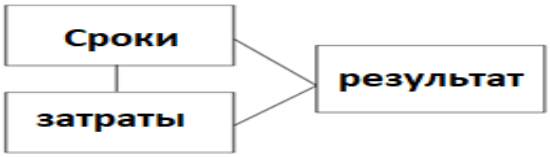


схема измерения проекта

Планирование и реализация проекта всегда связаны с тремя главными вопросами:

1. — сколько времени это займет?;
2. — во сколько это обойдется?;
3. — совпадет ли конечный результат с тем, что мы намечали вначале?.

Первый вопрос выводит на первый план проблему временных рамок, установленных для реализации всего проекта и отдельных его этапов.

Второй вопрос привлекает наше внимание к стоимости проекта

Третий вопроса касается о результативности проектной деятельности.

1. **Срок реализации проекта-** полный жизненный цикл проекта, от создания до закрытия проекта

Наш дедлай по проекту. Максимальный срок его завершения

2. **бюджет проект**- представляет собой план затрат, необходимых для его исполнения, в стоимостном выражении. Бюджет проекта включает затраты на закупку материалов, выплату заработной платы , услуги сторонних организаций, амортизацию зданий, техники, оборудования

в какую сумму обойдется проект

Как правило, бюджет формируется в разрезе этапов проекта – участков работ, выполнение которых контролируется индивидуально.

Основными параметрами, влияющими на бюджет проекта, являются:

1. длительность работ
2. количество участников
3. качество используемой техники
4. специфические требования к результату.

3. **Результат проекта**- является достижение поставленной основной цели проекта

совпадение конечного результата с тем, что мы планировали в начале создания проекта

Задача проект-менеджера — найти оптимальное соотношение этих трех ограничений проекта, с которыми неразрывно связаны интересы участников проекта.

В данном смысле задача трансформируется в соблюдение баланса интересов клиента;

как таковые ограничения становятся «вторым планом» действия в проекте, заглавная роль в котором принадлежит именно интересам.

Второй подход — Деятельностный — трактует проект как деятельность субъекта по переводу объекта из наличного состояния в состояние желаемого будущего, которое наиболее полно отвечает его представлениям.

может пониматься как творческая, деятельность , что представил то и реализовал, мы проектируем наши идеи.

Сущность любого проекта заключается в деятельности. Принимая во внимание определения проекта, можно дать определение проектной деятельности, или проектированию.

Проектирование — это процесс создания прототипа, прообраза предполагаемого или возможного объекта или состояния.

Проектант как бы выбирает из множества путей, версий развития объекта именно ту, которая в максимальной степени соответствует шкале его ценностей, предпочтений, замыслов.

• Многоуровневый шаблон. система программы разбивается на уровни, которые отображаются на диаграмме. При этом каждый уровень может вызывать только один другой уровень, находящийся ниже него.

Это дает возможность вносить изменения в определенные компоненты программы, не затрагивая другие области. Но это усложняет структуру архитектуры и делает ее довольно нагруженной, что влияет на производительность.

• Шаблон посредника. *Если программа состоит из большого количества модулей, прямое взаимодействие выглядит довольно сложным и запутанным. Чтобы облегчить его, внедряется посредник, который позволяет модулям наладить простое взаимодействие.* Функциональная совместимость компонентов сразу возрастает, но посредник – слабое звено системы. В случае выхода его из строя, может перестать работать вся система.

• Модель – Представление – Контроллер. Суть шаблона в том, что интерфейс отделяется от данных. Это позволяет менять его, не нарушая принципов работы системы программы. Используется в программах, где необходимо регулярно менять интерфейс.

• **Клиент-серверный паттерн.** Архитектура программного обеспечения с использованием клиент-серверного шаблона является довольно востребованной в тех приложениях, в которых необходимо ограничить права доступа потребителей к определенному числу ресурсов. Подобный подход позволяет масштабировать программу и делает систему доступной и понятной.

• **Паттерн «Фабричный метод».** Позволяет добавлять новые объекты разных типов. Если использовать стандартные методы добавления, код будет расти, что снизит скорость работы приложения, кроме того, компоненты будут разбросаны по всему коду. Данный шаблон *позволяет облегчить процессы добавления новых объектов и сохраняет систему независимой.*

Проектная деятельность носит двойственный характер:

1. С одной стороны, это деятельность идеальная, поскольку она связана с планированием будущего, представление того, что должно быть.

Дает возможность представить, что будет в результате по завершению проекта

2. С другой стороны, это деятельность технологическая, так как она отражает процессы реализации того, что задумано.

Использование доп. Технологий для отображения процессов реализации, лишние денежные средства.

Для того чтобы точно осмыслить суть проектирования, необходимо соотнести его с понятиями, близкими по смыслу

1. **Прогнозирование** — форма предвидения, предположительная оценка будущего состояния объекта, условий его возникновения.

Мы даем прогноз по выполнению проекта

Прогноз служит основой для формулировки целей развития и стратегии их достижения.

2. **Планирование** — это научное и практическое обоснование определения целей, выявление задач, сроков, темпов, пропорций развития того или иного явления, его реализация.

Мы ставим цели задачи, сроки, планируем как что мы будем реализовывать

Отличие программы от проекта

1. В основе планирования всегда лежит некая программа действий, включающая в себя совокупность целевых установок.

В планирование включается программа

2. Программа лишь обозначает, прорабатывает необходимый набор, комплекс необходимых направлений деятельности, обозначает желаемые конечные цели и результаты, эффективность достижения этих целей.

3. **Конструирование** — это интеллектуальная деятельность, состоящая в целенаправленном построении в идеальной форме какого-либо объекта.

Здесь мы корректирует часть проекта, настоящее, внедряем какие-то инновации, конструируя желаемое состояние.

ЛЕКЦИЯ 2

Тема: Проектирование программного продукта. Элементы проектной деятельности.

Цель: изучить основных понятий проекта. Научиться отличать объект проектирования от субъекта. Изучить кто может выступать объектом проектирования.

Основными элементами проектной деятельности являются:

1. Субъектом проектирования кто проектирует
2. Объектом проектирования то что проектируем

1) Субъектом проектирования- всегда служат различные носители управленческой деятельности

Они бывают отдельные личности, так и организации, коллективы, ставящие своей целью преобразование действительности.

Кроме субъектов и проектирования, могут и должны быть:

1. — *органы принятия решений*, чьи функции связаны с обеспечением проектов, их утверждением, контролем над их реализацией;

2. *государственные и негосударственные организации, научные и экспертные советы*, способные взять на себя ответственность за разработку, обоснование, экспертизу проектов, способные привлечь внимание населения, СМИ к проектам;

3. *общественность, группирующаяся вокруг конкретных программ, проектов*.

2) Объектами проектирования могут быть :

1. *объекты материальной природы* - в результате реализации проекта появляется новый объект, вещь, предмет;

пример :может быть строительство нового административного здания или создание нового компьютера;

2. *назначения и функции старой вещи*- подобные объекты чаще связаны с техническим проектированием;

пример: доработка какого либо модуля ПО

3. *нематериальные (невещные) свойства и отношения* - проекты, которые направлены не на достижение материального результата, а на получение информации о клиентах, изменение нашего отношения к той или иной проблеме.

Пример: рекламные кампании;

В этих системах имеет значение и идейная конструкция — концепция и соответствующие инструменты внедрения идей в сознание людей.

4. *организации и структурные подразделения* в рамках проектирования организаций реализуются замыслы разного масштаба

пример: проектируются учреждения социальной службы, отрасли производства, управления

5. *мероприятия (акции)* -подготовка мероприятий может производиться с применением проектных методик.

Пример: массовым мероприятиям — спортивным, праздничным, общественным

6. *законопроекты.*

Каждый из выделенных объектов проектирования обладает определенной спецификой, определенными чертами.

При проектировании важно выявить закономерности, характерные для данного типа объектов, применяя особые методики наряду с общими принципами и подходами.

условия проектной деятельности или проектный фон-это совокупность внешних по отношению к объекту проектирования условий, существенно влияющих на его функционирование и развитие.

Речь идет о необходимости учета местных условий.

Какие-то возможности, альтернативы могут быть реализованы, а какие-то нет — это зависит от местных условий, окружения проекта, внешних ограничений.

Цель проектирования — разработка определенного будущего состояния системы, процессов, отношений.

Средства — совокупность приемов и операций для достижения цели.

В общем плане средства проектирования можно определить, как все то, при помощи чего получается, анализируется информация о состоянии процессов и тенденций их развития.

К средствам проектирования относится:

1. создаются словесные описания,
2. таблицы,
3. схемы,
4. сети взаимодействий.

Методы — это пути и способы достижения целей и решения задач.

В практике проектирования наиболее часто используются такие методы:

1. мозговой штурм,
2. экспертная оценка,
3. метод аналогий,

4. календарное планирование,
5. структурная декомпозиция,
6. имитационное моделирование,

В рамках проекта методы и средства конкретизируются совокупностью планируемых мероприятий.

Есть мероприятие, а мы уже выбираем методы и средства как мы его будем организовывать

мероприятия определяют направления, формы и содержание деятельности, привлекают дополнительные ресурсы, необходимые для реализации целей каждого этапа.

Мероприятия могут быть направлены:

1. непосредственно на решение проблемы,

2. а могут быть необходимы для их финансового обеспечения

(аукционы, платные услуги), для формирования благоприятного общественного мнения населения через СМИ.

Контрольные вопросы:

1. что такое субъект проектирования?
2. Что такое объект проектирования
3. Какие органы должны быть кроме субъектов проектирования?
4. Кто может быть объектами проектирования?
5. Что такое проектный фон?
6. Цель проектирования?
7. Что такое средства и методы проектирования?
8. Назовите методы проектирования

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Искусство IT-проектирования Скотт Беркун

ЛЕКЦИЯ 5

Тема: Мозговой штурм, как средство поиска решений по конфликтным ситуациям.
Цель: изучить основных методы проектирования.

В практике проектирования наиболее часто используются такие методы:

7. мозговой штурм,
8. экспертная оценка,
9. метод аналогий,
10. календарное планирование,
11. структурная декомпозиция,
12. имитационное моделирование,

1. Мозговой штурм

Метод мозгового штурма был создан в 1941 году Алексом Осборном — сотрудником американского рекламного агентства суперпрофессионалов «BBD&O»

В мозговом штурме принимает участие группа людей, состоящая из тим-лидера и специалистов. Как только тим-лидер поставил основную задачу, специалисты начинают высказывать свои идеи.

Если в мероприятии принимают участие люди различных должностей, рангов, чинов и социального статуса, то лучше всего, чтобы идеи предлагались именно *по возрастанию статуса*, во исключение психологического фактора «согласия с начальством».

Интересно ещё и то, что в большинстве случаев *в начале штурма все выдвигаемые идеи совершенно обычны и тривиальны*, однако по мере вовлечения участников в процесс и активизации мышления и творческого потенциала начинают появляться оригинальные и необычные идеи.

На протяжении всего процесса ведущий лидер все озвученные предложения. И уже после этого осуществляется их отбор, анализ и развитие.

Результатом и становится наиболее эффективный и оригинальный способ решения поставленной проблемы.

Принцип метода

Метод служит для оперативного решения проблем и основывается на стимулировании творческой активности людей, принимающих в нём участие и предлагающих максимальное количество всевозможных вариантов решения. После того, как все варианты озвучены, выбираются те, которые более всего подходят для успешной реализации на практике. Обычно мозговой штурм состоит из трёх обязательных этапов, различных по организации и правилам проведения.

Основные этапы мозгового штурма и правила его построения

1. ***Постановка проблемы***-Этот этап считается предварительным.

Он подразумевает чёткую формулировку проблемы, отбор участников и распределение их ролей

Изучаем предметную область, ставим цели и задачи мозгового штурма

2. ***Генерация идей-***Это основной этап и именно от него зависит успех всего предприятия.

По этой причине важно соблюдать следующие правила:

2.1 Максимальное количество идей, без любых ограничений

2.2 Принимаются даже фантастические, абсурдные и нестандартные идеи

2.3 Идеи можно и нужно комбинировать и улучшать

2.4 Не должно быть никакой критики или оценивания предлагаемых идей

3. ***Отбор, систематизация и оценка идей-*** Заключительный этап

насколько данный этап пройдёт успешно, зависит от согласованности работы участников и общего направления их мнений относительно решаемой задачи и предлагаемых решений. Все участники голосуют за лучшую идею

для мозгового штурма создаётся две группы.

1. В первую группу входят люди – *генераторы идей*, предлагающие решения.

2. А вторая группа состоит из так называемой *комиссии*, занимающейся обработкой предложенных решений.

Главные плюсы метода мозгового штурма

1. Каждый участник имеет свой опыт, и на основе этого опыта он предлагает различные варианты поиска решений.

У каждого свой опыт в итоге у каждого свое виденье решения проблемы-результат множество вариантов проектирования

2. сам процесс мозгового штурма обладает особым творческим потенциалом.

тем самым преобразуясь в коллективную и даже игровую деятельность.

3. царящая во время мозгового штурма дружественная и позитивная обстановка позволяет его участникам не только конструктивно воспринимать любую критику, но и импровизировать и использовать максимум своего потенциала, а также служит усилению доверия и положительного настроя.

Объединяет коллектив, делает его более сплочённым

Однако многие учёные, в частности, психологи, утверждают, что если работа команды участников штурма организована неправильно, то и результаты штурма будут очень низкими, сведя достоинства метода на нет.

Чтобы этого избежать следует придерживаться нескольких простых правил.

10 правил эффективного мозгового штурма:

1. **Предварительная подготовка**. Задача штурма должна быть озвучена минимум за 2-3 дня до его проведения. За это время участники смогут неплохо обдумать стоящую перед ними проблему и уже в самом начале штурма предложить несколько интересных идей.
2. **Много участников**. Чем больше народу-тем больше идей
3. **Уточнение поставленной задачи.** Заново воспроизвести задачу мозгового штурма, чтобы все точно думали об одной теме
4. **Записи**. Каждый участник должен делать записи и пометки. Чтобы не забыть не про одну идею
5. **Никакой критики**. Ни в коем случае не отвергайте предлагающиеся идеи, какими бы нелепыми или фантастическими они не казались.
6. **Максимальная генерация идей**. Каждый участник процесса должен понять, что ему нужно предлагать как можно больше идей.
7. **Привлечение других людей**. Если за время штурма не нашлось решения, можно привлечь к мозговому штурму людей, которые либо не присутствуют на штурме, либо вообще не имеют к нему никакого отношения.
8. **Модификация идей**. соединять две идеи (и более) в одну. Лучший вариант соединения идей предложенные людьми различного статуса, должности, ранга.
9. **Визуальное отображение**. Для удобства восприятия и повышения результативности мозгового штурма следует использовать маркерные доски, флэш-панели, плакаты, схемы, таблицы и т.п.
10. **Отрицательный результат**. С помощью такого моделирования можно способствовать выработке дополнительных идей, а также морально и психологически подготовить себя к любой ситуации.

Контрольные вопросы:

9. Принцип метода мозгового штурм
10. Опишите этапы мозгового штурма
11. Назовите достоинства мозгового штурма

12. Назовите правила мозгового штурма, опишите их
13. Что такое метод экстремальных оценок

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами
5. https://habr.com/post/189626/
6. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
7. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 6

Тема: Экспертные оценки. Способы измерения объектов.
Цель: изучить основные методы проектирования.

В практике проектирования наиболее часто используются такие методы:

13. мозговой штурм,
14. экспертная оценка,
15. метод аналогий,
16. календарное планирование,
17. структурная декомпозиция,
18. имитационное моделирование,

Методы экспертных оценок

Метод экспертных оценок - это фактически метод прогнозирования, основополагающим критерием которого является достижение согласия всех членов экспертной группы.

Метод экспертных оценок – так же являются частью обширной области теории принятия решений, основанном на мнения специалистов (экспертов) с целью последующего принятия решения (выбора).

Суть метода- При применении метода экспертных оценок проводится опрос специальной группы экспертов (5-7 человек) с целью определения определенных переменных величин, необходимых для оценки исследуемого вопроса. В состав экспертов следует включать людей с разными типами мышления - образное и словесно-логическое, что способствует успешному решению проблемы.

В случае сложных задач мы обращаемся к экспертам (5-7 человек) и полагаемся на их опыт и мнение и знания в этих областях

Существует две группы экспертных оценок:

1. ***Индивидуальные оценки*** - это использование мнений экспертов, которые сформулированы лично каждым из них самостоятельно без учета мнений других экспертов. К индивидуальным экспертных методов относятся:

   1.1 ***интервью -*** состоит в организации собеседования аналитика с экспертом, в ходе которой эксперт дает ответа на вопрос аналитика о факторах влияния на исследуемый объект

Собеседование один на один с экспертом

   1.2 ***анкетирования***.- аналитического экспертного оценивания заключается в предоставлении экспертом письменных ответов на вопросы анкеты.

Однако этот метод имеет недостатки, в частности эксперт может не понять вопросы анкеты, проявить субъективизм, нежелание критиковать руководство и оставлять свою письменный ответ и тому подобное.

основаны на использовании мнения отдельных экспертов, независимых друг от друга.

2. **Коллективные методы** обеспечивают формирование единого общего мнения в результате взаимодействия привлеченных специалистов-экспертов.

оценки основаны на использовании коллективного мнения экспертов.

Среди коллективных методов экспертной оценки выделяют:

   2.1 ***метод комиссии*** Метод заключается в выработке экспертами лучшего варианта достижения поставленной цели с учетом всех высказанных на совещании предложений, идей.

Положительной возможность привлечения для экспертов с широким диапазоном знаний из смежных областей науки и практики.

Негативным возможен субъективизм, имеющиеся стереотипы мышления, которые сложились у экспертов, их склонность к компромиссу.

2.2 ***Метод Дельфи*** - один из методов коллективной экспертной оценки, который предусматривает проведение экспертного опроса среди группы специалистов в несколько туров (чаще в 3-4 туры) для выбора лучшего из решений.

Эксперты выдвигают варианты и в несколько туров отбирают лучший метод

Слово главного эксперта не подлежавшее сомнения и принималось за истину.

2.3 ***Метод отстраненного*** оценки заключается в выборе оптимального независимого решения из числа высказанных экспертами на совещании. Работа совещания разделена на две части: выдвижение идей и их критический анализ.

Выдвигаются идеи экспертами и анализируются, выбирается самая подходящая

2.4 ***Конференция*** идей подобная мозгового штурма, но отличается от него темпом проведения совещаний и разрешенной короткой доброжелательной критикой идей в форме реплик и комментариев. При этом стимулируется сочетание нескольких предложений, фантазирование, что способствует повышению качества идей.

Аналог мозгового штурма, только с возможностью критиковать идеи и совмещать с другими

Способы измерения объектов:

1. **Ранжирование** – это расположение объектов в порядке возрастания или убывания какого-либо присущего им свойства. Ранжирование позволяет выбрать из исследуемой совокупности факторов наиболее существенный.

Выбирается самое главное свойство и относительно его сортируют по возрастанию

2. **Парное сравнение** — это установление предпочтения объектов при сравнении всех возможных пар. Здесь не нужно, как при ранжировании, упорядочивать все объекты, необходимо в каждой из пар выявить более значимый объект или установить их равенство.

Не нужно сортировать просто последовательно сравнивают идеи и остается лучшая

3. **Непосредственная оценка.** Выбирается фактор более значимый чем другие. В этом случае диапазон изменения характеристик объекта разбивается на отдельные интервалы, каждому из которых приписывается определенная оценка (балл), например, от 0 до 10. Именно поэтому метод непосредственной оценки иногда именуют также балльным методом.

Выбирается фактор максимально влияющий на метод проектирования, а эксперта просят расположить признаки влияющие на реализацию в порядке предпочтения от 0 до 10

Необходимым условием эффективного применения методов есть

1. достаточное знание эксперта по исследуемой проблеме, высокий уровень эрудиции, способность его давать четкие исчерпывающие ответы, к тому же экспромтом.

Компетентный в этой области

2. эксперт не должен быть заинтересован в том или ином варианте решения поставленной перед ним проблемы.

3. Эксперты подбираются по признаку их профессионального статуса - должности, ученой степени, стажа работы и др. Такой подбор способствует тому, что в число экспертов попадают высокопрофессиональные, с большим практическим опытом в данной области специалисты.

Учитывается должность, опыт, количество положительно решенных проектов

Контрольные вопросы:

14. Что такое метод экстремальных оценок
15. Принцип работы метода экстремальных оценок
16. Назовите группы экстремальных оценок
17. Назовите способы измерения объектов
18. Что такое эксперт, кто им выступает?
19. Назовите необходимые условия эффективности метода

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами
5. https://habr.com/post/189626/
6. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
7. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 7

Тема: Синектика, как метод разработки игровых элементов
Цель: изучить основные методы проектирования.

В практике проектирования наиболее часто используются такие методы:

19. мозговой штурм,
20. экспертная оценка,
21. метод аналогий,
22. календарное планирование,
23. структурная декомпозиция,
24. имитационное моделирование,

Аналогия (греч. analogia - соответствие) - сходство объектов (явлений, процессов) в каких-либо свойствах.

Синектика -совмещение разнородных элементов

Аналогия — метод, который не имеет большой доказательной силы. Сходство, на основании которого производится доказательство, может оказаться случайным, а при выборочном анализе признаков существенные признаки могут быть заменены на несущественные.

Выводы, умозаключения по аналогии не достоверны, а лишь в той или иной степени вероятны.

Они опираются на имеющиеся в реальной действительности необходимые связи и отношения между признаками явлений.

Аналогию можно определить и через моделирование (хотя чаще используется противоположный подход); в таком случае аналогией следует называть «перенос информации» от прототипа к модели и обратно

Обычная схема умозаключения по аналогии: если первый предмет имеет признаки A, B, C, D, а второй — A, B, C, то, по-видимому, второй предмет обладает и признаком D.

Назначение метода

1. *Поиск решений в различных областях* человеческой деятельности. Областью систематического применения аналогии является теория подобия, широко используемая в моделировании.

Мы ищем решение проблемы, не привязываясь не к какой специфике деятельности.

Если один проект похож по принципу работы на другой, то мы можем смоделировать похожий проект за кратчайший срок

2. *Использование аналогий* - один из самых универсальных эвристических приемов, мобилизующих интеллектуальные ресурсы для поиска новых идей и решения творческих задач.

Мы проводи (для каждого свое) сравнение проекта с подобием ситуации, которая случалось в нашей жизни

Цель метода

Максимально растормозить мышление, уменьшить влияние психологической инерции, найти оригинальное решение задачи.

Суть метода

Аналогии не дают ответа на вопрос о правильности предположения, но наводят на мысль о том или ином положении.

Аналогии в определенной мере делают незнакомое знакомым, позволяя увиденному сходству решить проблему известным способом

и знакомое незнакомым, давая возможность взглянуть на проблему с неожиданной стороны, что может натолкнуть на новое оригинальное решение.

На основе выявлений аналогии с техническими объектами в другой области, с помощью группы эвристических приемов осуществляется поиск новых идей и решений.

Это приводит к тому, что отдельное слово, наблюдение могут вызвать в сознании воспроизведение ранее пережитых мыслей, восприятия и "включить" информацию прошлого опыта для решения поставленной задачи.

План действий- этапы выбора осуществляется поиск решения , аналогии в жизни

Среди различных приемов аналогий выделяют четыре фундаментальных типа (классификации)

Классификация аналогий:

1. Прямая*—реальные*
2. Субъективные*—телесные,* то как мы чувствуем
3. Символические*—абстрактные*
4. фантастические аналоги*и—нереальные* , которые охватывают мысли и опыт людей.

Каждому типу присущи свои правила поиска аналогии.

Особенности методов классификации

1. **Прямая аналогия**. Прием, направленный на рассмотрение решений сходных проблем в самых разных областях человеческого знания, в природе. Особое внимание следует уделять биологической аналогии. У природы запас идей практически неисчерпаем.

Принцип строения, фи-и сохраняется

Пример: Мост и паутина, сердце и насос, строение кожи дельфина - мягкая обшивка для подводных лодок...

2. **Субъективная аналогия**. Личностная аналогия, эмпатия. Прием вхождения в чужую "шкуру", вживания в образ совершенствуемого объекта, пытаясь слиться с ним воедино, с целью понять и представить состояние самого объекта.

Например: что я буду испытывать в роли футбольного мяча? какие силы будут на меня действовать? что меня будет заставлять котиться?

Вхождение в роль кого-то или чего-то мы тренируем, как у актера, нужны навыки и знания, необходимо развивать творческое воображение.

3. **Символическая аналогия**. При формулировании задачи пользуются поэтическими сравнениями, образами и метафорами, отражающими сущность символической

аналогии. Необходимо наглядно показать суть конфликта, лежащего в основе проблемы.

Пример: Нахождение «СА» может облегчить прием "поиск названия книги":

Из формулировки проблемы выделяется ключевое слово и по нему осуществляется поиск

Затем необходимо в двух словах дать образное определение сути этого, содержащее парадокс.

4. **Фантастическая аналогия**. Прием, при котором для решения задачи предлагается ввести какие-либо нереальные, фантастические средства (например, волшебную палочку) или персонажи, выполняющие то, что требуется по условию задачи. Как эту задачу решили бы сказочные персонажи?

Пример: ролики, коньки====сапоги скороходы

Что бы стало, если... (имя существительное)... (глагол)? Случайным образом выбирается имя существительное и глагол. Затем дается ответ.

Для того чтобы аналогия была доказанной и по своей форме напоминала индуктивный или дедуктивный вывод, необходимо соблюсти следующие условия.

Аналогия считается доказана при соблюдении условий:

1. аналогия должна основываться на сходстве максимального числа существенных признаков;

чем больше сходства, тем больше вероятность попадания в яблочко

2. связь между неизвестным, искомым признаком и остальными (известными) признаками должна быть предельно тесной и доказуемой;

связь, по которым мы проводим аналогию должна быть понятна (железная рука- рука человека)

3. аналогия не должна приводить к утверждению абсолютного сходства между аналогом и исследуемым предметом;

100% сходства не должно быть, так как мы придумываем что то новое, оно оставляет за собой часть свойств уникальных

4. исследование сходных признаков должно дополняться исследованием всех известных различий между аналогом и изучаемым объектом.

Исходный объект должен добавлять признаки (аналоги), которые мы нашли в другом объекте т.е. обувь+сапоги скороходы= ролики

Достоинства метода

Аналогии играют важную роль при выдвижении гипотез как средство уяснения проблемы и направления ее решения.

Мы лучше вживаемся в роль и понимаем проблему изнутри, что нужно, какие ф-и…

Недостатки метода

Негативное влияние эмпатии (сопереживание) на нервную систему.

Мы можем сильно проникнуться проектом, а как следствие будет болеть голова, или будем нервничать.

Пример: когда отыгрываем роль выбранного персонажа, мы за него переживаем, умер, мы расстроились.

Ожидаемый результат

Максимально новый продукт обладающий определенными свойствами признаков.

Контрольные вопросы:

20. Что такое метод экстремальных оценок
21. Принцип работы метода экстремальных оценок
22. Назовите группы экстремальных оценок
23. Назовите способы измерения объектов
24. Что такое эксперт, кто им выступает?
25. Назовите необходимые условия эффективности метода

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами
2. https://habr.com/post/189626/
3. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
4. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 8-10

Темы:

8.Календарное планирование работ по проекту. Диаграммы Ганта

9.Кан-бан методология, структура и принцам распределения работ и задач.

10. Сетевая модель. Объектное календарный график. Метод критических путей

Цель: изучить основные методы проектирования.

В практике проектирования наиболее часто используются такие методы:

25. мозговой штурм,
26. экспертная оценка,
27. метод аналогий,
28. календарное планирование,
29. структурная декомпозиция,
30. имитационное моделирование,

Календарное планирование

План проекта представляет собой организованную структуру документированной информации. Данная информация используется для планирования, организации и контроля деятельности проектной команды.

Календарное планирование-это тот же самый план проекта, только разделенный по времени выполнения

Календарное планирование в управлении проектами – это ключевой и важный процесс, результатом которого является утвержденный руководством компании календарный план проекта (часто его называют еще планом-графиком, календарным графиком, планом управления проектом).

Цель календарного планирования – *создать максимально точный план проекта* с учетом плановых и прогнозных сроков выполнения задач (работ), их длительностей, а также оценить возможные трудозатраты по задачам.

Правильно сформированный календарный план является основным элементом управления проектом.

В календарном планировании проекта имеются несколько дат завершения проекта, такие как:

Даты завершения проекта:

1. **Плановая – дата**, *устанавливаемая при создании задач* и контрольных точек в плане проекта;
2. **Фактическая – дата** фактического выполнения контрольной точки и/или задачи. *Устанавливается* автоматически по *факту выполнения задачи*
3. **Прогнозная** – *предполагаемая дата завершения задачи* и/или контрольной точки. Устанавливается исполнителем задачи или менеджером проекта.

Для того, чтобы достигнуть и определить даты завершения мы должны определиться с методами календарного планирования сначала и разобраться с рабочими графиками проекта.

Цель рабочих графиков с одной стороны — детализация объектного календарного плана и с другой — своевременная реакция на всевозможные изменения обстановки на стройке.

Детализация работ и своевременное реагирование на отклонения по времени их выполнения

Рабочие графики — наиболее распространенный вид календарного планирования. Как правило, они составляются очень быстро и зачастую имеют упрощенную форму.

как показывает практика, не всегда должным образом оптимизируются.

составляются лицами, непосредственно участвующими в этой стройке.

Методы календарного планирования производства:

1. ленточные графики Ганта (диаграммы Ганта)
2. Канбан (доска со стиками)
3. метод сетевого планирования производства
4. объемно-календарные графики

1. диаграммы Ганта

это популярный тип столбчатых диаграмм (гистограмм), который используется для иллюстрации плана, графика работ по какому-либо проекту. Диаграмма Ганта представляет собой отрезки, размещенные на горизонтальной шкале времени. Каждый отрезок соответствует отдельному проекту, задаче или подзадаче. Проекты, задачи и подзадачи, составляющие план, размещаются по вертикали. Начало, конец и длина отрезка на шкале времени соответствуют началу, концу и длительности задачи.

это таблица «работы (объекты) — время», в котором продолжительность работ изображается в виде горизонтальных отрезков линий.

Такой график обеспечивает возможности оптимизации самым разнообразным критериям, в том числе по равномерности использования рабочей силы, механизмов, строительных материалов

Достоинство линейных графиков является также их наглядность и простота.

недостатком линейных графиков является сложность их корректировки при нарушениях первоначальных сроков работ или изменении условий их проведения.

Эти недостатки устраняются при другой форме календарного планирования – сетевых графиках.

Диаграмма Ганта нужна, чтобы наглядно представить все этапы работы. Она показывает:

- задачи, включённые в проект;
- их продолжительность;
- даты начала и окончания проекта;
- время, которое занимает каждая задача;
- исполнителей, работающих над задачами;
- способы объединения задач.

Всё это позволяет оценить все ресурсы и взаимосвязи задач. А значит, запланировать работу так, чтобы не пришлось глобально пересматривать подход, менять команду или инструменты.

Создание диаграммы Ганта состоит из пяти главных шагов:

- определения временных рамок;
- добавления задач и подзадач;
- описания зависимости между задачами;
- добавления вех;
- обновления работы.

Определение временных рамок — первый шаг. Диаграмма Ганта — это визуализация хронологии проекта. На первом этапе назначают даты начала и завершения проекта — их размещают в начале и в конце графика.

График состоит из столбиков: это могут быть часы, дни, недели в зависимости от продолжительности проекта. Задачи размещают в столбиках в соответствии с тем, когда над ними будут работать.

Проект разбивают на задачи и подзадачи. Для каждой из них назначают даты начала и завершения. Потом задачи размещают на графике. Сразу можно назначить ответственных за выполнение задач.

Для дифференциации задач используют разные цвета. Например, можно пометить синим этапы, над которыми работает отдел маркетинга, а зелёным — этапы, над которыми работает отдел продаж.

Некоторые задачи невозможно начать без выполнения других. Например, нельзя начать тестирование приложения, если оно не готово. Связи между задачами на диаграмме Ганта чаще всего отмечают стрелками.

Потом на диаграмму добавляют вехи. Это даты, контрольные точки, обозначающие завершение больших частей работы. Вехи показывают, какие задачи нужно завершить к контрольным точкам, и помогают расставить приоритеты.

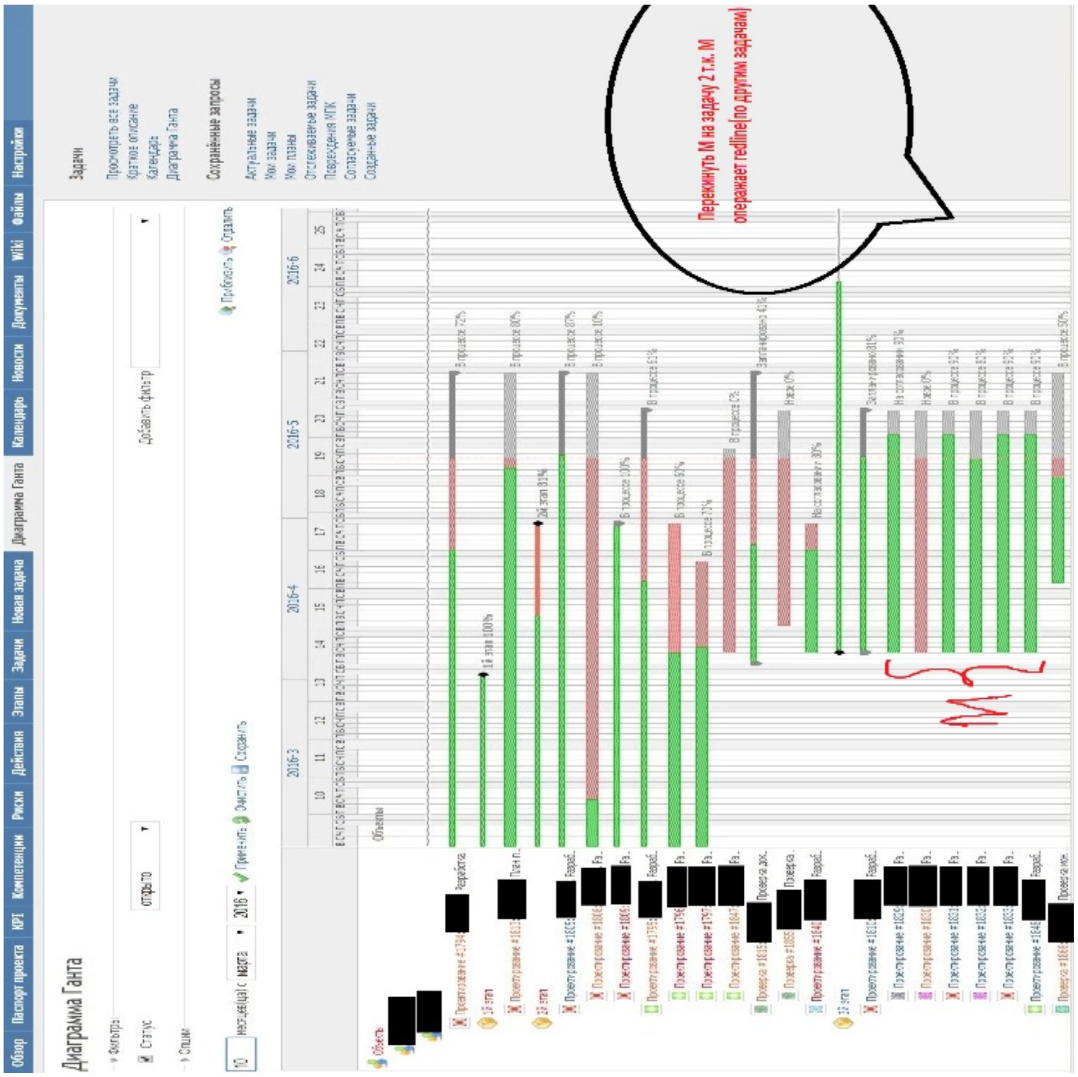
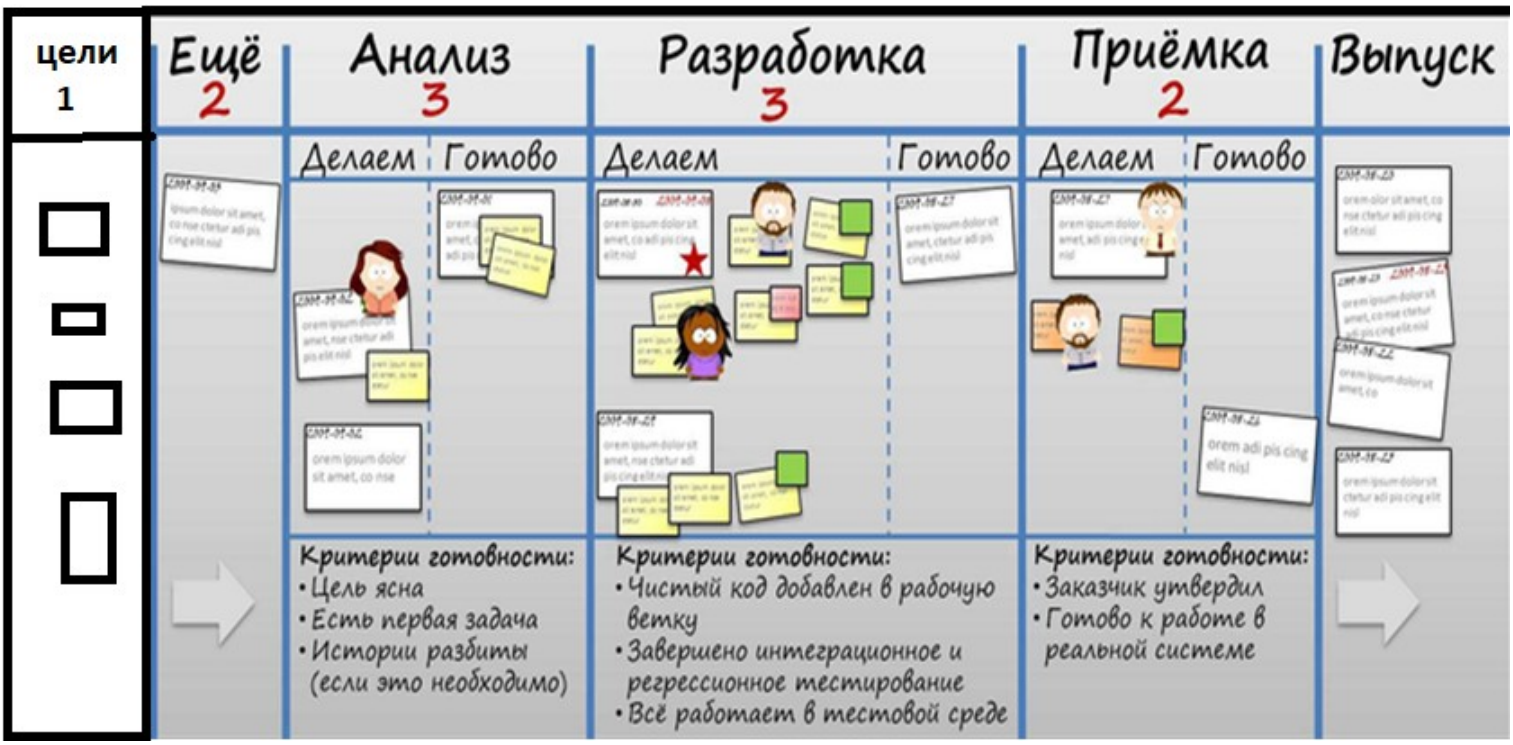


диаграмма Ганта

2. Канбан

“Кан” значит видимый, визуальный, и “бан” значит карточка или доска.



Только карточки, на которых указаны информация о сроках выполнения, описание или номер процесса и имя исполнителя, прикрепляется к доске с расчерченными колонками:

1. **Цели проекта**:

Необязательный, но полезный столбец. Сюда можно поместить высокоуровневые цели проекта, чтобы команда их видела и все про них знали.

2. **Очередь задач**:

Тут хранятся задачи, которые готовы к тому, чтобы начать их выполнять. Всегда для выполнения берется верхняя, самая приоритетная задача и ее карточка перемещается в следующий столбец.

3. **Разработка**:

Тут задача висит до тех пор, пока разработка фичи не завершена. После завершения она передвигается в следующий столбец.Или, если архитектура не верна или не точна — задачу можно вернуть в предыдущий столбец.

4. **Тестирование**:

В этом столбце задача находится, пока она тестируется. Если найдены ошибки — возвращается в Разработку. Если нет — передвигается дальше.

5. **Закончено**:

Сюда стикер попадает только тогда, когда все работы по задаче закончены полностью. Над

колонками обычно пишется число — лимит, указывающий на максимальное количество процессов в ней.

Методология базируется на культуре взаимного уважения и работе в команде, что обеспечивает успех, целесообразность работы и высокую вовлеченность сотрудников. К этому сводятся все девять ценностей канбана:

1. **Прозрачность** – открытый обмен информацией;
2. **Баланс** – равновесие между нагрузкой и возможностями;
3. **Сотрудничество** – совместная работа участников команды и ее совершенствование;
4. **Фокус на заказчике и его потребностях** – создание продукта, который нужен клиенту;
5. **Поток** – непрерывная работа;
6. **Лидерство** – вдохновление своим примером других участников. При этом нет иерархии, понятие применимо на всех уровнях;
7. **Понимание** – знание всеми участниками целей развития команды;
8. **Согласие** – совместное движение к целям и совершенствованию;
9. **Уважение** – понимание и положительная оценка всех участников команды.

Если отступиться хотя бы от одной из ценностей, у команды ничего не получится – так считают создатели краткого руководства по канбану Дэвид Дж. Андерсон и Энди Кармайкл .

Принципы

Чтобы успешно использовать систему в своей команде, нужно придерживаться основных принципов канбана:

- **визуализировать работу** – разделить задачи на этапы;
- **систематизировать доску** – создать колонки, которые будут отражать текущий этап работы над задачей. Например: «надо сделать», «в работе», «сделано»;
- **актуализировать задачи** – постоянно обновлять статус, перемещая карточки из одной колонки в другую на доске, и выстраивать приоритеты в бэклоге;
- **контролировать течение задач** – если выполнение каких-то операций затягивается и карточка долго не продвигается по доске, важно проанализировать причины и при необходимости перераспределить ресурсы или помочь в решении;
- **постоянно совершенствовать систему** – визуализация помогает выявлять проблемные этапы и задачи. Процесс можно и нужно корректировать, устраняя уязвимые места.

Инструменты

Главный инструмент канбана – доска с карточками. Это может быть физическая меловая доска, магнитная, со стикерами или электронная. К ней должны иметь доступ все участники команды в любой момент времени.

Колонки доски:

- «Бэклог» – поле для всех карточек, пул задач, который может пополняться, сортироваться по приоритетности;
- «В процессе» – включает несколько видов внутренних колонок, адаптированных под команду и обозначающих разные этапы работы над карточкой;
- «Готово» – полностью выполненные задачи, которые не требуют от команды дальнейших действий.

На одной доске можно вест

3. Сетевая модель

Сетевая модель – это графически изображенная технологическая последовательность и логические взаимосвязи выполняемых работ. При создании сложных изделий возникает необходимость четкого координирования научно-исследовательских, конструкторских, технологических и производственных работ.

Для построения такой модели используют такие понятия, как работа и событие.

Работа – это любой процесс, который приводит к совершению события. На графике изображается вектором.

Событие – это момент окончания работ.

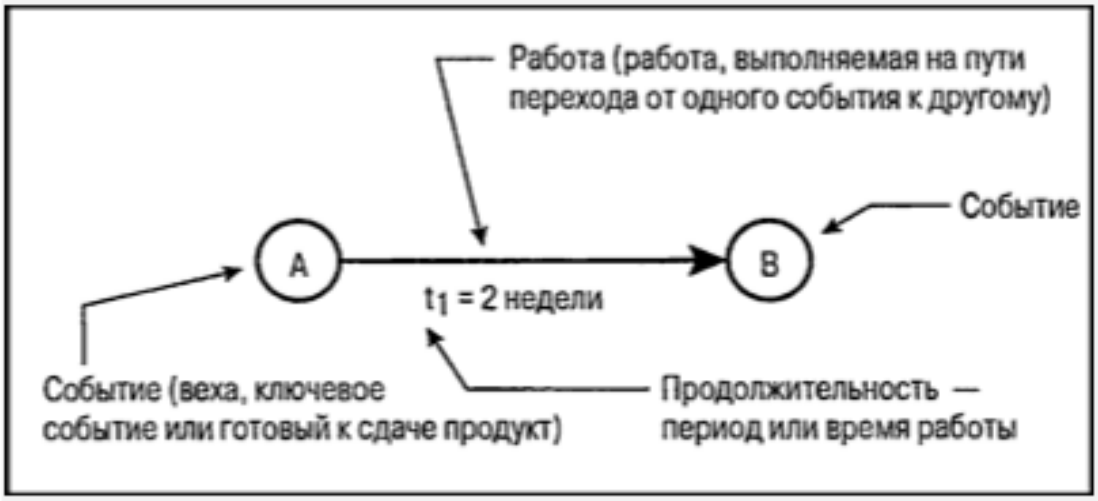


Рис 4.1. Сетевой график типа "события-работы "

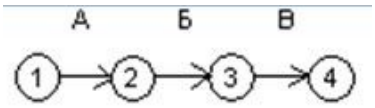
Разработка сетевой модели включает в себя 7 этапов:

1) составление перечня работ по объекту (изделию);
2) установление четкой последовательности и взаимосвязи работ;
3) построение сетевого графика;
4) определение продолжительности работ;
5) расчет параметров сетевой модели;
6) анализ модели оптимизации графика;
7) контроль функционирования сетевой модели.

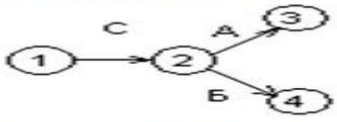
Правила построения сетевой модели:

вершины графа («кружки») отображают события(задача), а стрелки — работы(выполнение)

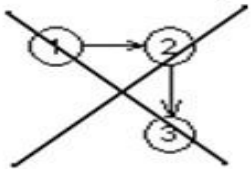
1. Если работы А, Б и В выполняются последовательно, то они изображаются следующим образом:



2. Если итогом выполнения работ А и Б является С,то она изображается так

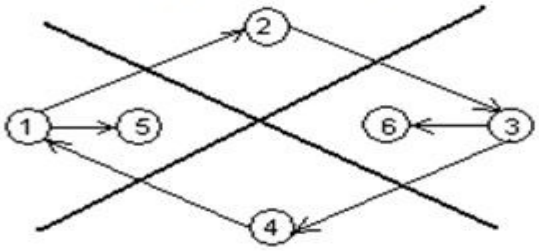


3. Номер начала события должен быть меньше конечного (2 → 4).
4. На графике не должно быть замкнутых контуров:



5. На сетевой модели не должно быть события, в которую не входит ни одна работа ⑤ и из которой не выходит ни одна работа ⑥.

Сетевое планирование и управление производством позволяет:



Пример логической взаимосвязи событий и работ можно представить следующим образом:

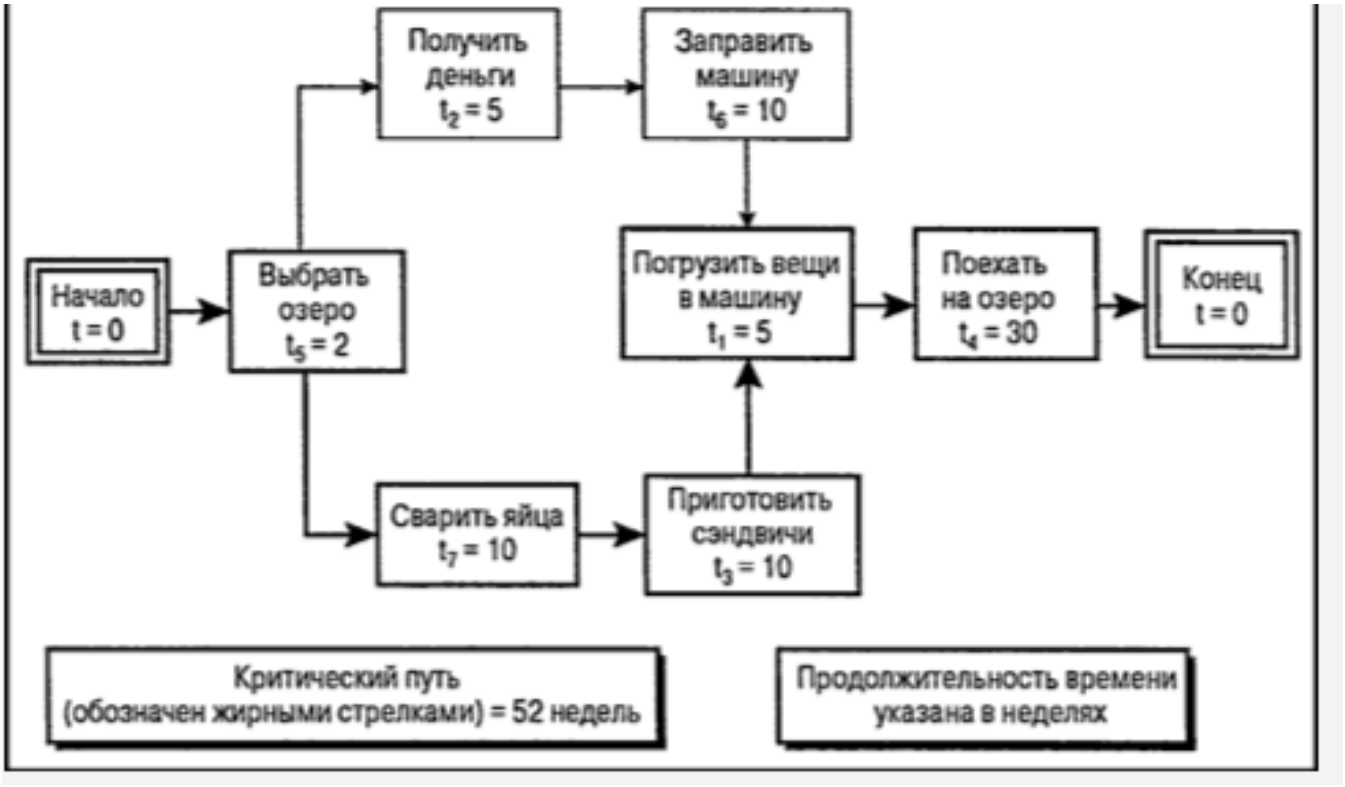


Рис. 4.7. **вид сетевого графика для организации пикника 1**

Теперь рассмотрим несколько важных вопросов.

Во-первых, сколько времени вам потребуется, чтобы собраться и добраться до озера?

• Верхний путь, включающий работы 2 и 6, — 15 минут.

• Нижний путь, включающий работы 7 и 3, составляет 20 минут.

• Самый длинный в графике — критический путь, он включает работы 5, 7, 3, 1 и 4. Его продолжительность — 57 минут.

Именно столько вам понадобится, чтобы добраться до озера, если следовать этому сетевому графику.

Можно ли задержать выполнение некоторых работ и все же уложиться в 57 минут?

Если да, то каких?

• Верхний путь, включающий работы 2 и 6, — не критический.

• Из сетевого графика следует, что поскольку работы 5, 7, 3, 1 и 4 находятся на критическом пути, они не могут быть задержаны ни в коем случае.

• Однако работы 2 и 6 можно выполнять одновременно с работами 7 и 3.

Работы 7 и 3 занимают 20 минут, в то время как работы 2 и 6 — 15 минут.

Поэтому работы 2 и 6 имеют резерв времени в 5 минут.

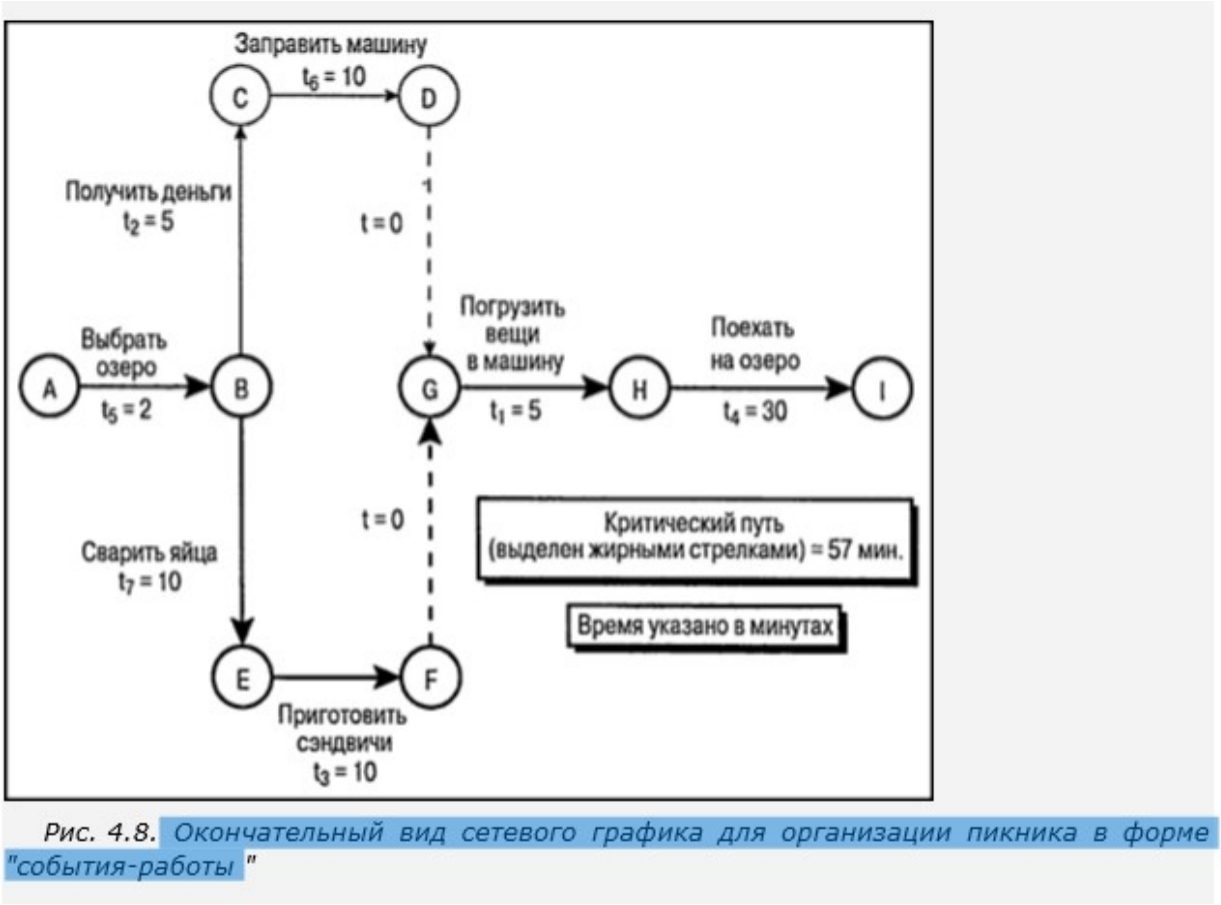


Рис. 4.8. Окончательный вид сетевого графика для организации пикника в форме "события-работы"

Стрелки пунктиром показывают, что в расчет включен не критический путь и что этот резерв времени в критическом пути израсходуетс.

Из сетевого графика вы можете извлечь следующую информацию, которая позволит вам продумать возможный график работ.

• **Критический путь**. Последовательность работ в проекте, которая требует больше всего времени для завершения.

• **Некритический путь**. Последовательность работ, которую можно выполнить с некоторой задержкой, что не помешает завершить весь проект в кратчайший срок.

• **Резерв времени**. Максимальное время, на которое можно задержать определенные работы и при этом закончить проект в кратчайший срок.

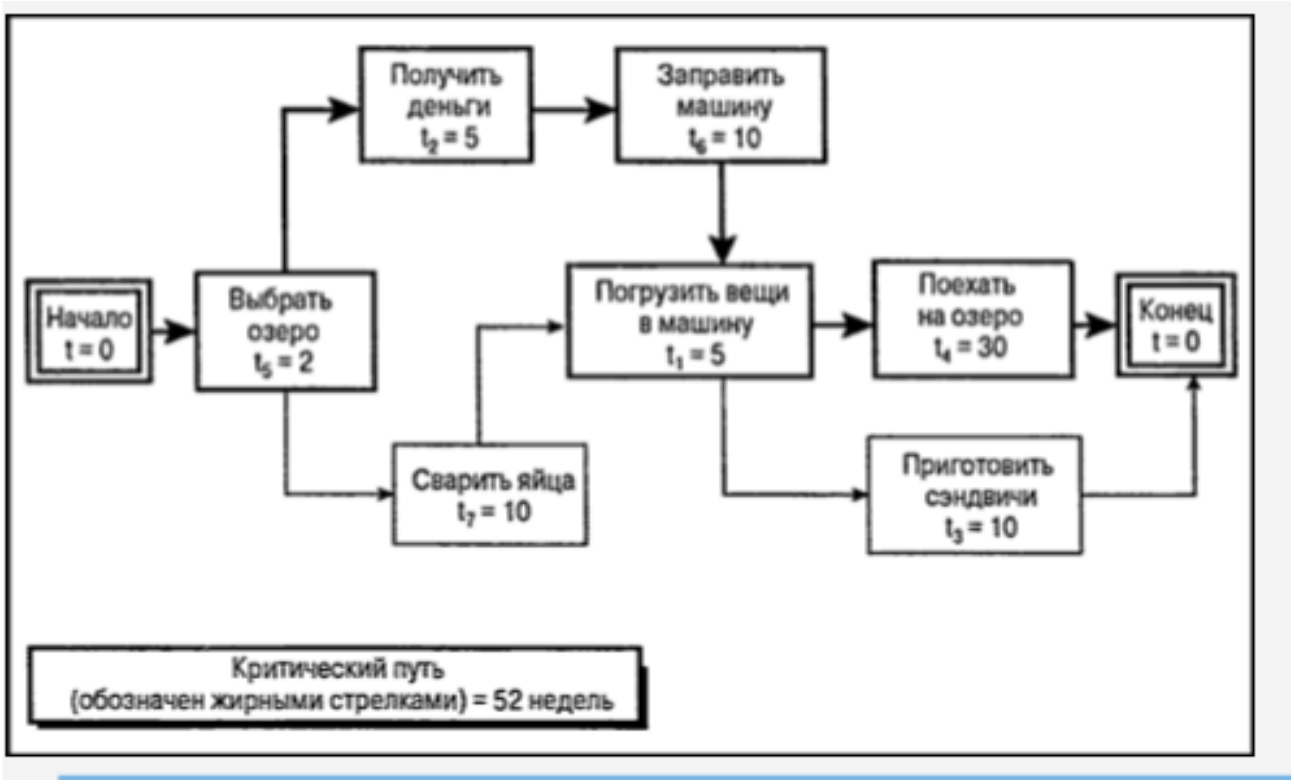
• **Самый ранний срок начала**. Наиболее ранний календарный срок, когда можно начать работу.

• **Самый ранний срок окончания**. Наиболее ранний календарный срок, когда можно закончить работу.

• **Самый поздний срок начала**. Наиболее поздний календарный срок, когда можно начать работу и при этом завершить проект в кратчайший срок.

• **Самый поздний срок окончания**. Наиболее поздний календарный срок, когда можно закончить работу и при этом завершить проект в кратчайший срок.

Еще варианты продуманный для сокращения времени:

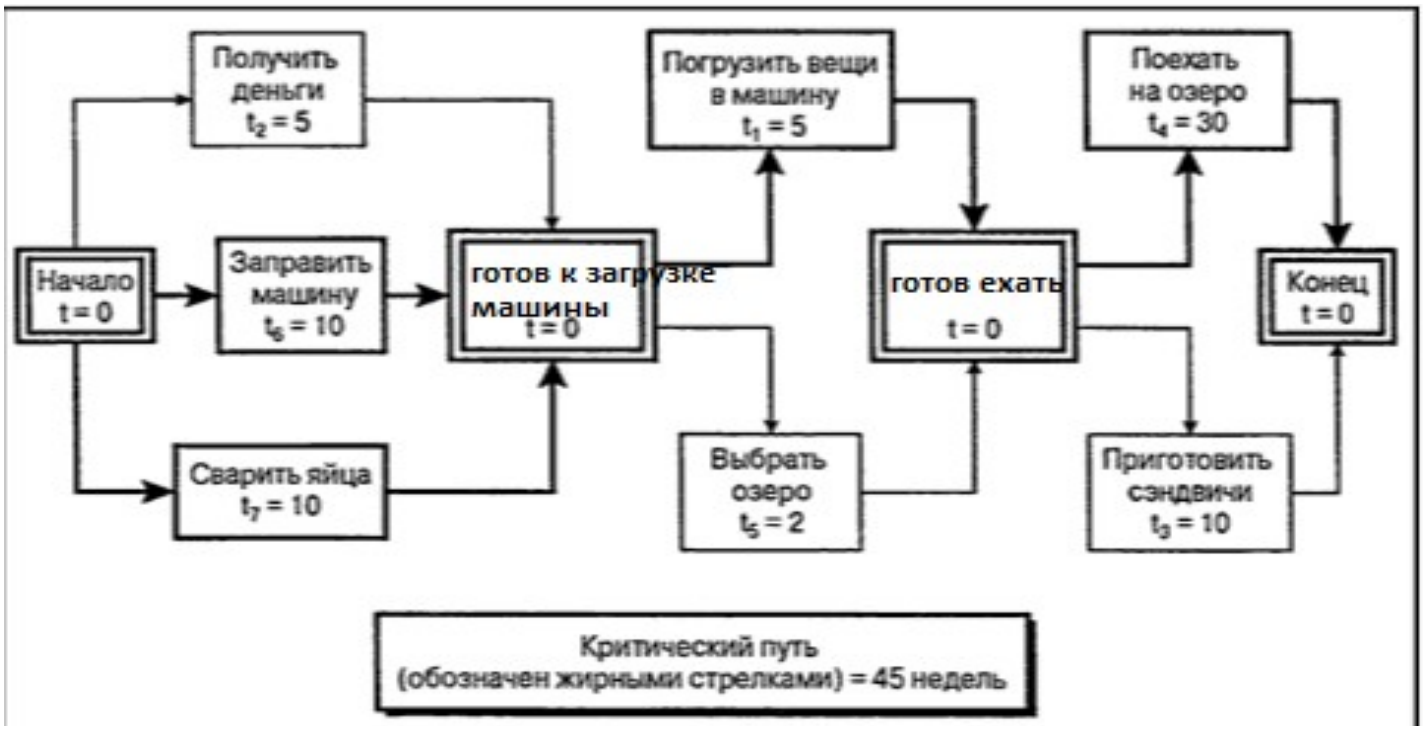


Изменения в сетевом графике: **приготовление сэндвичей в машине по пути к озеру 2**

Верхний путь, включающий работы 2 и 6, занимает 15 минут, а нижний, включающий работы 7 и 3, — 20.

Поскольку нижний путь — критический, экономия на нем 5 минут сократит время проекта на 5 минут. С этой точки зрения, у нас теперь два критических пути — каждый по 15 минут.

Если сэкономить еще 5 минут на нижнем пути, то это не отразится на проекте в целом, так как верхний путь все еще занимает 15 минут. Но зато у нас будет резерв времени в 5 минут на нижнем пути.



Время поездки на пикник сокращено до 45 минут

Вернемся снова к первоначальной идее получить деньги в банкомате, пока заправляется машина. Теперь это сэкономит нам еще 5 минут, поскольку верхний путь стал критическим. Наконец, вы можете решить, к какому озеру ехать, во время погрузки вещей в машину. Это сэкономит вам еще две минуты. Таким образом, общее время проекта сократится до 45 минут, что отражено на рис. 4.10.

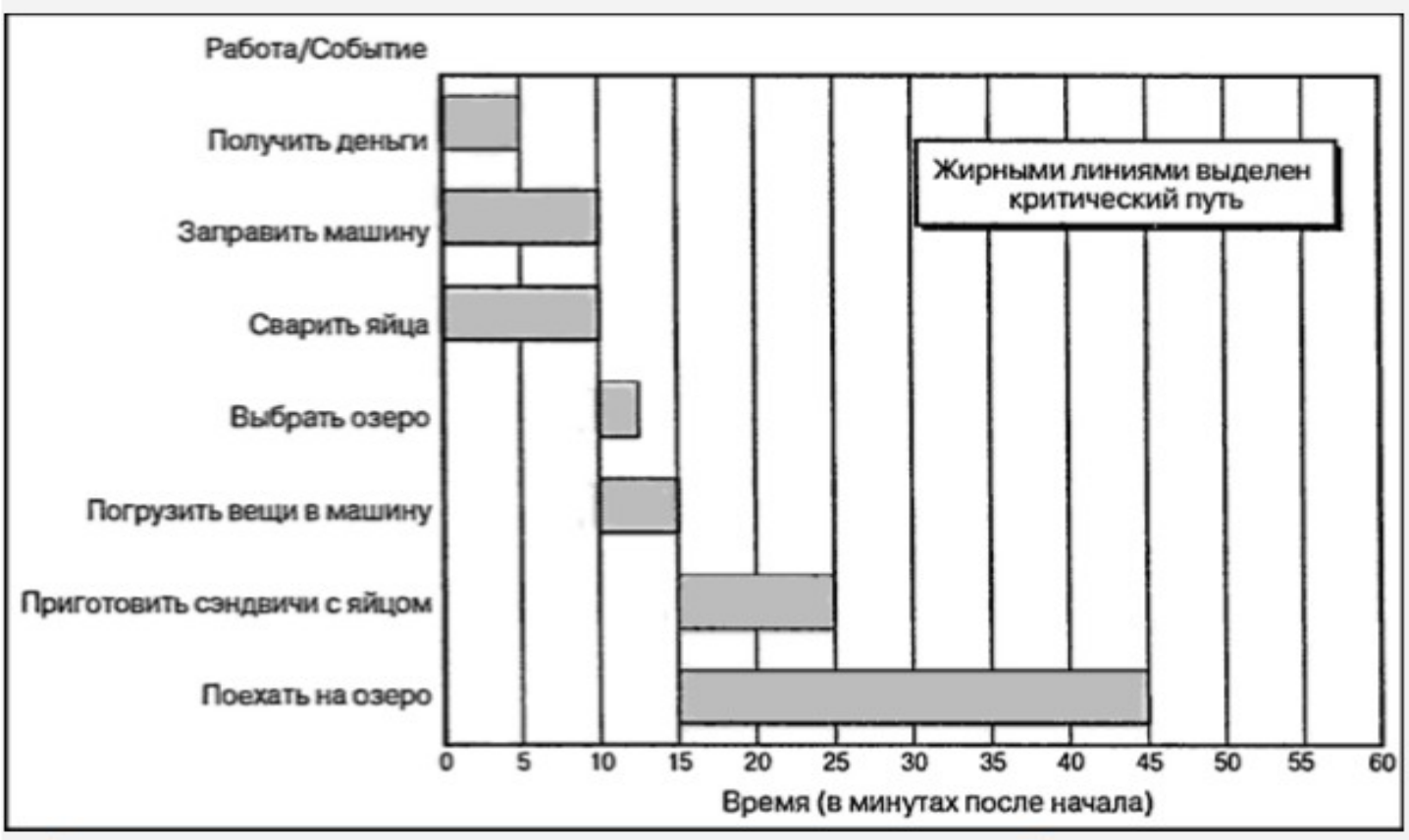


Рис. 4.14. График Гантта для организации пикника на озере

4.Объекта календарный график

Объектный календарный график-определяет очередность и сроки выполнения каждого вида работ на конкретном объекте с начала его возведения до сдачи в эксплуатацию. Обычно такой план имеет разбивку по месяцам или дням в зависимости от величины и сложности объекта. Не востребован.

Контрольные вопросы:

26. Что такое календарное планирование
27. Какие даты установки завершения проекта вы знаете
28. Что такое рабочий график
29. Назовите методы календарного планирования
30. Опишите метод диаграммы Ганта
31. Что такое Канбан, опишите его
32. Что такое сетевая модель принцип работы
33. Что такое объекта календарная модель

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами
5. https://habr.com/post/189626/
6. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
7. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

5. https://habr.com/post/189626/
6. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
7. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 11

Тема: Структурная декомпозиция. План-график и моделирование проекта. Пакеты работ

Цель: изучить основные методы проектирования.

В практике проектирования наиболее часто используются такие методы:

31. мозговой штурм,
32. экспертная оценка,
33. метод аналогий,
34. календарное планирование,
35. структурная декомпозиция,
36. имитационное моделирование,

Структурная декомпозиция работ (СДР)

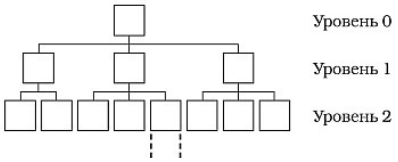
структурная декомпозиция (дерево) задач управления работами проекта имеет 3 основные уровня детализации , как дерево решений

Структурная декомпозиция работ (СДР) — это описание работы, которая будет сделана по проекту. Это иерархия задач, которая представляет понимание проектной группы композиции работы, а также размера, стоимости и продолжительности каждого компонента или задачи.

Это структурированная иерархия процессов с оценкой их стоимости и детализации самих процессов.

СДР имеет три главные цели: Описание декомпозиции или композиции работы в задачах.

1. Планирование работы проекта.
2. Оценка стоимости каждой задачи.
3. Степень детализации в СДР зависит от уровня точности, который необходим в оценках, и уровня отслеживать отслеживания, необходимого для этих оценок.



Определение степени детализации СДР

Выяснение степени детализации СДР включает в себя определение количества уровней СДР, количества и среднего размера пакетов работ, подходящих к конкретной ситуации и принятых в вашей отрасли.

Детализация пригодная для большинства малых и средних проектов в сферах информационных технологий, разработки программного обеспечения и продуктов:

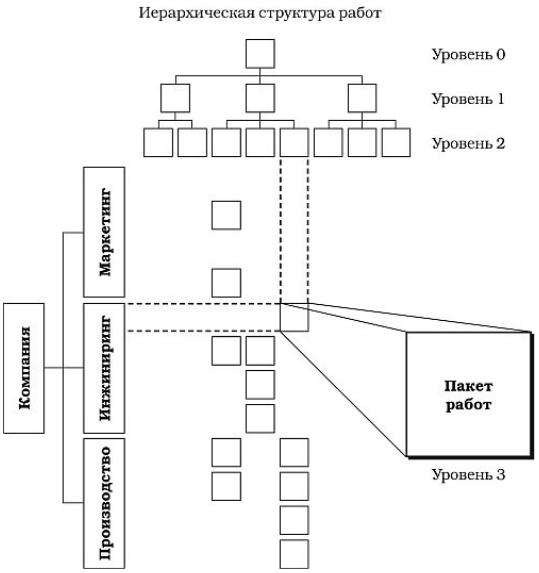
1. от трех до четырех уровней в СДР;
2. от 15 до 40 пакетов работ
3. от 40 до 80 часов на средний пакетработ

длительность среднего пакета работ – от одной до двух недель;

4. от 3 до 7% общего бюджета рабочих часов на средний пакет работ.

Если количество уровней создалось слишком много, это вносит беспоряд в проект и лишние материальные расходы!

Пакет работ – ключевое звено в управлении иерархической структурой работ



Две группы методов: моделирующие функциональную структуру и структуру данных
Наибольшее распространение получили методологии:

- IDEF0 – функциональные модели, основанные на методе SADT;
- IDEF1X – диаграммы данных «сущность-связь» (ERD);
- IDEF3 — диаграммы потоков работ (Work Flow Diagrams);
- DFD — диаграммы потоков данных (Data Flow Diagrams)

IDEF0-модель состоит из диаграмм и фрагментов текста. На диаграммах все функции системы и их взаимодействия представлены как блоки (функции) и дуги (отношения).

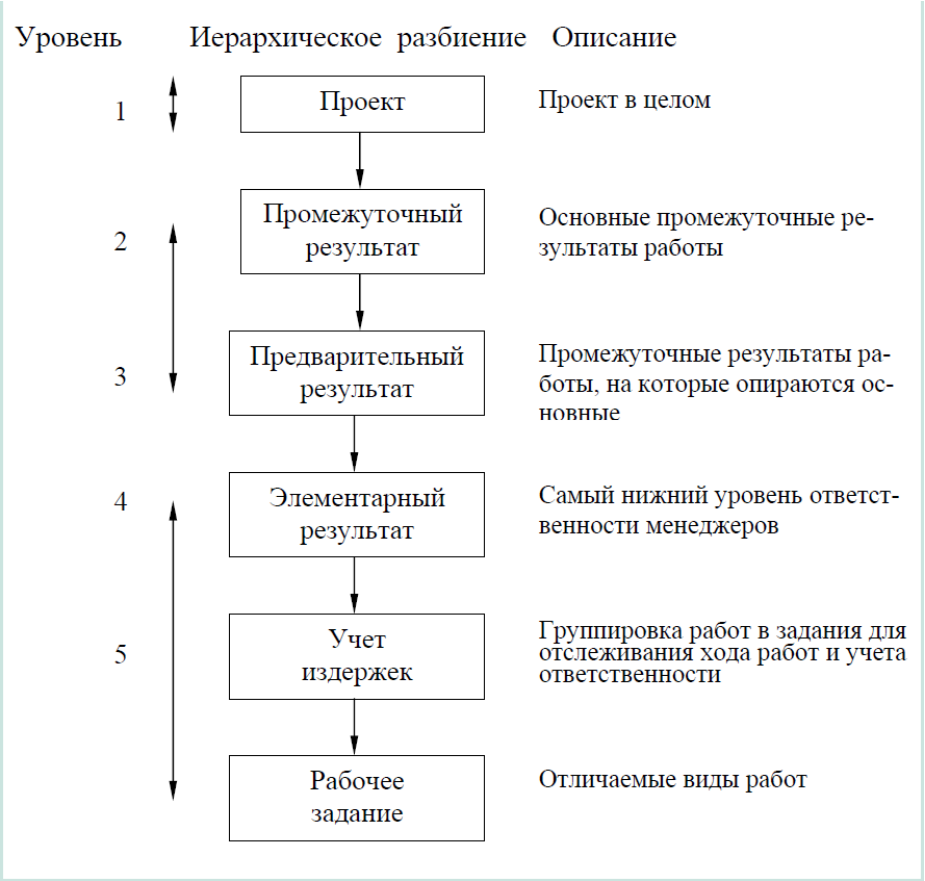
Преимущества использования СДР :

1. •***эффективная визуализация***.
2. ***простота***. обычно для того, чтобы участники проекта смогли читать и строить СДР, достаточно весьма небольшой подготовки.

Недостатки:

1. •*чрезмерно большая СДР требует слишком много времени, что сводит «на нет» производительность.*
2. Если СДР состоит из слишком *большого количества уровней* и пакетов работ, то ее использование в качестве каркаса для интеграции функций планирования и контроля *проекта становится бессмысленным*, времяемким и требующим больших затрат ресурсов.

В результате мы получаем наш проект в таком виде



Контрольные вопросы:

34. Что такое календарное планирование
35. Какие даты установки завершения проекта вы знаете
36. Что такое рабочий график
37. Назовите методы календарного планирования
38. Опишите метод диаграммы Ганта
39. Что такое Канбан, опишите его

40. Что такое сетевая модель принцип работы
41. Что такое объекта календарная модель

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами
8. https://habr.com/post/189626/
9. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
10. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

Лекция 12-14

Темы:

12 Основы проектирования ПС.

13 Роли и участники проекта.

14 Ключевые вопросы проектирования .Параллелизм. Асинхронные агенты.

Проектирование программной системы. Этот процесс можно определить как процесс создания *проекта* программной системы (ПС) — набора схем, диаграмм, технических заданий и другой документации, содержащих описание разрабатываемого ПП в объеме, достаточном для его конструирования. Проект необходим для того, чтобы все его участники понимали цель разработки, какой продукт и с какими характеристиками будет создан в результате их деятельности.

Целью проектирования является определение внутренних свойств системы и детализации ее внешних (видимых) свойств на основе выданных заказчиком требований к ПС (исходных условий задачи).

Процесс проектирования — инженерная деятельность в рамках ЖЦ ПП, в рамках которой анализируются требования и создается описание внутренней структуры ПС, которое является основой для его дальнейшего конструирования.

В зависимости от сложности создаваемого ПП процесс проектирования может обеспечиваться как «ручным» проектированием, так и различными средствами его автоматизации.

В процессе проектирования ПС также могут использоваться различные графические средства: блок-схемы, ER-диаграммы, UML-диаграммы, а также *макеты.*

Блок-схемы — распространенный тип схем (графических моделей), описывающих алгоритмы или процессы, в которых отдельные шаги изображаются в виде блоков различной формы, соединенных между собой линиями, указывающими направление последовательности их выполнения.

ER-диаграммы используются при высокоуровневом проектировании БД.

UML-диаграммы (UML, Unified Modeling Language — унифицированный язык моделирования) — графическое описание процесса моделирования в области разработки ПО.

Проектированию обычно подлежат архитектура ПО, устройство компонентов ПО, пользовательские интерфейсы.

Первоначально программа рассматривается как черный ящик. Ход процесса проектирования и его результаты зависят не только от состава требований, но и от выбранной модели процесса, опыта проектировщика.

Проектирование является сложным и, как правило, трудоемким процессом, требующим большого опыта и знаний, которые позволяют находить компромиссные решения при разработке требуемого программного обеспечения.

Роли участников процесса проектирования. При проектировании должны быть задействованы определенные люди, которые примут участие в анализе и разработке программного проекта.

Участникам проекта принято назначать роли — функции, которые они будут выполнять в проекте. В зависимости от квалификации и величины проектной команды один человек может совмещать различные роли (например, если программу пишет один разработчик, то он может совместить их все). Для осуществления процесса проектирования выделяют следующие основные роли:

1. заказчик;
2. руководитель проекта;
3. системный администратор;
4. администратор базы данных;
5. системный архитектор;
6. архитектор базы данных;
7. бизнес-аналитик;
8. аналитик (системный аналитик);
9. тестировщик.

Заказчиком является лицо, которому нужно разрабатываемое ПО. В качестве заказчика может выступать один человек, группа людей либо какое-либо предприятие, юридическое лицо. Заказчик определяет требования к разрабатываемому программному продукту, которые влияют на то, каким будет конечный результат разработки.

Руководитель проекта управляет процессом разработки ПП. Он отвечает за выполнение всех задач в срок перед заказчиком, обеспечивает взаимодействие всех участников программного проекта, контролирует процесс выполнения.

Системный администратор создает условия для непрерывной работы над разрабатываемым программным продуктом, следит за работоспособностью серверов, выявляет и устраняет неполадки. При проектировании он выступает в качестве консультанта по требованиям к аппаратному обеспечению (серверам, рабочим компьютерам, мощности процессоров, необходимых для работы, объему оперативной памяти и т.д.), необходимому для нормального функционирования программного продукта.

Администратор базы данных обеспечивает непрерывность работы сервера базы данных, контролирует его работу. При проектировании выступает в качестве консультанта по требованию к серверу базы данных, а также к системе управления базой данных, необходимой для программного продукта.

Системный архитектор проектирует архитектуру всего программного продукта в целом и более детально производит проектирование самого приложения, не вдаваясь в подробности проектирования базы данных.

Архитектор базы данных проектирует укрупненную структуру базы данных. Более детальная проработка архитектуры базы данных производится на этапе конструирования ПО.

Бизнес-аналитик описывает требуемое поведение системы с точки зрения конечных пользователей. Например, с точки зрения пользователя, покупка выбранного товара будет заключаться в щелчке левой кнопкой мыши по кнопке «Купить» и заполнении данных банковской карты. С точки зрения разработчика, данное действие будет заключаться в получении события щелчка левой кнопкой мыши по кнопке «ЫпВиу», вызове формы «бгтВиу» и проверке заполненных полей.

Аналитик (системный аналитик) пишет технические задания для разработчиков. В зависимости от проекта технические задания могут быть различной степени детализации — от подробных, по которым разработчикам остается только закодировать написанное ТЗ на выбранном языке программирования, до общих, которые требуют дополнительного проектирования (проработки структуры таблиц, классов, методов и пр.).

Тестировщик производит тестирование ПП. При проектировании его необходимо ознакомить с подготовленной документацией по программному продукту для подготовки к процессу тестирования. В настоящее время существуют определенные технологии разработки программных проектов, которые основываются на тестах. При проектировании пишутся тесты для разрабатываемого функционала, а при разработке систему конструируют в соответствии с написанными тестами.

Для каждого программного продукта набор ролей будет различным. Например, при разработке интернет-магазина могут потребоваться все перечисленные роли, а при разработке мобильного приложения может быть достаточно заказчика, руководителя проекта, системного архитектора и системного аналитика.

Ключевые вопросы проектирования

На этапе проектирования программного продукта необходимо определиться с ключевыми вопросами, которые оказывают существенное влияние на архитектуру программного продукта.

Параллелизм. *Параллельные вычисления* — способ организации компьютерных вычислений, при котором программы разрабатываются как набор взаимодействующих вычислительных процессов, работающих параллельно (воспринимаемых пользователем как одновременные).

Существуют различные способы реализации параллельных вычислений.

Например, каждый вычислительный процесс может быть реализован в виде процесса операционной системы (ОС) либо вычислительные процессы могут представлять собой набор потоков выполнения внутри одного процесса ОС.

Параллельные программы могут физически исполняться либо последовательно на единственном процессоре, осуществляя по очереди шаги выполнения каждого вычислительного процесса, либо параллельно, выделяя каждому вычислительному процессу один или несколько процессоров (находящихся рядом или объединенных в компьютерную сеть).

Основной сложностью при проектировании параллельных программ является 1. обеспечение правильной последовательности взаимодействий между различными вычислительными процессами, а также координация ресурсов, разделяемых между процессами.

Например, для интернет-магазина по продаже сотовых телефонов можно выделить следующие параллельные процессы:

1) формирование и выдача страниц по запросам пользователей (обеспечивается на уровне архитектуры Web-сервера);

2) выдача запросов базой данных (обеспечивается на уровне архитектуры базы данных).

2. ряд действий нельзя выполнять параллельно.

Например, при регистрации, когда резервируется уникальный код доступа к учетной записи, нельзя одновременно давать регистрироваться двум пользователям, так как это может вызвать конфликт кодов. Функции обработки заказов также нельзя делать параллельно, так как товар на складе может закончиться при обработке одного из заказов.

Асинхронные агенты. «Тяжелые» (продолжительные) задачи необходимо выделять в отдельные потоки, что позволяет разгрузить потоки, обрабатывающие сообщения от графического интерфейса. Например, это позволяет отделить прорисовку графической информации от вычислений, занимающих большой объем времени. К категории задач, которые необходимо выделять на обработку в отдельные потоки, можно отнести следующие: печать, лингвистические проверки, КОМПИЛЯЦИЮ и т.д.

Основным принципом для взаимодействия данных процессов является асинхронный обмен сообщениями. В случае с графическим интерфейсом данный подход является наиболее предпочтительным, так как работа графического интерфейса основана на сообщениях.

Для WEB-приложений выделить асинхронные события несколько трудней, так как обработка графической информации производится в браузерах клиентов.

Но для них также возможно создание асинхронных событий на основе использования специальных возможностей встроенного языка разметки.

Например, для интернет-магазина сотовых телефонов в качестве асинхронного события можно выделить автоматическое обновление статуса заказов на страницах без их перерисовки.

Контроль и обработка событий. *Событийно-ориентированная архитектура* позволяет реализовать механизм работы приложения не на основе прямого вызова методов классов, а на основе реагирования на те или иные события. Таким образом можно уменьшить связность в системе.

Изначально событийность была присуща графическому интерфейсу, где вся работа основывалась на передаче сообщений. Событийность возможна как асинхронное решение для обмена информацией в многослойных приложениях (когда архитектура делится на несколько слоев), особенно это касается приложений, слои которых

разделены по разным серверам, так как в этом случае прямой вызов методов невозможен.

Обеспечение отказоустойчивости. Одной из задач проектирования является обеспечение корректной обработки ошибок. В частности, при проектировании необходимо описать, как обеспечить корректную работу системы, даже если произошел сбой.

Существует два наиболее распространенных метода контроля ошибок.

1. *Исключения* — метод, позволяющий разработчикам получить наиболее подробную информацию по ошибкам и достаточно легко их локализовать, классифицировать и обрабатывать централизованно. Недостатком этого способа является невысокая скорость об-

работки ошибок, поскольку, чтобы собрать необходимые для исключения данные, необходимо проделать значительный объем вычислений. Поэтому данный метод не рекомендуется использовать при обработке больших потоков информации, когда выдвигаются требования к скорости.

2. *Коды ошибок* — метод, широко использующийся при обработке большого потока данных, когда приоритетом ставится скорость обработки информации. При появлении ошибки функции возвращают лишь ее код, «надеясь» на то, что в вызывающих процедурах есть логика по их обработке. Это менее надежный, но более производительный метод.

Не все языки программирования поддерживают обе возможности работы с ошибками. Например, часть языков баз данных имеют возможность работы только с кодами ошибок.

Лекция 15-18

Темы:

15Модели анализа и определения спецификаций. Компоненты полной спецификации методологии структурного анализа.

16 Диаграммы переходных состояний SDT

17 Диаграммы переходных состояний SDT. Разбор ошибок в построении схем.

18Функциональные диаграммы.

Все функциональные спецификации разрабатываемого программного обеспечения описывают перечень функций и состав обрабатываемых данных. Они различаются только системой приоритетов (акцентов), которая используется разработчиком в процессе анализа требований и определения спецификаций.

Так, диаграммы переходов состояний определяют некоторые аспекты поведения программного обеспечения во времени, диаграммы потоков данных — направление и структуру потоков данных, а концептуальные диаграммы классов — отношение между основ ными понятиями предметной области.

На рис. 2.1 показана классификация моделей, используемых в качестве спецификаций разрабатываемого программного обеспечения. **В рамках структурного подхода на этапе анализа и определения спецификаций используют три типа моделей**:

1.ориентированные на функции,

2.ориентированные на данные

3. ориентированные на потоки данных.

Каждый тип модели целесообразно использовать для своего специфического класса программных разработок. Разные модели описывают проектируемое программное обеспечение с разных сторон.

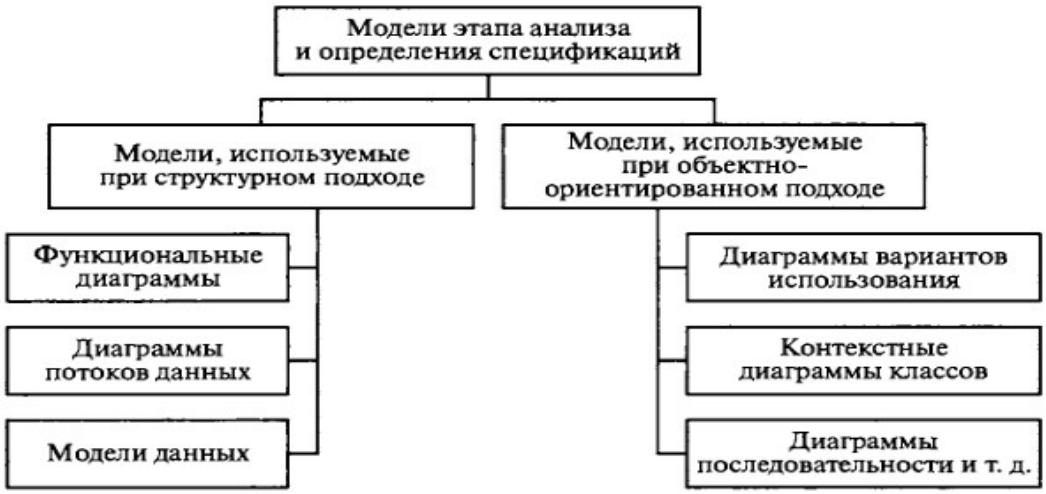


Рис. 2.1. Классификация моделей программного обеспечения, используемых на этапе определения спецификаций

19

Методологии структурного анализа и проектирования, основанные на моделировании потоков данных, обычно используют комплексное представление проектируемого программного обеспечения в виде совокупности следующих моделей:

■ диаграммы переходов состояний (SDT — State Transition Diagrams), характеризующие поведение системы во вре мени;

■ функциональные диаграммы (SADT — Structured Analisis and Design Technique);

■ диаграммы потоков данных (DFD — Data Flow Diagrams), описывающие взаимодействие источников и потребите лей информации через процессы, которые должны быть реализованы в системе;

■ диаграммы «сущность—связь» (ERD — Entity—Relation ship Diagrams), описывающие базы данных разрабатыва емой системы.

Компоненты спецификации методологий структурного анализа приведены на рис. 2.2. Спецификацию процесса обычно представляют в виде кратко го текстового описания, схем алгоритмов, псевдокодов. Словарь данных — это краткое описание основных понятий, используемых при составлении спецификаций.

Часть 16

диаграммы переходов состояний SDT

State & Transition Diagram (сокращенно S&T) — схема состояний и переходов. Техника для визуализации ТЗ. Она наглядно показывает, как некий объект переходит из одного состояния в другое.

Вот объект находился в состоянии А, потом произошло какое-то действие, и он попал в состояние В. Потом он попадет в состояние С и другие... Принцип не меняется, было одно состояние, стало другое.

Диаграммы переходов состояний (*STD*) предназначены для моделирования и документирования аспектов систем, зависящих от времени или реакции на событие. Они позволяют осуществлять декомпозицию управляющих процессов и описывают отношения между входными и выходными управляющими потоками для управляющего процесса-предка. С помощью STD-диаграмм можно моделировать последующее функционирование системы на основе ее предыдущего и текущего функционирования. **Моделируемая система в любой заданный момент времени находится точно в одном из конечного множества состояний**.

Объекты std

Состояние - условие устойчивости для системы: способность системы сохранять свои функции без их произвольного изменения. *Имя состояния* - отражает реальную ситуацию, в которой находится система.

Цель - определяет будущее состояние по прошлому и текущему.

Переход - определяет перемещение системы из одного состояния в другое. *Имя перехода* - идентифицирует события, которые являются причиной перехода.

Событие - состоит из какого-либо управляющего потока (внешнего, внутреннего) и происходит при выполнении некоторого условия. При этом следует отметить:

1. Не все события вызывают переходы.
2. События не всегда вызывают переходы.
3. События не всегда вызывают переход в одно и то же состояние.

Условие - событие, вызывающее переход и названое именем перехода.

С переходом из одного состояния в другое может связываться действие или совокупность действий.

Действие при переходе - операция, которая выполняется при переходе. Действие может быть или физическим, или управляющим потоком.

Пример STD диаграммы для работы банкомата приведен на рисунке 5.

На диаграмме элементы нотации обозначаются следующим образом:

- ***состояния*** – как узлы (например**, Ожидание)**;

- ***переходы*** – как дуги (например**, Корректный пароль**);

- ***условия***- идентифицируются именем перехода (например,**Корректный пароль**);

- ***действия*** - отклики на события, которые "привязываются" к переходам, записываются под условием (например,**Обеспечить требуемый сервис**).

STD имеет только одно начальное состояние. Но система может иметь большое количество завершающих состояний.

Рисунок 5 - Диаграмма переходов состояний (STD) для банкомата

Рекомендации. При построении STD желательно выполнять правила:

- строить диаграмму на наиболее высоком уровне;
- детализировать (обеспечивать иерархичность диаграмм);
- использовать те же термины (имена событий, действий, потоков,

Часть 17

Состояние и переход

Мы рисуем:

- кружочки — состояния объекта;
- стрелочки — то, благодаря чему из состояния A в состояние B.

  Это действие, но его может совершить не только пользователь, но и система сама. Например, задача запустилась автоматически в 10 часов вечера.

Такая схема позволяет нам сразу визуально оценить, какие переходы вообще возможны и что надо протестировать. Ведь нам надо протестировать и эту стрелку, и эту... Так что стрелочки — это наши готовые тест-кейсы!

Схема состояний и переходов относится к техникам тест-дизайна. Значит, про неё спрашивают на собеседованиях.

Как рисовать диаграмму

Очень важно: S&T рисуется на объект! Один объект. В идеале — на объект, имеющий аналог в базе данных продукта.

Шаг 1. Вы выбираете объект в своём проекте (рабочем или учебном, не суть).

Шаг 2. Думаете, какие у него состояния. Они не должны пересекаться, то есть: объект не может быть разом в двух состояниях, и при этом он всегда хоть в каком-то одном есть

Шаг 3. Рисуете эти состояния кружочками.

Шаг 4. Соединяете их стрелочками. Стрелочки - это действия, их надо подписать.

Шаг 5. Смотрите, что получилось и анализируете, есть ли у объекта другие состояния? А другие действия между текущими состояниями? Переход на шаг 2.

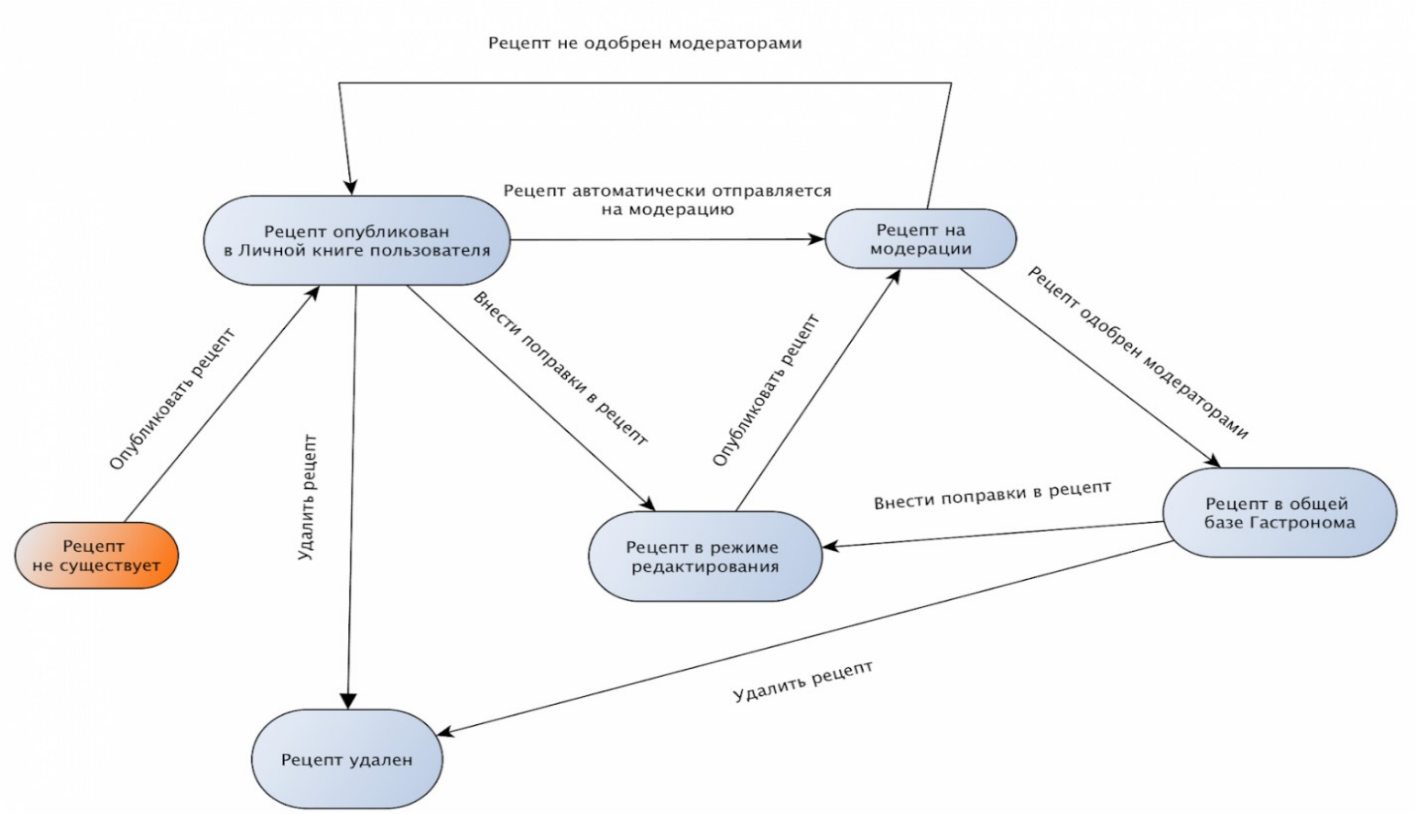
Чтобы начать, задайте себе вопросы:

1. Какой конкретно объект вы выбрали? Как он называется? *(только один!)*
2. Какие у этого объекта есть состояния?

Основное определение состояния — "набор доступных и недоступных действий с объектом". Продукт всегда должен знать, в каком состоянии каждый его объект. Вообще, когда будете думать об объектах и состояниях, старайтесь представлять их аппаратную реализацию.

Объект — это практически всегда строка в базе данных, старайтесь абстрагироваться от интерфейса вообще, и представляйте те действия, которые вы могли бы делать с объектом прямыми запросами в базу.

Вот пример хорошей диаграммы:



Типовые ошибки при составлении карты

1. Вместо объекта — GUI

Очень важно: S&T рисуется на объект! Это не зарисовка графического интерфейса «открыта страница такая, открыта страница сякая»... Если вы описываете разные странички GUI — это уже не S&T.

Зарисовывать страницы смысла обычно нет. Это как при рисовании майнд-карты — мы не рисуем графический интерфейс, мы описываем функционал. То, зачем пользователь вообще пришел на сайт. Это намного полезнее!

Другой вариант той же ошибки: искать билет — (результаты поиска) — открыть форму покупки — (форма открыта) — ввести данные кредитной карты — (данные введены).

2. Несколько объектов в одной карте

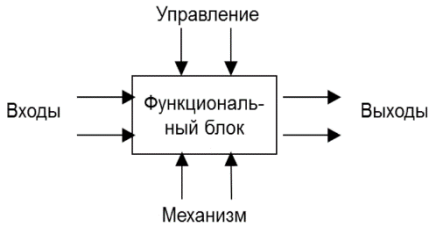
На прошлой картинке у нас несколько объектов: результаты поиска, форма, данные. И все — плохие. **Потому что там мы явно что-то покупаем, вот это «что-то» и есть объект!**

Но когда мы описываем покупку, тоже легко скатиться в несколько объектов в одной карте: "пицца в корзине", "заказ оформлен".

Товар тоже очень часто путают, потому что есть два варианта:

1. как на авито — продается конкретная вещь: "Нет на сайте", "Продается", "Продан".
2. просто "товарная позиция", как какие-

нибудь носки в магазине одежды: "Отсутствует", "В наличии", "Ожидается поступление" и так далее.



Если речь о сайте типа авито, то объект лучше выбрать "объявление", будет логичнее. А вот если мы покупаем пиццу — это будет товарной позицией.

3. Несколько одинаковых состояний

Вспомните пример с тортиком:

1. Купила.
2. Поставила в холодильник на потом.
3. Передумала, достала, надкусала.
4. Снова передумала, решила съесть целиком, осилила половину.
5. Расстроилась, решила не доедать вообще и выкинуть.

Половину пунктов можно объединить. Ведь состояние торта не меняется от того, купили вы его только что или час назад, сидите любуетесь на него, переставляете с места на место или убираете в холодильник:

- 1-2 это торт куплен;
- 3-4 в процессе уничтожения;
- 5 выброшен.

Часть 18

Результатом применения методологии SADT является модель, которая состоит из диаграмм, фрагментов текстов и глоссария, имеющих ссылки друг на друга.

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

Основные элементы модели SADT**:**

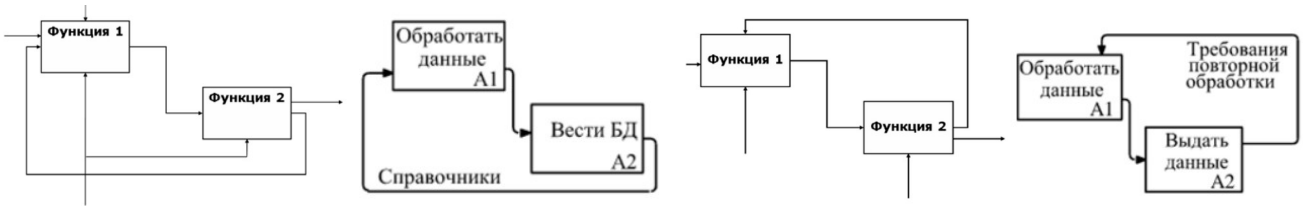
1. **входа** (входят в левую грань работы) - изображают данные или объекты, изменяемые в ходе выполнения работы;
2. **управления** (входят в верхнюю грань работы) - изображают правила и ограничения, согласно которым выполняется работа;
3. **выхода** (выходят из правой грани работы) - изображают данные или объекты, появляющиеся в результате выполнения работы;

4. **механизма** (входят в нижнюю грань работы) - изображают ресурсы, необходимые для выполнения работы, но не изменяющиеся в процессе работы (например, оборудование, людские ресурсы и т.д.);
5. **вызова** (выходят из нижней грани работы) - изображают связи между разными диаграммами или моделями, указывая на некоторую диаграмму, где данная работа рассмотрена более подробно.

Таким образом, **IDEF0-модель состоит из** набора иерархически связанных диаграмм

Типы связей между блоками:

1.Выход-вход 2.**вход-управление** **3.выход-механизм**



5.обртная связь по выходу_ **4.обратная связь по упрвлению**

1.Отношение входа возникает тогда, когда выход одного блока становится входом для одного из последующих блоков. Такие отношения являются наиболее часто встречающимися связями.

2.Отношения управления — когда выход одного блока непосредственно влияет на работу какого-либо последующего блока.

3.Связь "выход — механизм" встречается нечасто и отражает ситуацию, при которой выход одного блока становится средством достижения цели для другого блока. Данная связь характерна при распределении источников ресурсов: физического пространства, оборудования, финансирования, материалов, инструментов, обученного персонала

4,5.Обратные связи по управлению и по входу предназначены для представления итерации или рекурсии.

4.**Обратная связь по управлению** возникает тогда, когда выход некоторого блока влияет на работу какого-либо предыдущего блока.

5.**Обратная связь по входу** применяется тогда, когда нужно показать, что выход одного блока становится входом для какого- либо из предшествующих блоков, например в случае возврата проекта документа на доработку.

Название связи	**Вид связи**	**Смысл связи**
Связь предшествования		Обозначает, что вторая работа начинает выполняться после завершения первой работы.
Связь отношения		Обозначает, что вторая работа может начаться и даже закончиться до того момента, когда закончится выполнение первой работы.
Связь потоков объектов		Одновременно обозначает временную последовательность работ и материальный либо информационный поток. В данном случае вторая работа начинает выполняться после завершения первой работы. При этом выходом первой работы объект название которого надписано над стрелкой (в данном случае документ). Эта связь также обозначает, что объект порождаемый первой работой, используется в последующих работах.

*стрелка с двумя наконечниками показывает, что работы будет продолжаться и в других процессах /схемах, чаще всего горизонтально используется

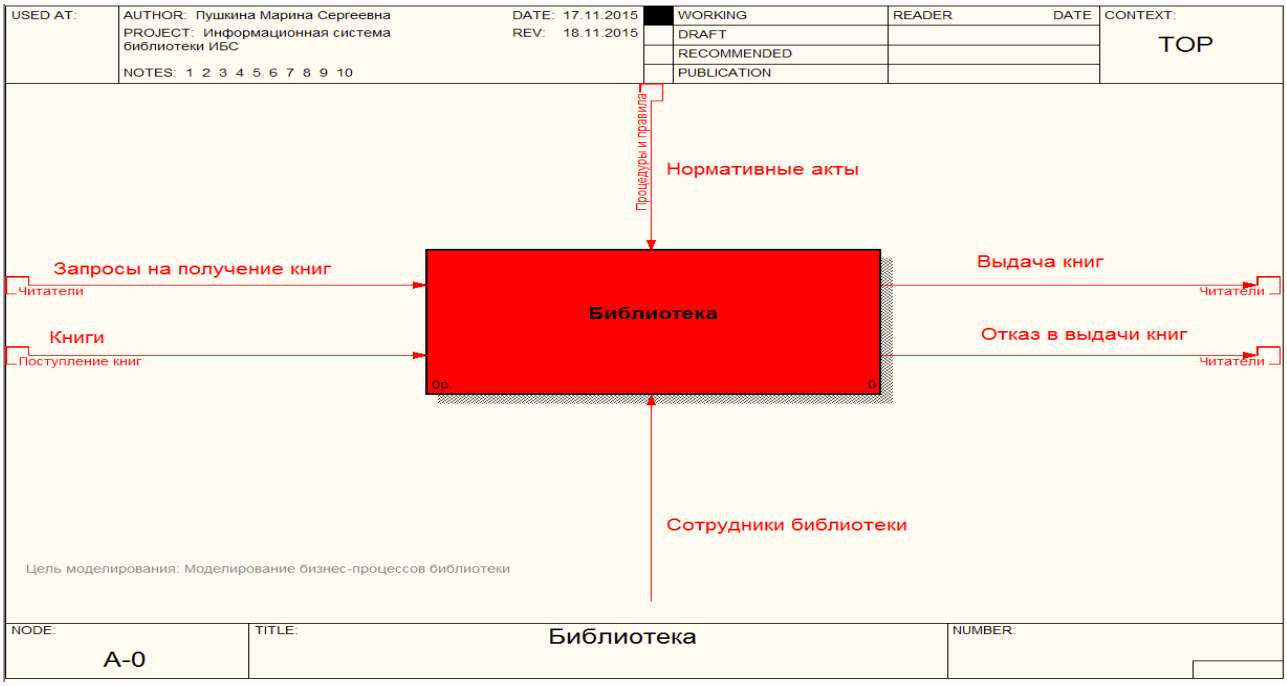


Рисунок 1.1 – Контекстная диаграмма библиотеки

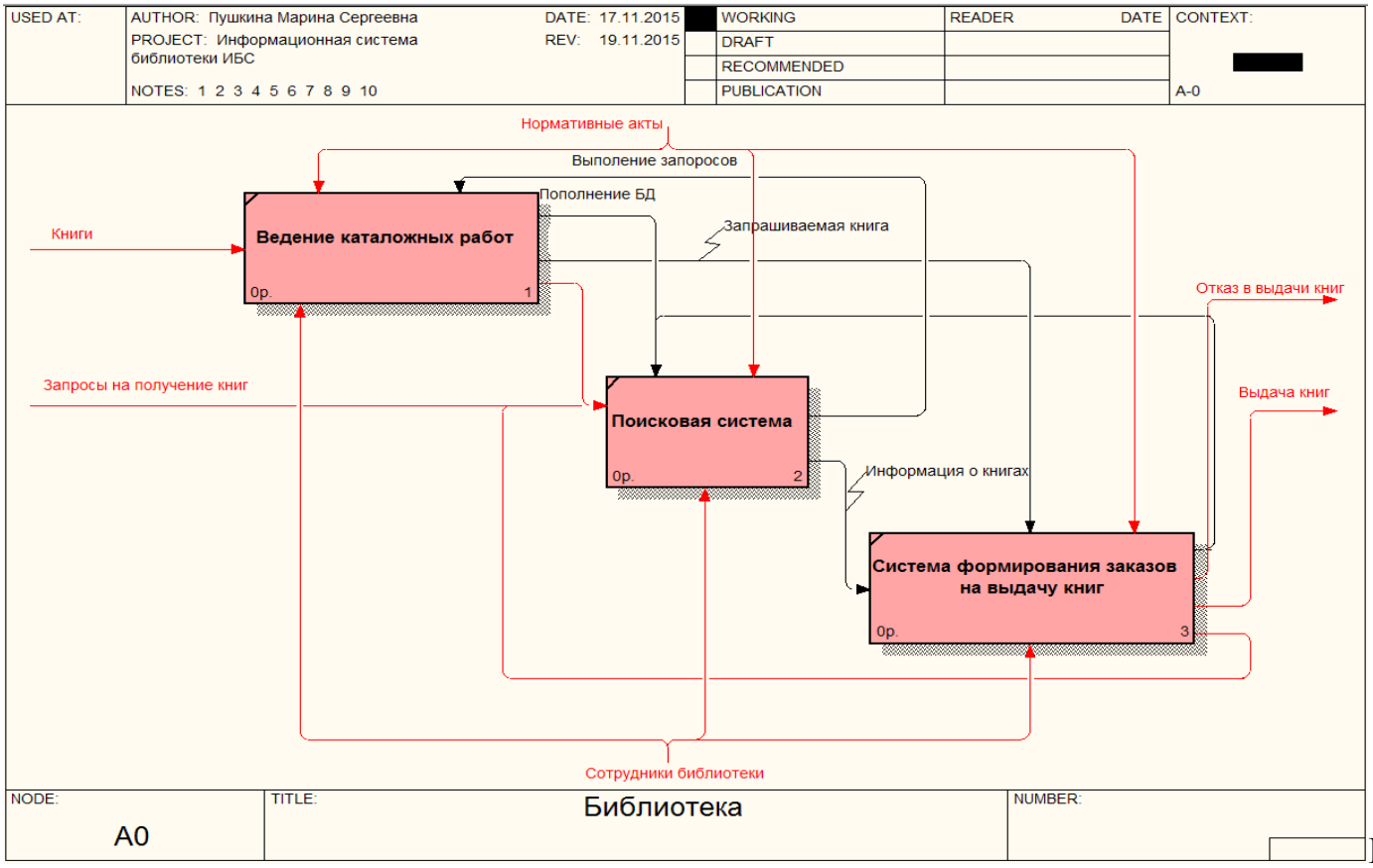


Рисунок 1.2 – Взаимодействие основных компонентов системы

Лекция 19

Тема: Структурная декомпозиция. План-график и моделирование проекта

Принципы структурной декомпозиции — это правила, по которым проект «разбирают» на составляющие. В разных источниках описывают множество принципов. **пяти правилах, которым следуют чаще всего.**

1) **В схему нужно включать все этапы работы над проектом.** То есть всё, что может повлиять на его результат. Если не включить какой-то «второстепенный» этап, есть риск забыть про него.
2) **При декомпозиции нужно соблюдать иерархию.** Это значит, что у каждого подпункта должен быть только один «родительский» пункт. Элементы (задачи) не должны дублироваться.
3) **Верхние блоки должны быть равнозначными и автономными.** Сверху располагают главные этапы работы над проектом. У каждого из них должны быть свои исходные данные и измеримый результат.
4) **Каждый блок работ нужно детализировать.** Этапы нужно декомпозировать до уровня простых задач, которые может выполнить или проконтролировать один сотрудник. Если для работы нужно несколько человек, значит, эту задачу тоже можно декомпозировать.
5) **В схеме не должно быть разночтений.** Для каждой задачи в схеме составляют описание и указывают дедлайн, ответственного и желаемый результат. Это описание не должно вызывать вопросов и уточнений.

Как сделать структурную декомпозицию шаги:

Менеджер проектов может сделать декомпозицию самостоятельно или привлечь к этому команду.

Чтобы получить иерархическую схему работ, нужно пройти семь шагов.

Первый шаг — **постановка задачи.** На этом этапе определяют, к какому результату нужно прийти. Желательно сразу прописать чёткие цели проекта — например, «создать сайт компании с каталогом товаров и корзиной для покупок».

Второй — **поиск результатов**, необходимых для достижения целей, они же главные этапы работы. Например, для запуска сайта нужно:

- собрать информацию и подготовить контент;
- провести анализ конкурентов, рынка, компании;
- составить техническое задание;
- разработать интерфейс;
- выбрать стек разработки и написать код;
- выполнить вёрстку страниц;
- провести тестирование.

Третий — **перечисление блоков работ**. Это самые важные этапы — те, что были определены на предыдущем шаге. Для каждого блока нужно прописать результат —

чего нужно достичь на этом этапе. Например, на этапе составления технического задания — получить документ, описывающий требования к сайту.

Четвёртый — **разбивка блоков на задачи**. Каждый этап работы нужно детализировать до задач и по каждой из них определить дедлайн. На основе этих дедлайнов можно определить сроки сдачи каждого блока и проекта в целом.

Пятый — **распределение задач.** На этом этапе нужно проанализировать трудозатраты и загруженность команды — и распределить задачи между сотрудниками. Если они не успеют выполнить в срок все работы, можно подключить подрядчиков и назначить тех, кто будет контролировать их работу.

Шестой — **составление бюджета**. Исходя из стоимости работы специалистов и подрядчиков, нужно рассчитать стоимость всего проекта. При необходимости в неё включают дополнительные затраты — например, расходы на покупку софта.

Седьмой — **создание контрольных точек.** Благодаря декомпозиции проект детализирован и контролировать каждую задачу не нужно. Но стоит определить контрольные точки — ключевые этапы проекта, результат которых нужно утвердить с клиентом. Часто в качестве контрольных точек используют блоки — главные этапы, которые мы нашли на втором шаге.

Структурная декомпозиция базируются на таких планах проекта, включая:

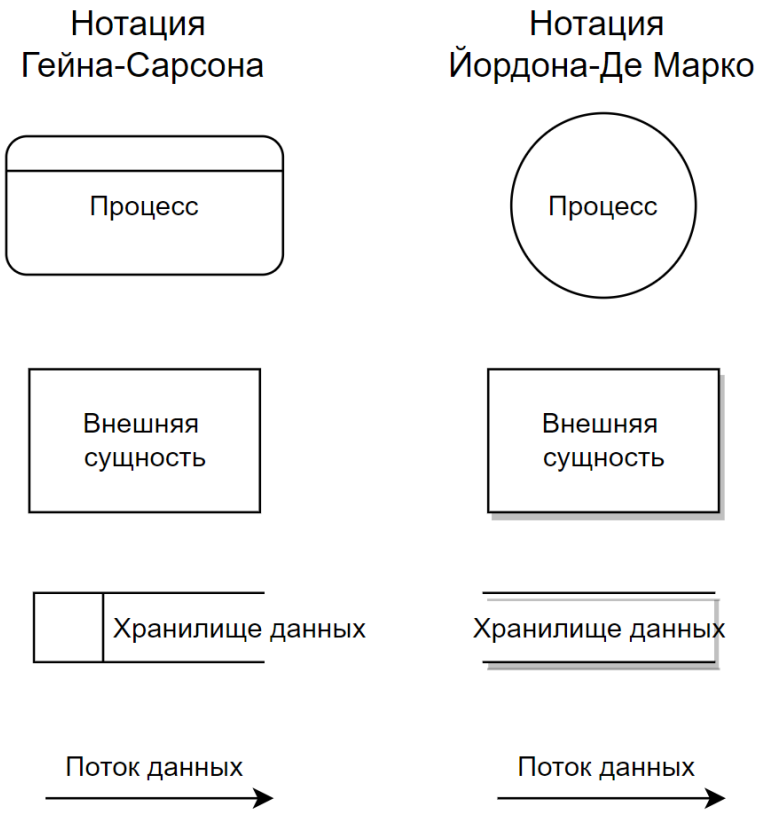
- Календарный план-график проекта
- План распределение ресурсов
- Бюджетный план
- План управления качеством
- План управления рисками

Лекция 20

Тема: Диаграмма потоков данных DFD

DFD-нотация относится к SADT-методологии и соответствует структурному подходу, поддерживая принципы декомпозиции, иерархической упорядоченности и смыслового разделения сущностей. Хотя DFD и не содержит логических операторов (XOR, AND, OR), , а также имеет очень ограниченное число элементов, она отлично **позволяет описать последовательность возникновения, изменения и преобразования данных через их движение между процессами и хранилищами.** Существует 2 разновидности DFD-диаграмм (Гейна-Сарсона и Йордана-Де Марко), которые немного отличаются лишь обозначениями некоторых элементов.

Итак, DFD-диаграмма включает следующие компоненты:

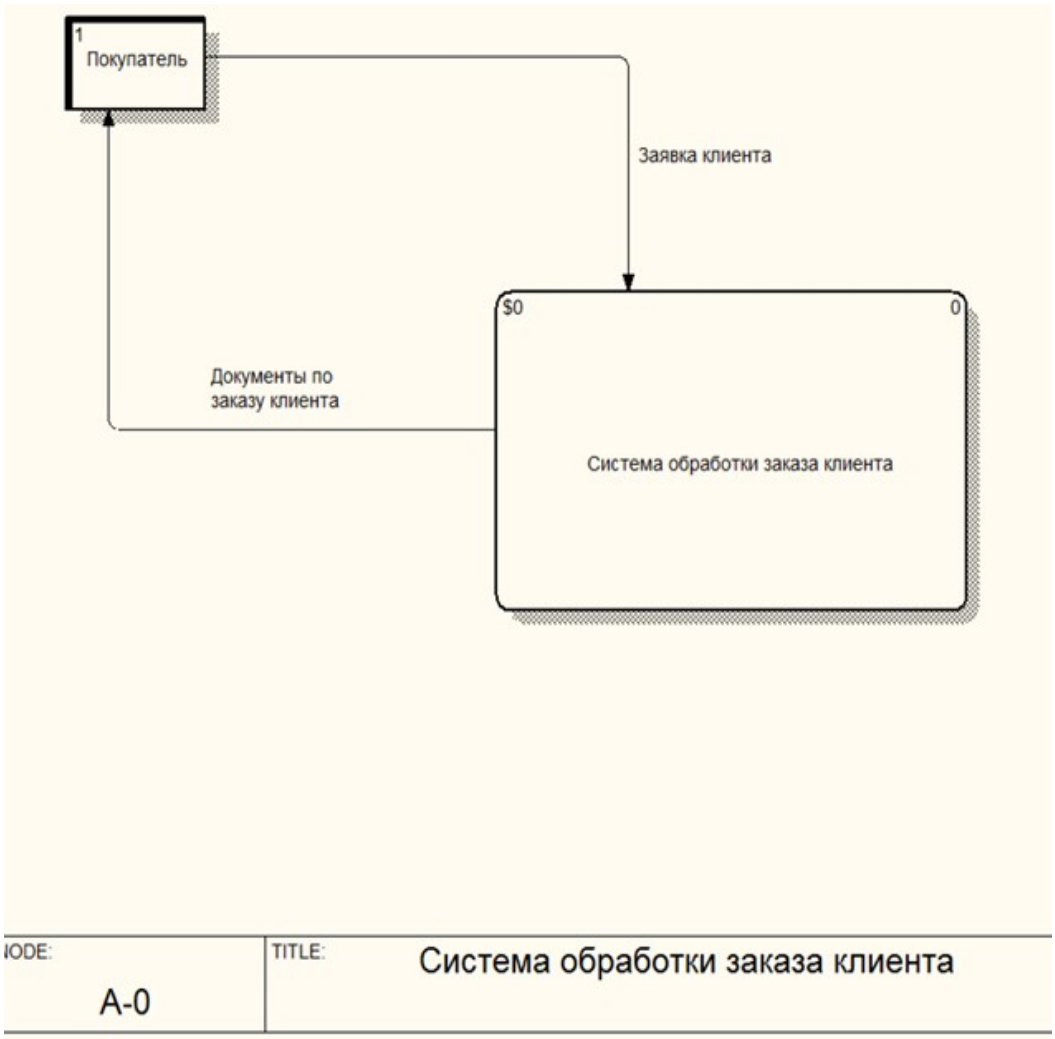


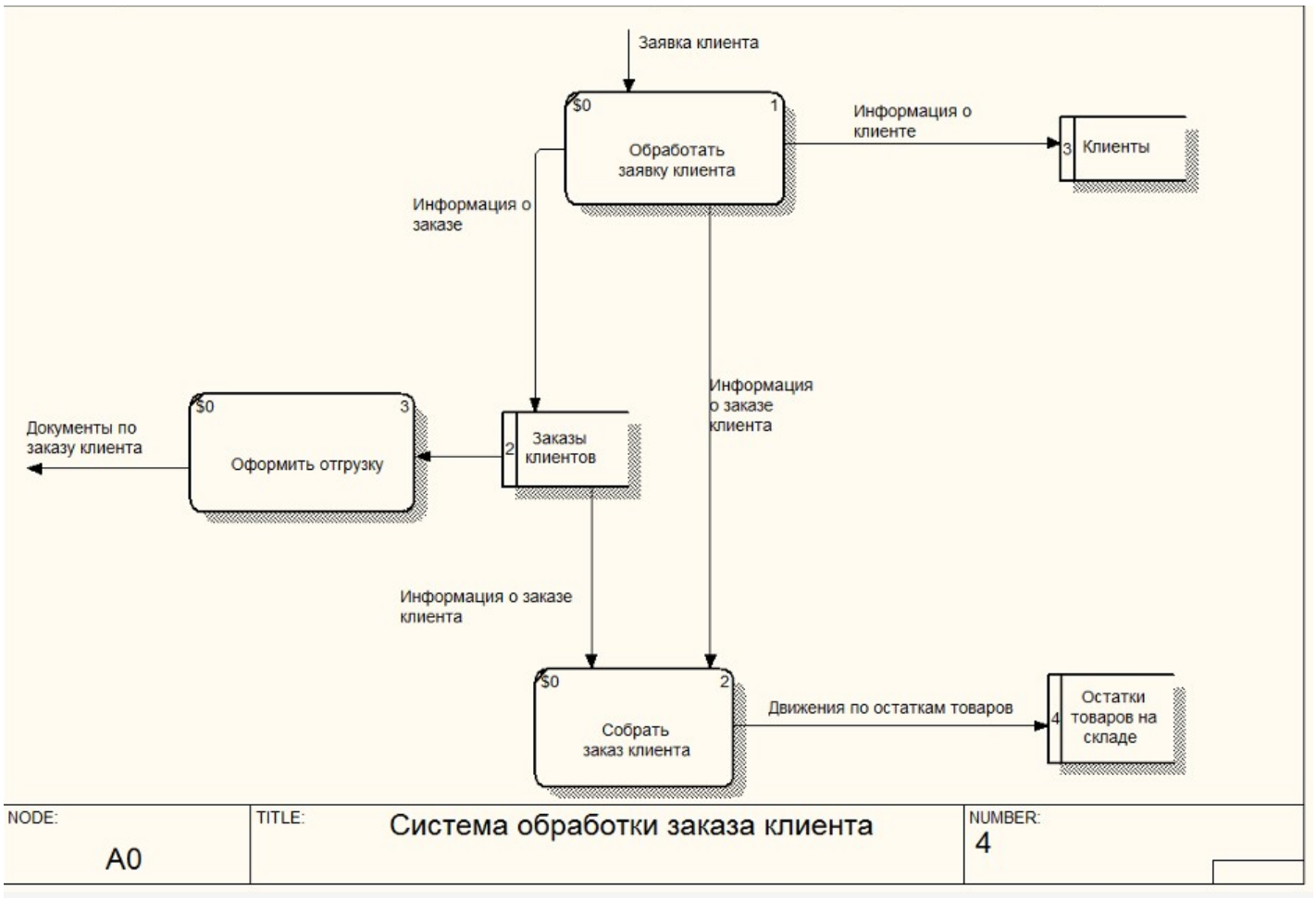
- ***Процесс*** – функция или действия по обработке данных. Обозначается в виде круга или прямоугольника со скругленными краями и горизонтальной чертой внутри. Поскольку используется для описания действия, называется как глагол, например, «отправить заявку», «просмотреть документ» и пр.
- ***Внешняя сущность*** – объект за пределами моделируемой системы, который является отправителем или получателем данных – человек, внешний сервис, носитель информации, сторонний источник данных и пр. Обозначается квадратом. Поскольку является объектом, называется как существительное, к примеру, «Клиент», «Поставщик», «БД ГИБДД», «Госуслуги» и пр.
- ***Хранилище данных*** – источник, приемник или промежуточное хранилище данных внутри моделируемой системы – база данных, таблица, документ, список, файл и пр. Обозначается в виде прямоугольника с незакрытым правым краем, может иметь вертикальную черту слева. Поскольку является объектом, называется как существительное: «Список дел», «1С:Предприятие», «CRM-система», «Файл с данными заказов», «Заявка» и пр.
- ***Поток данных*** – непосредственно данные, которые входят в процессы и хранилища или выходят из них. Например, «ФИО клиента», «Номер договора», «Сведения по заявке», «Запрос» и т.д. Обозначаются как сплошные стрелки с подписями.

Несколько правил построения диаграмм:

- Процесс должен иметь входной и выходной поток данных.
- Хранилища данных также должны иметь входные и выходные потоки данных.
- Данные с внешних сущностей должны обязательно проходить через процесс чтобы попасть в хранилище.

В DFD диаграммах также выделяют 2 разные нотации. Поэтому стоит обращать внимание на условные обозначения каждого элемента в зависимости от используемой нотации. Ниже представил картинку сравнения элементов разных нотаций.





Лекция 21

Тема: Диаграмма сущность-связь ER

МОДЕЛЬ "СУЩНОСТЬ-СВЯЗЬ"

Средством моделирования предметной области на этапе концептуального проектирования является модель "сущность-связь". Часто ее называют ER-моделью. В ней моделирование структуры данных предметной области базируется на использовании

графических средств ER-диаграмм (диаграмм "сущность-связь"). В наглядном виде они представляют связи между сущностями.

Основные понятия ER-диаграммы — сущность, атрибут, связь.

Сущность — это некоторый объект реального мира, который может существовать независимо. Сущность имеет экземпляры, отличающиеся друг от друга значениями атрибутов и допускающие однозначную идентификацию.

Атрибут — это свойство сущности.

Например, сущность КНИГА характеризуется такими атрибутами, как автор, наименование, цена, издательство, тираж, количество страниц.

Конкретные книги являются экземплярами сущности КНИГА.

Они отличаются значениями указанных атрибутов и однозначно идентифицируются атрибутом "наименование".

Атрибут, который уникальным образом идентифицирует экземпляры сущности, называется ключом. Может быть составной ключ, представляющий комбинацию нескольких атрибутов.

Предположим, что проектируется БД, предназначенная для хранения информации о деятельности некоторого банка.

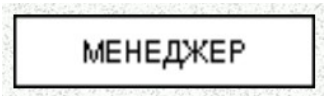
Этот банк имеет филиалы.

Филиалы управляются менеджерами.

Клиенты имеют в филиалах счета разных типов — текущие, срочные, до востребования, депозитные, карточные.

Филиалы обрабатывают эти счета.

Описываемую предметную область назовем БАНК. В ней могут быть выделены четыре сущности: филиал, менеджер, счет, клиент.



На ER-диаграмме сущность изображается прямоугольником, в котором указывается ее имя.

В реальном мире существуют связи между сущностями. Связь представляет взаимодействие между сущностями. Она характеризуется мощностью, которая показывает, сколько сущностей участвует в связи. **Связь между двумя сущностями называется бинарной**, а связь между более чем с двумя сущностями — тернарной.

В рассматриваемой предметной области БАНК можно выделить три связи.

1. МЕНЕДЖЕР — УПРАВЛЯЕТ — ФИЛИАЛ
2. ФИЛИАЛ — ОБРАБАТЫВАЕТ - СЧЕТ
3. КЛИЕНТ — ИМЕЕТ – СЧЕТ

На ER-диаграмме связь изображена ромбом.

Важной характеристикой связи является тип связи (кардинальность).

Рассмотрим типы связей 1-3.

Так как менеджер управляет только одним филиалом, то каждый экземпляр сущности МЕНЕДЖЕР может быть связан не более чем с одним экземпляром сущности

Рис. 1.1. ER-диаграмма связи 1:1

ФИЛИАЛ. В этом случае связь 1 имеет тип **"один-к-одному"** (1:1). На рис. 1.1 представлена ЕВ-диаграмма для связи типа 1:1.

МЕНЕДЖЕР УПРАВЛЯЕТ ФИЛИАЛ

Так как филиал обрабатывает несколько счетов, а счет обрабатывается только одним филиалом, то каждый экземпляр сущности ФИЛИАЛ может быть связан более чем с одним

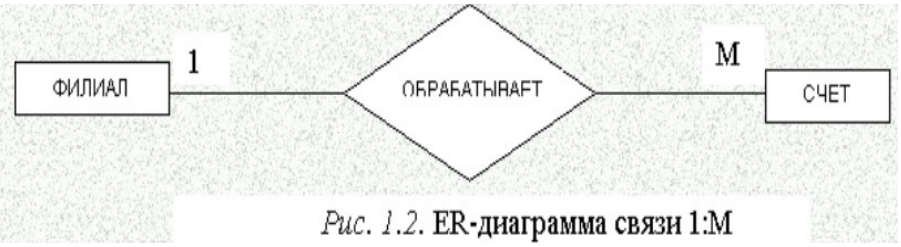


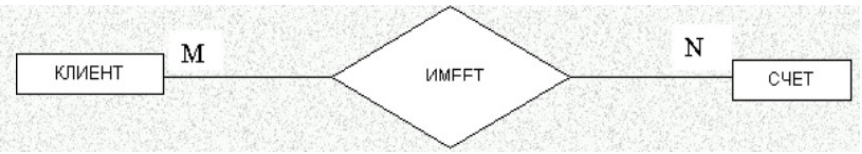
Рис. 1.2. ER-диаграмма связи 1:M

экземпляром сущности СЧЕТ, а каждый экземпляр сущности СЧЕТ может быть связан не более чем с одним экземпляром сущности ФИЛИАЛ. В этом случае связь 2 имеет

тип "**один-ко-многим**" (1:M).На рис. 1.2 представлена ER-диаграмма для связи типа 1:M.

ФИЛИАЛ ОБРАБАТЫВАЕТ СЧЕТ

Так как счет может совместно использоваться несколькими клиентами и клиент может иметь несколько счетов, то каждый экземпляр сущности СЧЕТ может быть связан с несколькими экземплярами сущности КЛИЕНТ и каждый экземпляр сущности КЛИЕНТ может быть связан с несколькими экземплярами сущности СЧЕТ. В этом случае связь 3



имеет тип "**многие-ко-многим**" (M:N). На рис. 1.3 представлена ER-диаграмма для связи типа M:N

КЛИЕНТ ИМЕЕТ СЧЕТ

Рассмотрим понятие класс принадлежности сущности.

1) **Если каждый экземпляр сущности А связан с экземпляром сущности В. то класс принадлежности сущности А является обязательным.**

Этот факт отмечается на ER-диаграмме черным кружочком, помещенным в прямоугольник, смежный с прямоугольником сущности А.

2) **Если не каждый экземпляр сущности А связан с экземпляром сущности В, то класс принадлежности сущности А является необязательным**.

Этот факт отмечается на ER-диаграмме черным кружочком, помещенным на линии связи возле прямоугольника сущности А.

В качестве примера на рис. 1.4 изображены возможные ER-диаграммы для связи M:M с учетом класса принадлежности сущности.

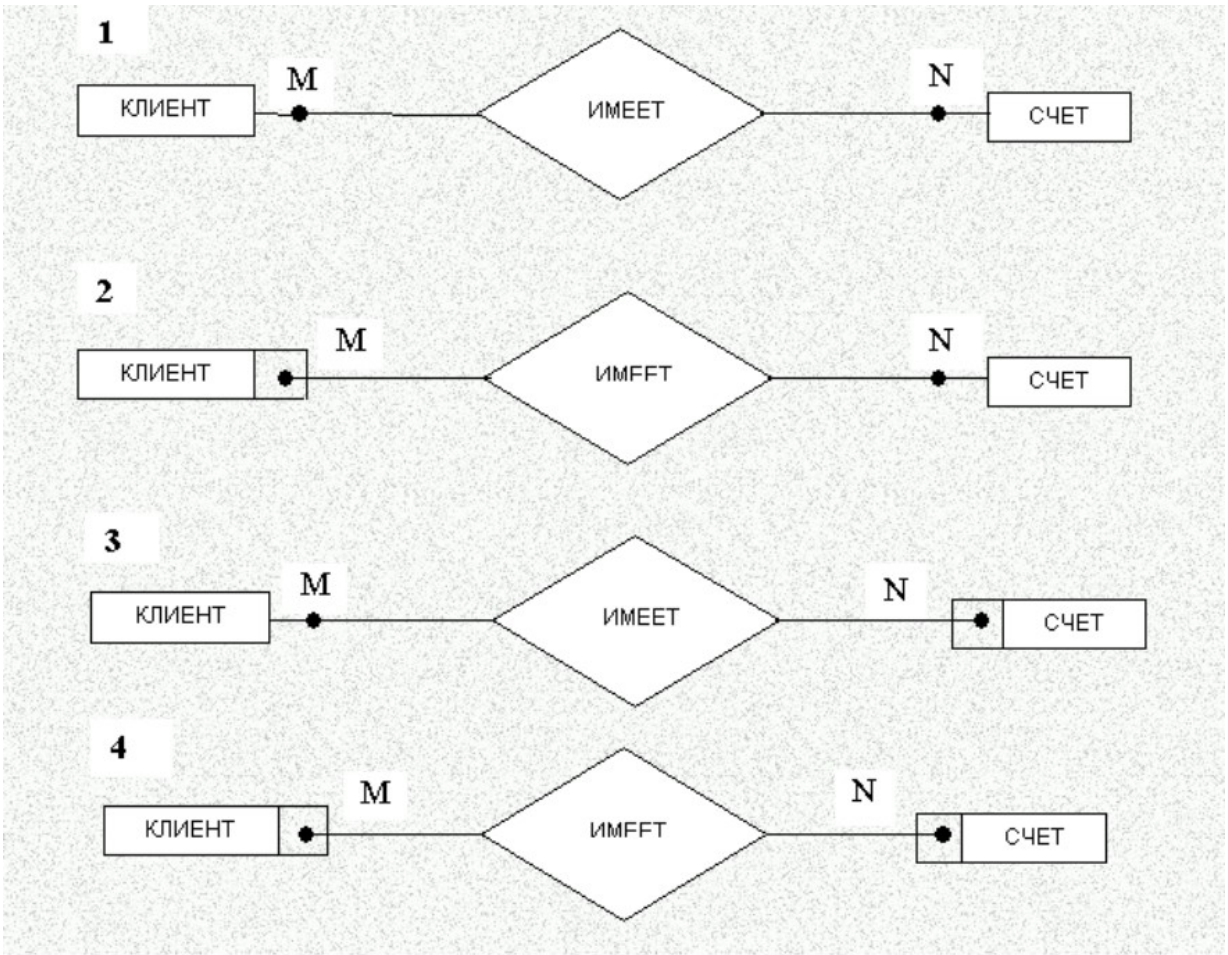


Рис. 1.4. **ER- диаграммы связи M:N с учетом класса принадлежности сущности**

На Er-диаграмме 1 класс принадлежности обеих сущностей необязательный.

На ER-диаграмме 2 класс принадлежности сущности КЛИЕНТ обязательный, а сущности СЧЕТ необязательный.

На Er-диаграмме 3 класс принадлежности сущности КЛИЕНТ необязательный, а сущности СЧЕТ обязательный.

На ER-диаграмме 4 класс принадлежности обеих сущностей обязательный.

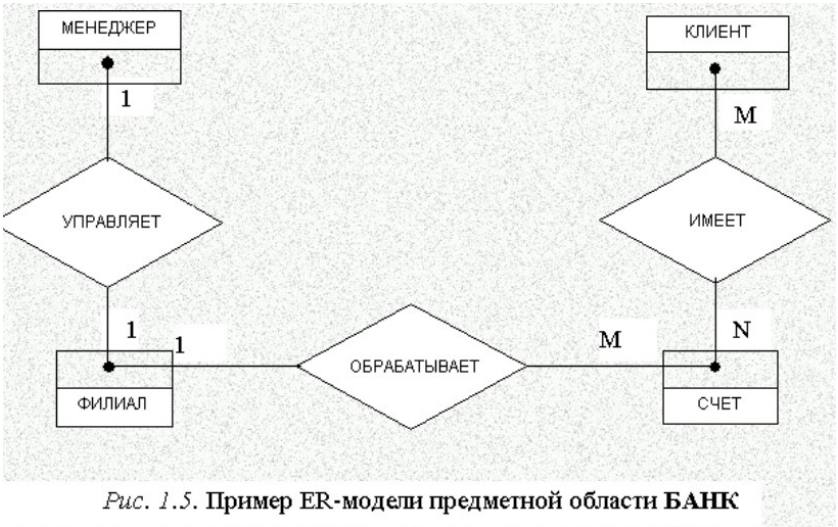


Рис. 1.5. **Пример ER-модели предметной области БАНК**

Предположим, что в рассматриваемой предметной области БАНК класс принадлежности всех четырех сущностей является обязательным. Тогда ER-модель области БАНК будет иметь вид, представленный на рисунке.

Каждая из четырех сущностей приведенной ER-модели может быть описана своим набором атрибутов (рис. 1.6).

ER-модель в совокупности с наборами атрибутов сущностей может служить примером концептуальной модели предметной области или концептуальной схемы базы данных.

В связи с наглядностью представления концептуальных схем баз данных ER-модели получили широкое распространение в case-средствах. Эти средства предназначены для автоматизированного проектирования реляционных баз данных.

МЕНЕДЖЕР
Номер менеджера (НМ)
Стаж работы (СТАЖ)
Специальность (СПЕЦ)

ФИЛИАЛ
Номер филиала (НФ)
Адрес филиала (АДР_Ф)

СЧЕТ
Номер счета (НС)
Тип счета (ТИП)
Остаток на счете (ОСТ)

КЛИЕНТ
Номер клиента (НК)
Ф.И.О. клиента (ФИО_К)
Социальное положение (СОЦ)
Адрес клиента (АДР_К)

Рис.1.6. Наборы атрибутов сущностей предметной области **БАНК**

Тогда ER-модель предметной

ЛЕКЦИЯ 22

Тема: Разработка целевой структуры ПО. WBS, как средство построения иерархической зависимости программных модулей.

Цель: изучить структурную декомпозицию. Научиться применить и разделять на классы элементов

Имеющие дело со сложными ИТ проектами руководители скажут, что разделение задач на более мелкие и управляемые части делает рабочий процесс намного проще

разработка целевой структуры проекта

когда мы говорим о целевой структуре, мы предполагаем определения границ проекта.

Цель *разработки целевой структуры* — определить границы содержания проекта и работы, направленные на успешное выполнение и завершение проекта.

Управление содержанием *определяет иерархическую структуру* работы (ИСР WBS Work Breakdown Structure) и *декомпозицию проекта* на более мелкие и более управляемые компоненты или разработку целевой структуры проекта.

WBS(иерархическая структура работы)-это согласованная с результатами проекта иерархическая декомпозиция работ, которые команда проекта должна выполнить для достижения целей проекта и создания оговоренных результатов проекта

Мы делим наш проект на взаимосвязанные части и определяем их иерархию(в всегда есть самое главное, что станет точкой начала)

Почему следует использовать WBS структурные декомпозиции?

1. *Помогает правильно организовать проекты;*
2. *Оказывает помощь в описании содержания проекта;*
3. *Помогает распределить обязанности;*
4. *Показывает основные этапы проекта и все ракурсы контроля;*
5. *Позволяет правильно оценить затраты, риски и время работ.*

целевая декомпозиция проекта является системообразующим *инструментом для управления проектом*, так как строится для того, чтобы определить *точное количество и корректный контент пакета задач* проекта

инструмент для управления проектом и для точного определения количества пакетов и его составляющего контента. Т.е. что у нас будет входить в один узел в плоть до описания каждого сотрудника участвующего в разработке и его ф-й.

стандартная целевая структура (ДЕРЕВО ЦЕЛЕЙ *) состоит из четырех уровней*, но в некоторых случаях есть необходимость в использовании пятого уровня, в зависимости от масштаба

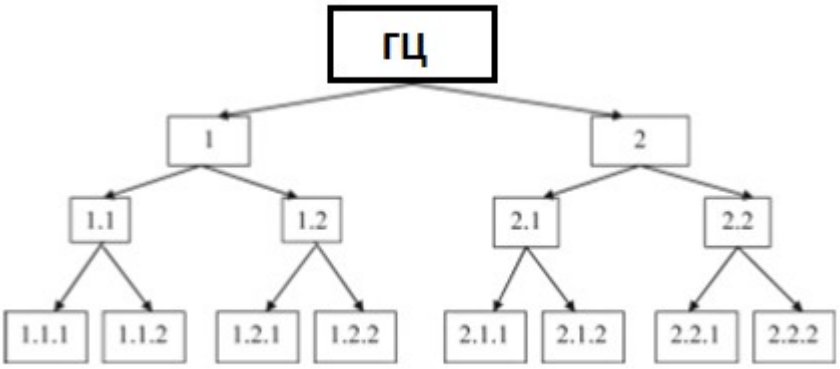


Рис. 2.1. «Дерево целей»

правила построения целевой структуры: построение целевой структуры, как правило, начинается сверху вниз, отражая логику дедуктивного продвижения от уровня большей степени общности к уровню меньшей степени общности; от абстрактного к конкретному; от общего к частному.

первый уровень — *уровень генеральной цели* ГЦ, которая представляет собой формулировку конечной целевой установки того, что, собственно, планировалось достичь в проекте.

второй уровень — *это уровень общих целей* (обозначены как 1 и 2), которые представляют собой основные направления деятельности.

Это то, на что направлены действия проектанта.

третий уровень — *уровень специфических целей* (обозначены как 1.1, 1.2 и 2.1, 2.2), которые представляют собой основные формы работ и обеспечивают основные направления деятельности.

Действия проекта разобраны более детально на под части, основная работа проекта

Четвертый уровень — уровень конкретных задач проекта обозначены 1.1.1, 1.1.2, 1.2.1, 1.2.2 и т.), представляющих собой те конкретные работы проекта, которые, будучи выполнены в установленные сроки, определенным способом, с запланированным результатом, приведут в конечном счете к выполнению гц проекта.

Как каждая задача должна выполнятся, кем в какие сроки и каким инструментами

необходимо заметить, способ структурной декомпозиции *предпринимается ради достижения четвертого уровня* и точного определения всех задач проекта.

дальнейшие действия проектанта по разработке проекта будут связаны только с четвертым уровнем целевой структуры.

правила построения целевой структуры:

1. цели должны формулироваться предельно ясно, четко, однозначно;
2. они должны быть независимы и несводимы друг к другу;
3. цель более высокого уровня должна быть разбита не менее чем на две цели более низкого уровня;
4. цели более низкого уровня в сумме должны давать цель более высокого уровня как по содержанию, так и по объему понятий;
5. задачи должны быть сформулированы как конкретные распоряжения менеджмента, однозначно истолкованные и не оставляющие свободы для интерпретаций.

специалисты **выделяют следующие подходы к построению WBS**:

1.продуктивный
2.функциональный
3.оргнизационый

1. Продуктовый — построение по компонентам продукции проекта. в качестве элементов WBS выбираются элементы продукции проекта, его материальные результаты. для определения названия пакетов работ и отдельных работ используются существительные

В результате выпуска ПП мы можем пощупать итог нашего конструирования, его объекты(стол, парта, пучка)

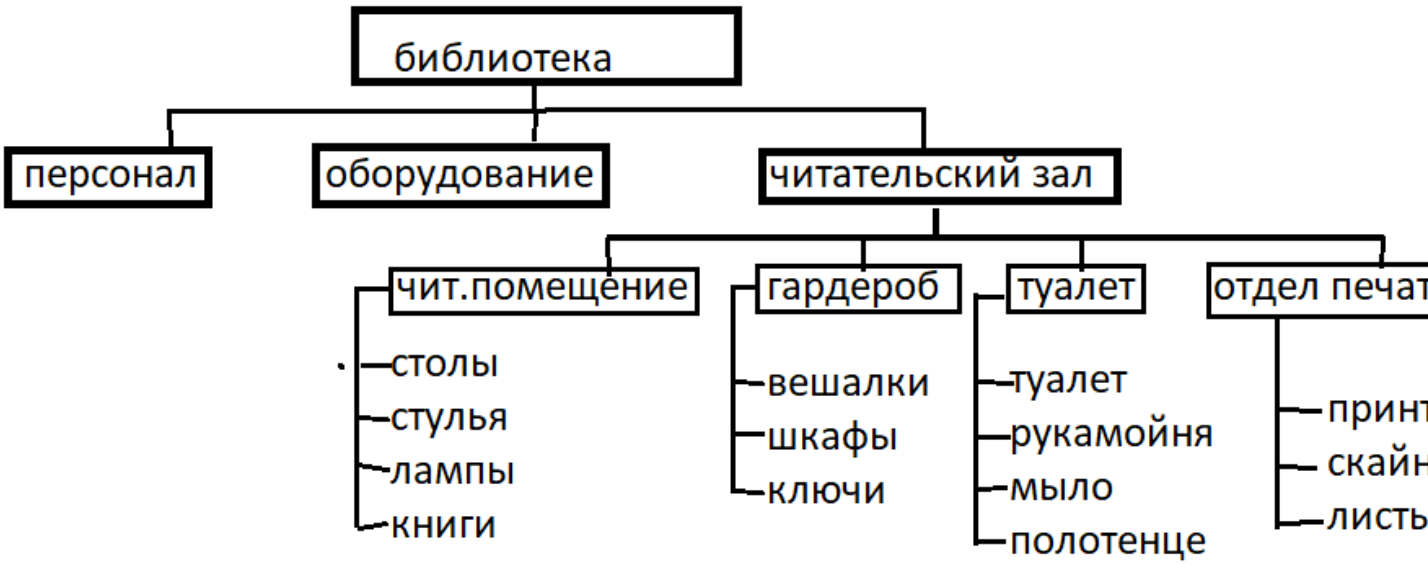


Рис 1 Схема продуктового принципа построения целевой структуры

2. Функциональный — построение WBS по функциональным элементам деятельности. в качестве элементов WBS выбираются элементы операций технологического цикла производства продукции. для определения названия пакетов работ и отдельных работ используются глаголы или отглагольные существительные (рис. 2).

В результате выпуска ПП мы можем получаем результаты не материальные, а исследовательские, набор ф-й, какой либо анализ области , смету, прогноз погоды

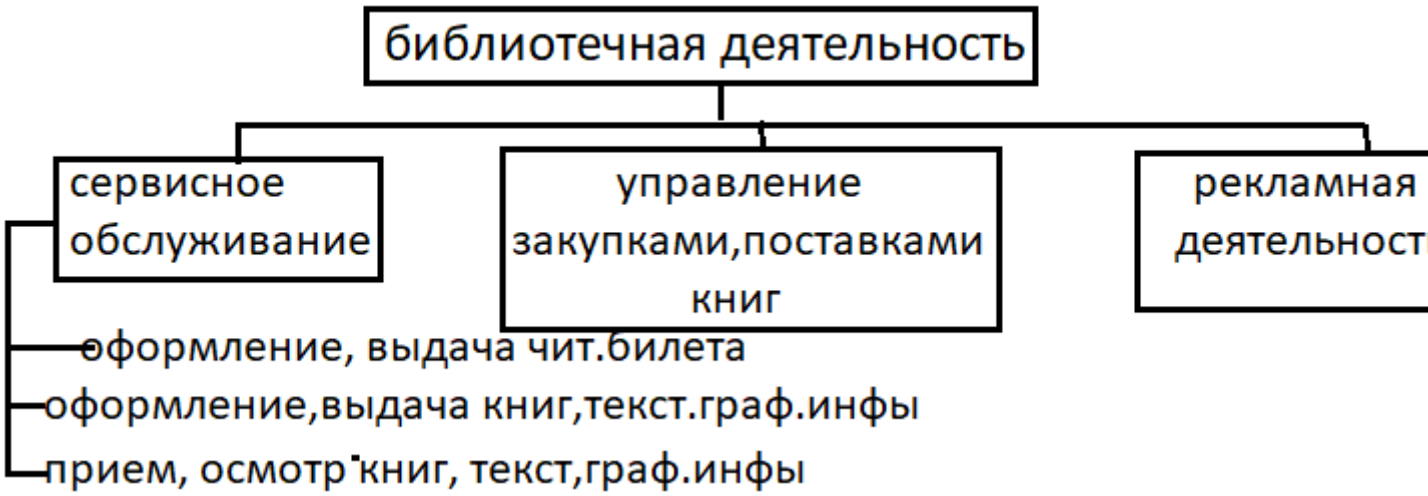


Рис.2 схема функционального построения целевой структуры

3. Организационный — построение WBS по элементам организационной структуры. в качестве элементов WBS выбираются элементы организационной структуры. для определения пакетов работ и отдельных работ используются в основном существительные

Распределение обязанностей с кем сотрудничает, в плоть да конкретного человека.

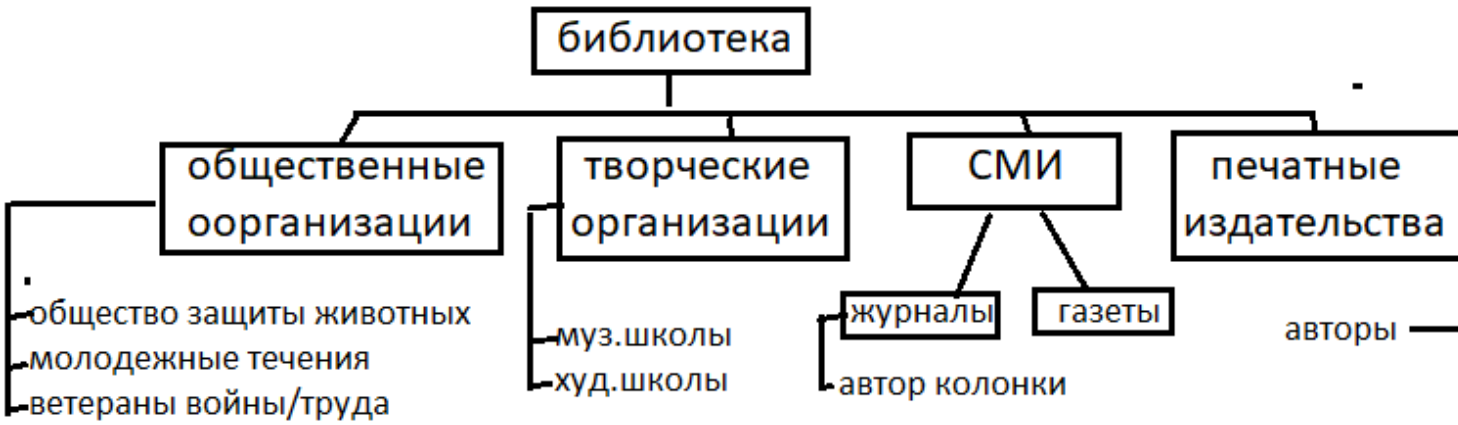


Рис.3 схема организационного принципа построения целевой структуры

WBS + диаграмма Ганта = улучшенный процесс планирования

Иерархическая структура работ — распространенный и удобный способ планирования ИТ и любых других проектов. А если отобразить ее диаграммой Ганта, то в таком виде она значительно упрощает процесс не только планирования, но и управления. С ней можно:

- Ставить задачи и распределять их между участниками;
- Задавать сроки выполнения задач, их продолжительность, прогресс;
- Устанавливать зависимости между задачами;
- Визуализировать важные ключевые события — вехи;
- Определять критический путь;
- Взаимодействовать с командой.

Контрольные вопросы:

42. что подразумевает под собой разработка целевой структуры?
43. Что такое WBS?
44. Что такое целевая декомпозиция?
45. Из скольких уровней состоит дерево целей
46. Опишите каждый уровень дерева целей
47. Назовите правила построения целевой структуры
48. Назовите подходы построения WBS
49. Опишите каждый подход построения WBS
50. В связке с какой моделью лучше всего использовать WBS?

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами

11. https://habr.com/post/189626/
12. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
13. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 23

Тема: метод проектирования системы по средствам имитационного моделирование.
Цель: изучить основные методы проектирования.

В практике проектирования наиболее часто используются такие методы:

37. мозговой штурм,
38. экспертная оценка,
39. метод аналогий,
40. календарное планирование,
41. структурная декомпозиция,
42. имитационное моделирование,

Метод позволяет виртуально «отработать» некоторый период времени с заданными параметрами деятельности и посмотреть, какие результаты получит компания за этот период.

За исходные данные в имитационном моделировании берутся результаты работы компании за прошедшие периоды. На основе них строится упрощённая модель, представляющая собой макет вашей компании, существующий только в электронном виде. «Работает» такая модель в десятки тысяч раз быстрее, чем ваши сотрудники. Таким образом, вы можете за несколько минут увидеть, как внесённые изменения отразятся на компании в целом и на каждом отдельном подразделении.

Таким образом можно, например, проверить, как предлагаемые решения повлияют на работу компании до начала их претворения в жизнь.

Современные системы бизнес-моделирования позволяют провести несколько расчётов с различными параметрами и выбрать наиболее эффективный вариант из предложенных командой руководителей.

Имитационное моделирование -метод, позволяющий строить модели, описывающие процессы так, как они проходили бы в действительности, так же это метод исследования, при котором изучаемая система заменяется моделью с достаточной точностью описывающей реальную систему и с ней проводятся эксперименты с целью получения ин-формации об этой системе. Экспериментирование с моделью называют имитацией (имитация - это постижение сути явления, не прибегая к экспериментам на реальном объекте).

Имитационное моделирование- это метод, позволяющий строить модели, описывающие процессы так, как они проходили бы в действительности.

Такую модель можно «проиграть» во времени как для одного испытания, так и заданного их множества.

К имитационному моделированию прибегают, когда:

1. дорого или невозможно экспериментировать на реальном объекте;
2. невозможно построить аналитическую модель: в системе есть время, причинные связи, последствие, нелинейности, стохастические (случайные) переменные;
3. необходимо сымитировать поведение системы во времени.

Цель имитационного моделирования состоит в воспроизведении поведения исследуемой системы на основе результатов анализа наиболее существенных взаимосвязей между ее элементами или другими словами - разработке симулятора исследуемой предметной области для проведения различных экспериментов.

Исследовать проект, ускоряем нужные процессы и анализировать их результаты, и уже потом делаем вывод стоит новый проект создавать или лучше взять дорабатывать старый(оставаться на поддержке).

Имитационное моделирование позволяет имитировать поведение системы, во времени. Причём плюсом является то, что временем в модели можно управлять: замедлять в случае с быстропротекающими процессами и ускорять для моделирования систем с медленной изменчивостью. Можно имитировать поведение тех объектов, реальные эксперименты с которыми дороги, невозможны или опасны.

Рассмотрим чем отличается имитируемая операция от моделируемой операции

Моделируемая операция: Реальная или проектируемая операция. Описание ее может содержать следующие атрибуты:

1. участвующие в операции объекты и субъекты,
2. события, возникающие в процессе выполнения операции

Пример: операция «Забить гвоздь», которая описывается так:

- участвовали в операции: доска, молоток,гвоздь, исполнитель Хруничев Геннадий Петрович
- события: начало операции в 9-00, окончание — в 9-01.

Имитирующая операция: *Объект, созданный в программе для имитации* моделируемой операции.

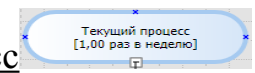
Имитирующая операция описывается следующими имитирующими событиями:

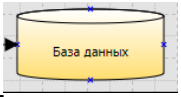
1. начало операции,
2. завершение операции,
3. завершение времени технологического ожидания,
4. постановка операции в очередь к ресурсам,
5. прерывание выполнения операции,
6. возобновление выполнения операции,
7. действие с переменной

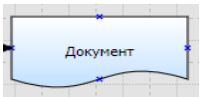
Имитационная модель *содержит элементы непрерывного и дискретного действия,* поэтому применяется для исследования динамических систем, когда требуется анализ узких мест, исследование динамики функционирования, когда желательно наблюдать на имитационной модели ход процесса в течение определенного времени.

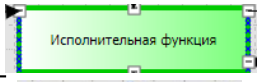
Это есть вариация бизнес моделирования!

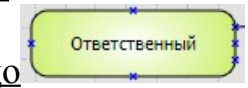
Элементы моделировании:

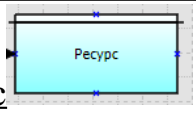
Текущий процесс 

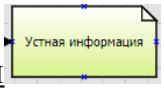
База данных 

Документ 

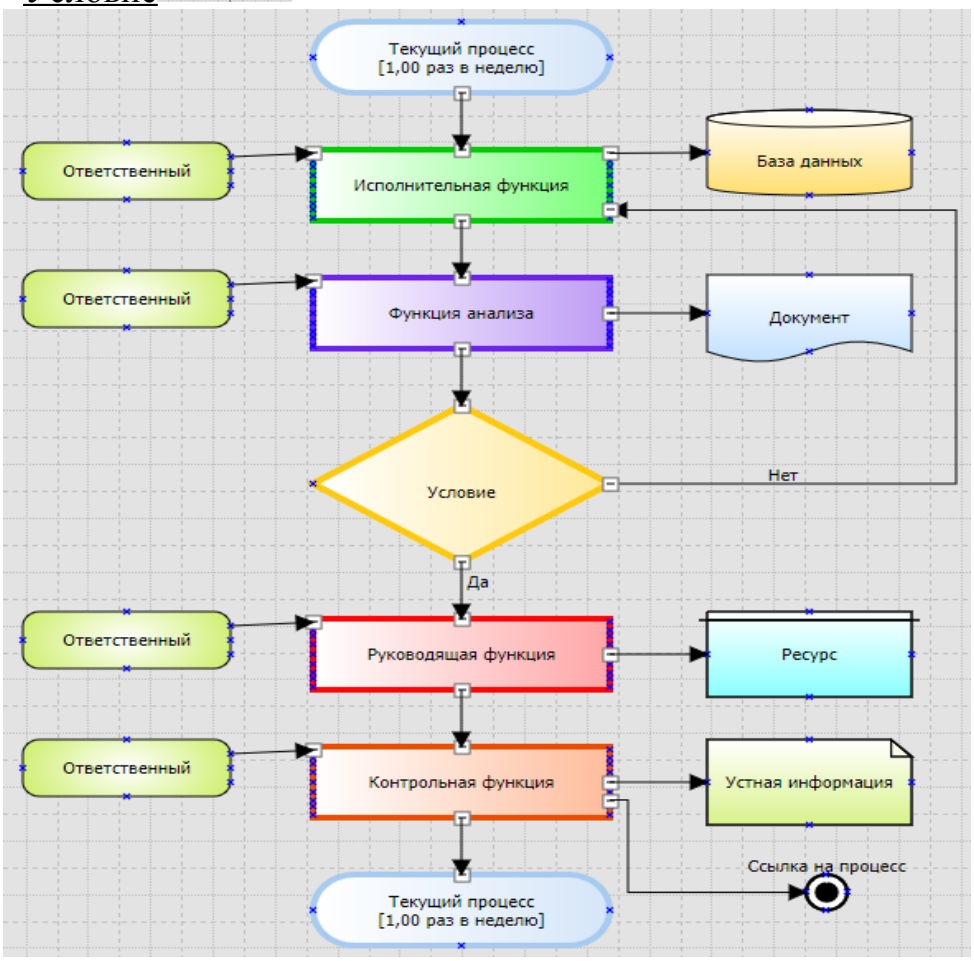
Функция 

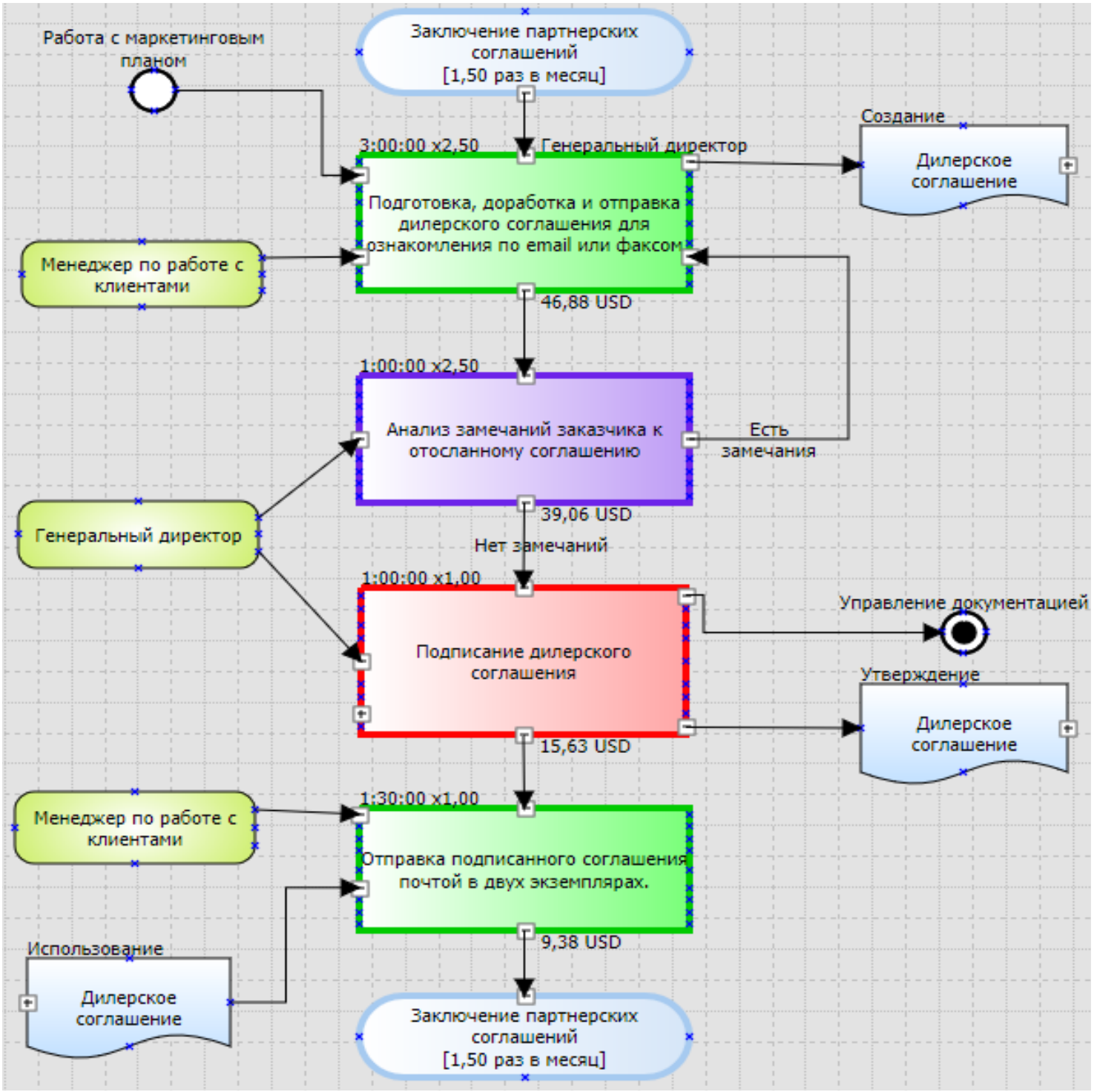
Ответственное лицо 

Ресурс 

Устная информация 

Условие 





Существует два типа имитационных моделей :

1. непрерывные
2. дискретные.

Непрерывные модели используются для систем, поведение которых изменяется непрерывно во времени.

Дискретные модели имеют дело с системами, поведение которых изменяется лишь в заданные моменты времени. Те моменты времени, в которые в системе происходят изменения, определяют события модели.

Идея метода, с точки зрения его программной реализации, состоит в следующем. Что, если элементам системы поставить в соответствие некоторые программные компоненты, а

состояния этих элементов описывать с помощью переменных состояния. Элементы, по определению, взаимодействуют (или обмениваются информацией), значит, может быть реализован алгоритм функционирования отдельных элементов, т.е., *моделирующий алгоритм*.

достоинствам имитационных моделей:

1. простота алгоритма;
2. малая связность алгоритма;
3. устойчивость к случайным сбоям компьютера, так как при большом числе реализаций (прогонов) модели сбой в одной из них исказит статистику несущественно.

Недостатком имитационного моделирования

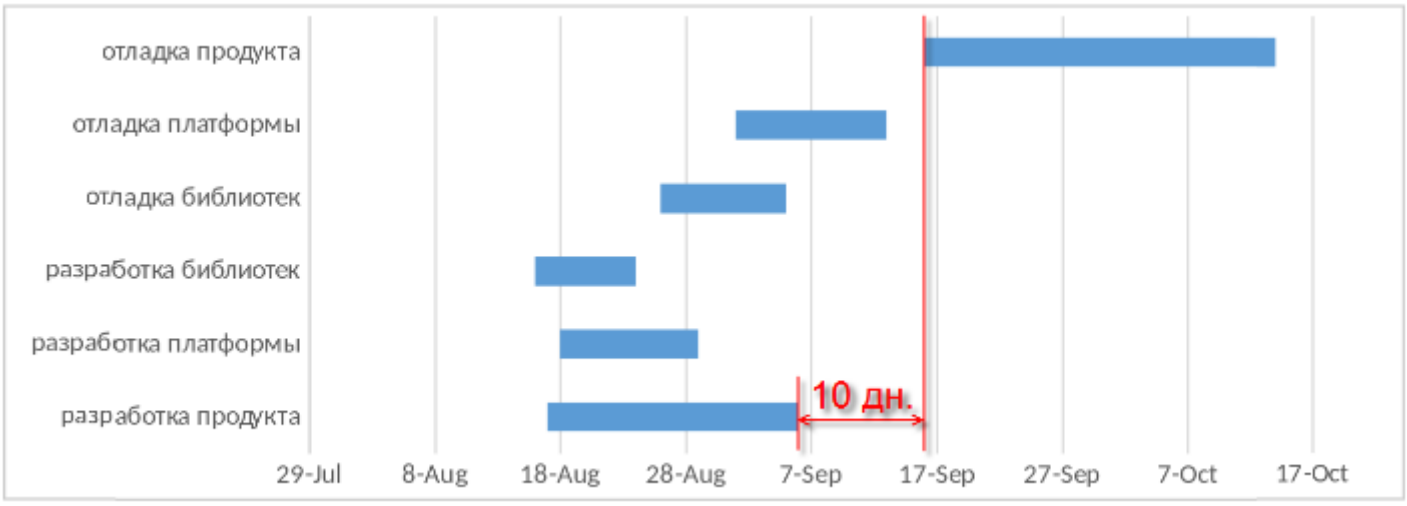
1. является то, что решение, результат является численным, частным, справедливым только для конкретных значений исходных данных.

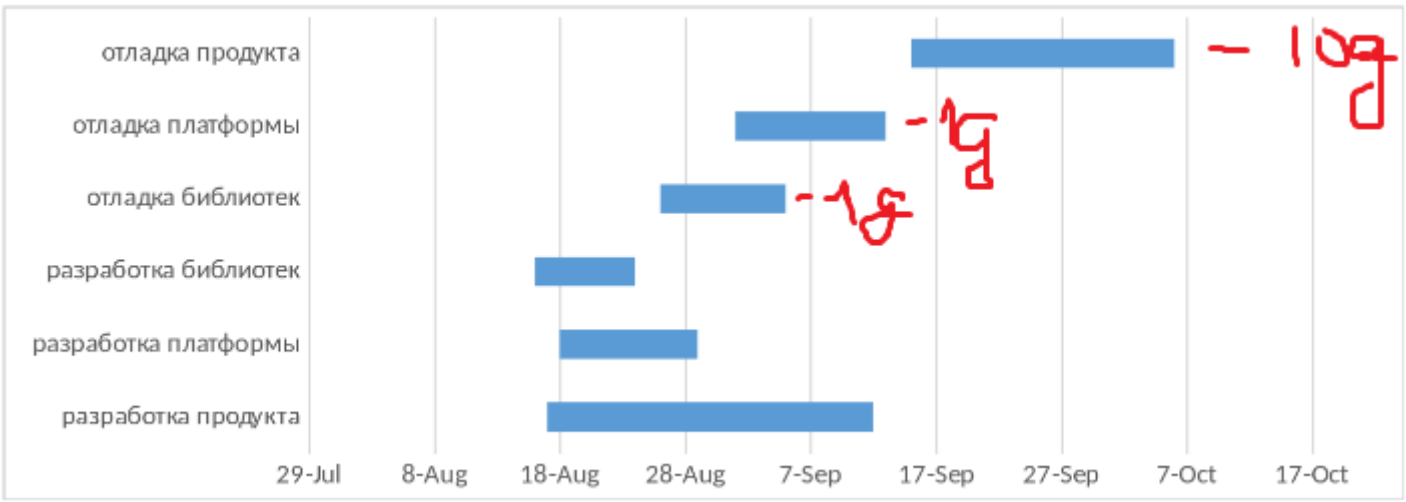
   Нет исходных данных нет моделирования

2. большое количество вариаций решений, при попытке найти нужный алгоритм

   возьмем пример разработки ПО

   что нужно смоделировать по-другому, чтобы уменьшить срок разработки и сократить затраты??





Виден большой промежуток времени в 10 дней между завершением этапа разработки продукта и началом этапа отладки продукта, в который разработчики продукта оказываются не заняты на текущем проекте.

Желательно их как можно скорее *загрузить работой по исправлению ошибок*, чтобы сократить время на переключение между задачами и исправить их «по свежим следам» быстрее.

Смоделировав протекающие процессы разработка сократиться.

Когда используют Использование имитационного моделирования, как инструмента в условиях ограниченных затрат времени, позволяет находить пути оптимизации, максимально полно учитывающие особенности взаимосвязей всех процессов компании.

Контрольные вопросы:

51. Что такое имитационное моделирование?
52. Когда используют имитационное моделирование?
53. Что такое моделируемая операция?
54. Чем отличается имитирующая операция от моделирующей операции?
55. В каких случаях используют имитационное моделирование?
56. Какие типы имитационных моделей вы знаете, опишите их
57. Назовите элементы моделирования
58. Назовите достоинства и недостатки модели

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами

14. https://habr.com/post/189626/
15. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
16. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 24

Тема: процесс руководства программным проектом

Цель: научиться равнозначно и эффективно распределять ресурсы в виде сотрудников и ПО.

сущность любого проекта заключается в деятельности, но для того, чтобы он был успешным, необходимо тщательное и продуманное управление этим проектом, служащее гарантией эффективной деятельности, ее направленности на достижение конечной цели.

управление проектами — *это методология, искусство организации, планирования, руководства, координации трудовых, финансовых, материально-технических ресурсов на протяжении всего проектного цикла*, направленное на достижение его целей путем применения современных методов, техники и технологии управления для получения определенных в проекте результатов по составу и объему работ, стоимости, времени, качеству и удовлетворению участников проекта.

задачи управления проектом:

1. определить цели проекта и провести его обоснование;

2. выявить структуру проекта (подцели, основные этапы работ, которые предстоит выполнить);

3. определить необходимый объем и источники финансирования;

4. подобрать исполнителей и сформировать команду проектантов;

5. подготовить и заключить контракты;

6. определить сроки выполнения проекта, составить график его реализации;

7. рассчитать необходимые ресурсы;

8. рассчитать смету и бюджет проекта;

9. планировать и учитывать риски;

10. обеспечить контроль за ходом выполнения проекта и многое другое.

структура управления проектом обеспечивает основу для понимания управления проектами и включает в себя следующие большие разделы

Стадии проекта

Каждый проект, большой или маленький, сложный или простой, проходит пять стадий развития

1) 

Замысел (концепция). Рождение идеи проекта.

2) *Разработка*. Создание плана проекта.

3) *Начало*. Формирование команды исполнителей.

4) *Исполнение*. Выполнение работ по проекту.

5) *Завершение*. Проект закончен

Все проекты начинаются с идеи. Например, заказчик изложил вам свои требования, или ваш босс нацелился на новые рынки сбыта, или вы придумали, как улучшить систему снабжения. Если есть идея — ваш проект уже на стадии замысла.

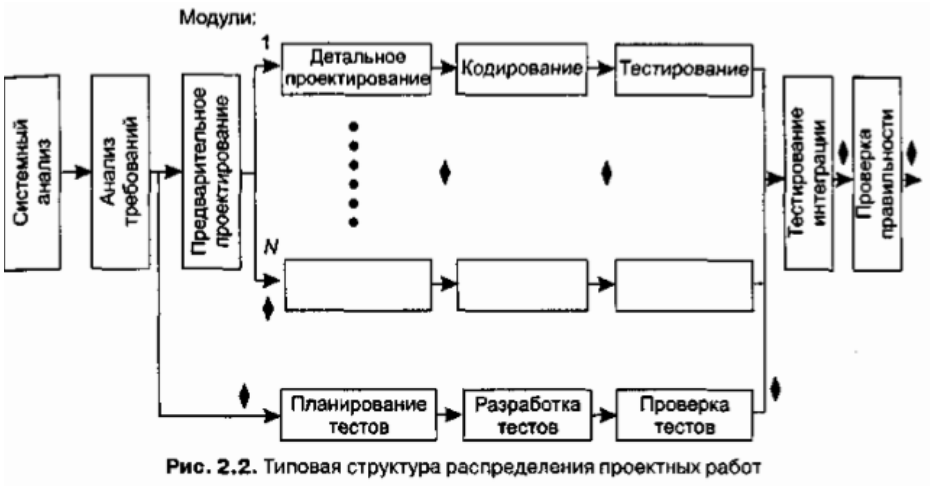


Рис. 2.2. Типовая структура распределения проектных работ

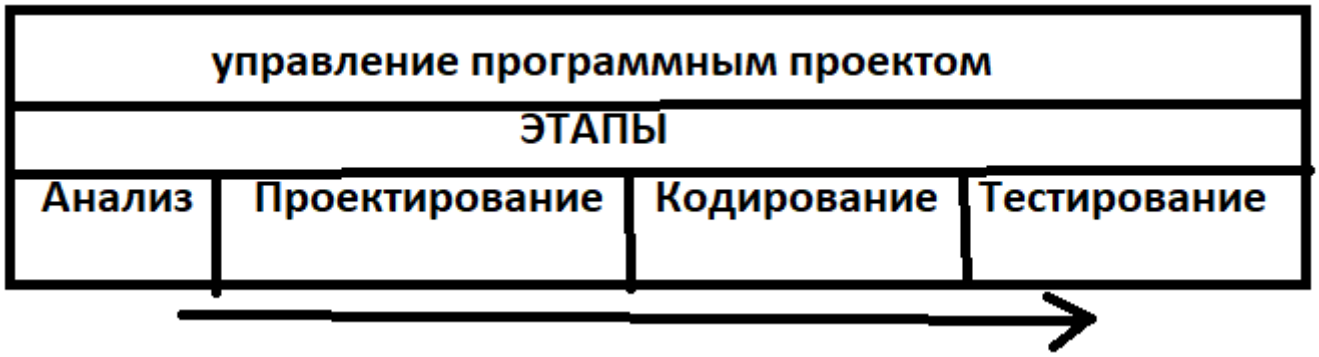


Рис. 1. управление в процессе конструирования ПО

Принцип управления программным проектом .На этом рисунке прямоугольник обозначает процесс конструирования, в нем выделены этапы, а вверху, над каждым из этапов, размещен слой деятельности «руководство программным проектом».

Для проведения успешного проекта нужно понять объем предстоящих работ, возможный риск, требуемые ресурсы, предстоящие задачи, прокладываемые вехи, необходимые усилия (стоимость), план работ, которому желательно следовать.

Руководство программным проектом обеспечивает такое понимание.

Оно начинается перед технической работой, продолжается по мере развития ПО от идеи к реальности и достигает наивысшего уровня к концу работ

Этапы управления ПП

1. **Начало проекта**

Перед планированием проекта следует: ❱ установить цели и проблемную область проекта; ❱ обсудить альтернативные решения; ❱ выявить технические и управленческие ограничения.

2. **Измерения, меры и метрики**

Благодаря измерениям мы понимаем с чем мы имеем дело, что за продукт ,как его можно будет улучшить.

Мера нам помогает дать оценку (показатель)точное значение, на сколько мы сможем улучшить наш продукт

Пример : на 25% повысится скорость работы сайта

Метрика-это тоже самое, что и мера, она нам помогает дать точное значении эффективности нашей работы и или планов для разработки.

Измерения помогают понять как процесс разработки продукта, так и сам продукт. Измерения процесса производятся в целях его улучшения, для повышения его качества.

В результате измерения определяется мера — количественная характеристика какого-либо свойства объекта.

Метрика определена как мера степени обладания свойством, имеющая числовое значение. В программной инженерии понятия мера и метрика очень часто рассматривают как синонимы.

3. Процесс оценки

При планировании программного проекта надо оценить людские ресурсы (в человеко-месяцах), продолжительность (в календарных датах), стоимость (в тысячах долларов). Обычно исходят из прошлого опыта.

Если новый проект по размеру и функциям похож на предыдущий проект, вполне вероятно, что потребуются такие же ресурсы, время и деньги.

4. Анализ риска

На этой стадии исследуется область неопределенности, имеющаяся в наличии перед созданием программного продукта. Анализируется ее влияние на проект. Нет ли скрытых от внимания трудных технических проблем? Не станут ли изменения, проявившиеся в ходе проектирования, причиной недопустимого отставания по срокам? В результате принимается решение — выполнять проект или нет.

Анализируются все риски при разработки и принимается решение стоит вообще начинать проект разрабатывать или нет

5. Планирование

Определяется набор проектных задач. Устанавливаются связи между задачами, оценивается сложность каждой задачи. *Определяются людские и другие ресурсы*. *Создается сетевой график задач, проводится его временная разметка*.

6. Трассировка и контроль

Каждая задача, помеченная в плане, отслеживается руководителем проекта. Временная метка, к которой привязано подведение промежуточных итогов. В результате повторного планирования: ❱ могут быть перераспределены ресурсы; ❱ могут быть реорганизованы задачи; ❱ могут быть пересмотрены выходные обязательства.

тим-лмд смотри, насколько разработчики загружены, кто опережает план, кто отстает и перенаправляет кадры или увеличивает временные рамки разработки.

7. Планирование проектных задач

Основной задачей при планировании является определение (структуры распределения работ*). Первыми выполняются задачи системного анализа и анализа требований*. Они закладывают фундамент для последующих параллельных задач.

Системный анализ проводится с целью:

1) выяснения потребностей заказчика;

2) оценки выполнимости системы;

3) выполнения экономического и технического анализа;

4) распределения функций по элементам компьютерной системы (аппаратуре, программам, людям, базам данных и т. д.);

5) определения стоимости и ограничений планирования;

6) создания системной спецификации.

В системной спецификации описываются функции, характеристики системы, ограничения разработки, входная и выходная информация.

Анализ требований дает возможность:

1*) определить функции и характеристики программного продукта*;

2) *обозначить интерфейс* продукта с другими системными элементами;

3*) определить проектные ограничения* программного продукта;

4) построить модели: процесса, данных, режимов функционирования продукта;

5) создать такие формы представления информации и функций системы, которые можно использовать в ходе проектирования.

Контрольные вопросы:

59.расскажите сжатую классификацию проектов
60.расскажите развернутую классификацию проектов
61.опишите принцип работы управления ПП
62.опишите каждый этап управления ПП
63.нарисуйте и опишите типовую структуру проектных работ

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами

17. https://habr.com/post/189626/

18. https://4brain.ru/blog/%D0%BC%D0%BE%D0%B7%D0%B3%D0%BE%D0%B2%D0%BE%D0%B9-%D1%88%D1%82%D1%83%D1%80%D0%BC/
19. http://studbooks.net/1951034/ekonomika/klassifikatsiya_proektov
20. http://studbooks.net/15236/ekonomika/metody_ekspertnyh_otsenok

ЛЕКЦИЯ 25-26

Темы:

25. Назначение АИС. Жизненный цикл АИС. Различие между АИС, АС и ИС.
26. Сопровождение и поддержки ПО

Цель: изучить основных понятий сопровождения. Понять два различных похода к сопровождению ПО. Научиться отличать ПО от АИС

Автоматизированная информационная система (АИС)- совокупность программно-аппаратных средств, предназначенных для автоматизации деятельности, связанной с хранением, передачей и обработкой информации.

АИС являются, с одной стороны, разновидностью информационных систем (ИС) система, которая обрабатывает информацию, с другой — автоматизированных систем (АС) это любая система в которой присутствует автоматизация любого процесса.

АИС может быть определена как комплекс автоматизированных информационных технологий, предназначенных для информационного обслуживания – организованного непрерывного технологического процесса подготовки и выдачи потребителям научной, управленческой и др. информации, используемой для принятия решений, в соответствии с нуждами для поддержания эффективной деятельности.

Пример АИС Автоматизированная информационная система: являются банковские системы, автоматизированные системы управления предприятиями, там где по мимо сбора и хранения инфы используют и автоматизацию внутренних процессов.

Пример ИС информационных систем**-** любой одностраничный сайт, сайт визитка, портфолио**.**

Пример АС автоматизированных систем**-**Нахождение интеграла, расчет любого алгоритма, где раньше выполнялись расчеты в ручную.

Назначение АИС

Основной причиной создания и развития АИС является необходимость *ведения учёта информации о состоянии и динамике объекта*, которому посвящена система. На основании информационной картины, создаваемой системой, руководители различного звена могут принимать решения об управляющих воздействиях с целью решения текущих проблем.

Учётные данные системы могут быть подвергнуты автоматической обработке для последующего тактического и стратегического анализа с целью принятия управленческих решений большего горизонта действия.

Плюсы от использования АИС:

1. повышение производительности работы персонала;

2. улучшение качества обслуживания клиентов;
3. снижение трудоемкости и напряженности труда персонала;
4. снижение количества ошибок в его действиях.

Жизненный цикл автоматизированных информационных систем (ЖЦ АИС) это период создания и использования ИС, начиная с момента возникновения потребности в ИС и заканчивая моментом полного ее выхода из эксплуатации.

Стадии жизненного цикла информационной системы:

1)Предпроектное обследование:

1. ·сбор материалов для проектирования, при этом выделяют формулирование требований, с изучения объекта автоматизации, даются предварительные выводы пред проектного варианта ИС;
2. анализ материалов и разработка документации, обязательно дается технико экономическое обоснование с техническим заданием на проектирование ИС.

2) Проектирование:

2.1 предварительное проектирование;

1. · выбор проектных решений по аспектам разработки ИС;
2. · описание реальных компонент ИС;
3. · оформление и утверждение технического проекта (ТП).

2.2 детальное проектирование:

1. · выбор или разработка математических методов или алгоритмов программ;
2. · корректировка структур БД;
3. · создание документации на доставку и установку программных продуктов;
4. · выбор комплекса технических средств с документацией на ее установку.

2.3 разработка техно-рабочего проекта ИС (ТРП).

2.4 разработка методологии реализации функций управления с помощью ИС и описанием регламента действий аппарата управления.

3) Разработка ИС:

1. · получение и установка технических и программных средств;
2. · тестирование и доводка программного комплекса;
3. · разработка инструкций по эксплуатации программно-технических средств.

4) Ввод ИС в эксплуатацию:

1. · ввод технических средств;
2. · ввод программных средств;

3. · обучение и сертификация персонала;
4. · опытная эксплуатация;
5. · сдача и подписание актов приемки-сдачи работ.

5) Эксплуатация ИС:

1. · повседневная эксплуатация;
2. · общее сопровождение всего проекта.

После того ,как мы сдали наше ПО в эксплуатацию идет сопровождение и поддержка

Сопровождение и поддержка ПО.

Имеются две разных точки зрения на границы применимости терминов "сопровождение ПО" и "поддержка ПО".

1)техническое Сопровождение ПО осуществляется сопроводителем. Сопроводителем может быть внешняя организация или же сама та организация (ее отдел, отдельный сотрудник)

Услуга, направленная на устранение сбоев в работе программно-аппаратного обеспечения, его обслуживание и оптимизацию работы функционала штатными средствами.

Основное отличие услуги *сопровождения* от технической *поддержки* состоит в том, что сопровождается система в целом, а не отдельный экземпляр продукта.

Ключевой особенностью услуги является то, что специалисты, участвующие в процессе сопровождения, не только консультируют и решают технические вопросы, связанные с эксплуатацией системы , но и решают проблемы «на стыке» пограничных систем.

Услуга позволяет выбрать, на каких условиях будет производиться сопровождение, параметры и метрики предоставления услуги.

2)техническая Поддержка осуществляется исключительно сотрудниками отдела той организации, которая использует ПО в своей работе (эта организация называется "заказчик" ISO/IEC 14764:99). Это менее квалифицированные специалисты, чем сопроводители, а потому они не выполняют полностью тех работ, которые предусмотрены ISO/IEC 14764:99.

поддержка, доработка, исправление ошибок мясными разработчиками, они добовляют модули не влияющие на все ПО.

Услуга технической поддержки направлена на оказание помощи в решении вопросов по установке, настройке и эксплуатации продуктов, а также на получение информационной поддержки по приобретенным продуктам.

Техническая поддержка может оказываться как в гарантийный, так и в послегарантийный период.

Обязательным условием возможности оказания данной услуги считается приобретение у производителя продукта подписки на обновления продукта, поскольку рекомендацией при решении проблемы достаточно часто бывает установка обновленной версии продукта, либо патча.

Например, сотрудники отдела поддержки не выполняют работы по обнаружению и корректировке скрытых ошибок для предотвращения явного проявления этих ошибок.

Контрольные вопросы:

64. что такое сопровождение ПО?
65. В чем отличие АИС от ИС и АС?
66. Что такое ИС?
67. Назначения АИС?
68. Перечислите стадии жизненного цикла АИС
69. Опишите стадии ЖЦАИС
70. В чем отличие сопровождения ПО от поддержки ПО?
71. Какие услуги предоставляет тех поддержка?
72. Какие услуги предоставляет техническое сопровождение?

Список использованных источников:

1. Гагарина, Л.Г. Разработка и эксплуатация автоматизированных информационных систем: учебное пособие для студ. учрежд. СПО/
2. Технологии разработки программного обеспечения С.А. Орлов
3. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
4. Project Management For Dummies / Управление проектами для "чайников"
5. Л. Н. Боронина З. В. Сенук основы управления проектами
6. http://upr.ru/article/infrastruktura-it/SOPROVOZHDENIE_PROGRAMMNYH_SISTEM.html

ЛЕКЦИЯ 27

Тема: Модели жизненного цикла информационной системы. Каскадная модель ЖЦ АИС. Процесс разработки АИС по каскадной схеме

Цель: изучить каскадную модель АИС. Понять принцип работы, в каких случаях ее применяют на практике. Понять достоинства и недостатки данной стратегии

Модель жизненного цикла АИС— это структура, описывающая процессы, действия и задачи, которые осуществляются и ходе разработки, функционирования и сопровождения в течение всего жизненного цикла системы.

• По возможности нужно выбирать стратегию с наименьшим количеством всевозможных рисков, неопределенных и неконтролируемых событий.

Запомните, что нужно использовать не те методы, которые могут работать, а которые действительно будут работать.

• Если проект связан с рисками, подготовьте несколько запасных стратегий на случай, когда первоначальные методы себя не оправдают

Выбор модели жизненного цикла зависит от специфики, масштаба, сложности проекта и набора условий, в которых АИС создается и функционирует.

Модель ЖЦ АИС включает:

• стадии;

• результаты выполнения работ на каждой стадии;

• ключевые события или точки завершения работ и принятия решений.

В соответствии с известными моделями ЖЦ ПО определяют модели ЖЦ АИС — каскадную, итерационную, спиральную.

Модели жизненного цикла информационной системы:

1. **каскадная модель** 1970-80гг
2. **инкрементная модель**
3. **спиральная модель** 1986г
4. **Компонентно-ориентированная модель**

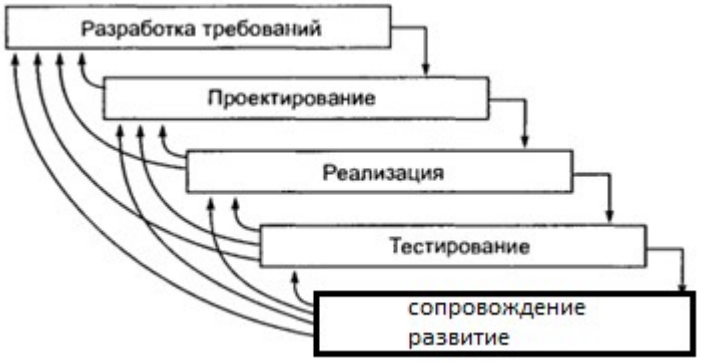
В модели водопада, называемой также "каскадная модель жизненного цикла" или "каскадная модель жизненного цикла с обратными связями", сопровождение ПО выделяется в отдельную фазу жизненного цикла.

Класичебский жизненный цикл ПО (каскадная модель)

Каскадная стратегия (однократный проход, водопадная или классическая модель) подразумевает линейную последовательность выполнения стадий создания информационной системы (рис.3.1). Другими словами, переход с одной стадии на следующую происходит только после того, как будет полностью завершена работа на текущей.

Данная модель применяется при разработке информационных систем, для которых в самом начале разработки можно достаточно точно и полно сформулировать все требования.

Рис.3.1. Каскадная стратегия



На первом этапе проводится исследование проблемы, которая должна быть решена, четко формулируются все требования заказчика. Результатом, получаемым на данном этапе, является техническое задание (задание на разработку), согласованное со всеми заинтересованными сторонами.

Ставим цели, задачи, исследуем предметную область, тесно общаемся с клиентом, что должен выполнять ПП

На втором этапе разрабатываются проектные решения, удовлетворяющие требованиям, сформулированным в техническом задании. Результатом, получаемым на данном этапе, является комплект проектной документации, содержащей необходимые данные для реализации проекта.

Составляем техническую документацию по проекту исходя из поставленных целей и задач

Третий этап — реализация проекта. Здесь осуществляется разработка программного обеспечения (кодирование) в соответствии с проектными решениями, полученными на

втором этапе. Методы, используемые для реализации, не имеют принципиального значения. Результатом, получаемым на данном этапе, является готовый программный продукт.

На четвертом этапе проводится проверка полученного программного обеспечения на предмет соответствия требованиям, заявленным в техническом задании. Опытная эксплуатация позволяет выявить различного рода скрытые недостатки, проявляющиеся в реальных условиях работы информационной системы.

Тестирование блоков отдельных частей ПО

Пятый этап — сдача готового проекта. Главная задача этого этапа — убедить заказчика, что все его требования выполнены в полной мере.

Достоинства модели:

- на каждой стадии формируется законченный набор документации, программного и аппаратного обеспечения, отвечающий критериям полноты и согласованности;

- выполняемые в четкой последовательности стадии позволяют уверенно планировать сроки выполнения работ и соответствующие ресурсы (денежные, материальные и людские).

Недостатки модели:

1. - реальный процесс разработки информационной системы редко полностью укладывается в такую жесткую схему. Особенно это относится к разработке нетиповых и новаторских систем;

2. основана на точной формулировке исходных требований к информационной системе. Реально в начале проекта требования заказчика определены лишь частично;

3. основной недостаток – результаты разработки доступны заказчику только в конце проекта. В случае неточного изложения требований или их изменения в течение длительного периода создания ИС заказчик получает систему, не удовлетворяющую его потребностям.

4. существенная задержка в получении результатов;

Задержка в получении результатов проявляется в том, что при последовательном подходе к разработке согласование результатов производится только после завершения очередного этапа работ.

В результате может оказаться, что разрабатываемая АИС не соответствует требованиям, и такие несоответствия могут возникать на любом этапе разработки; кроме того, ошибки могут непреднамеренно вноситься и проектировщиками-аналитиками, и программистами, так как они не обязаны хорошо разбираться в тех предметных областях, для которых разрабатывается АИС.

5. ошибки и недоработки на любом из этапов проявляются, как правило, на последующих этапах работ, что приводит к необходимости возврата;

Возврат на более ранние стадии. ошибки, допущенные на более ранних этапах, обнаруживаются только на последующих стадиях. В результате проект возвращается на предыдущий этап, перерабатывается и только затем передается в последующую работу.

Это может послужить причиной срыва графика и усложнения взаимоотношений между группами разработчиков, выполняющих отдельные этапы.

6. сложность параллельного ведения работ по проекту;

Чем сильнее взаимосвязь отдельных частей проекта, тем чаще и тщательнее должна выполняться синхронизация, тем сильнее зависят друг от друга группы разработчиков. В результате преимущества параллельного проведения работ просто теряются; отсутствие параллелизма негативно сказывается и на организации работы всего коллектива.

7. чрезмерная информационная перенасыщенность каждого из этапов;

Проблема информационной перенасыщенности возникает вследствие сильной зависимости между различными группами разработчиков. Дело в том, что при внесении изменений в одну из частей проекта, необходимо оповещать тех разработчиков, которые использовали (могли использовать) ее в своей работе.

При наличии большого числа взаимосвязанных подсистем синхронизация внутренней документации становится отдельной важнейшей задачей: разработчики должны постоянно знакомятся с изменениями и оценивать, как скажутся эти изменения на полученных результатах.

8. сложность управления проектом;

Сложность управления проектом в основном обусловлена строгой последовательностью стадий разработки и наличием сложных взаимосвязей между различными частями проекта. Регламентированная последовательность работ приводит к тому, что одни группы разработчиков должны ожидать результатов работы других команд, поэтому требуется административное вмешательство для согласования сроков и состава передаваемой документации.

9. высокий уровень риска и ненадежность инвестиций.

Высокий уровень риска. Чем сложнее проект, тем дольше длится каждый этап разработки и тем сложнее взаимосвязи между отдельными частями проекта, количество которых также увеличивается. Причем результаты разработки можно реально увидеть и оценить лишь на этапе тестирования, т. е. после завершения анализа, проектирования и разработки — этапов, выполнение которых требует значительного времени и средств.

10. Возникновение конфликтов между разработчиками.

Возврат части проекта на ранние стадии обусловлен писком виновных в ошибке. И как следствие ценится не тот руководитель, который имеет высокую квалификацию и большой опыт, а тот, кто может отстоять своих подчиненных.

Контрольные вопросы:

73. Что такое модель ЖЦ АИС?
74. Отчего зависит выбор модели ЖЦ?
75. Что включает в себя модель ЖЦ?
76. Что такое каскадная стратегия?
77. Опишите этапы каскадной стратегии
78. На какие две фазы делится каскадная стратегия?
79. Назовите достоинства и недостатки каскадной стратегии

Список использованных источников:

- Гагарина, Л.Г. Разработка и эксплуатация автоматизированных информационных систем: учебное пособие для студ. учрежд. СПО/
- Технологии разработки программного обеспечения С.А. Орлов
- Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
- https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema3#p32
- http://www.computer-museum.ru/books/n_collection/models.htm

ЛЕКЦИЯ 28-29

Темы:

28Модели ЖЦ информационной системы. Инкрементная стратегия.

29 Процесс разработки АИС по Rad схеме

Цель: изучить инкрементной модель АИС. Понять принцип работы, в каких случаях ее применяют на практике. Понять достоинства и недостатки данной стратегии

Модели жизненного цикла информационной системы:

5. **каскадная модель** 1970-80гг
6. **инкрементная модель,RAD**
7. **спиральная модель** 1986г
8. **Компонентно-ориентированная модель**

Инкрементная стратегия (англ. increment – увеличение, приращение) подразумевает разработку информационной системы с линейной последовательностью стадий, но в несколько инкрементов (версий), т. е. с запланированным улучшением продукта.

Инкрементная модель является классическим примером инкрементной стратегии разработки ПО, объединяя элементы последовательной водопадной модели с итерационной философией макетирования.

Данная стратегия в реальной жизни выдает проект заказчику от набросками, одну ф-ю написали вылили в продакшин.

методология быстрой разработки приложений инкрементной модели содержит 3 элемента:

1. небольшую команду программистов (от 2 до 10 человек);
2. короткий, но тщательно проработанный производственный график (от 2 до 6 мес.);
3. повторяющийся цикл Т.е. выпускаются версии программы по результату общения с клиентом, каждая версия усовершенствуется

Каждая итерация обеспечивает прохождение всех фаз проекта, обеспечивая инкремент (прирост) функциональности. Однако итерация, как правило, недостаточна для выпуска

новой версии продукта. По окончании итерации команда разработчиков оценивает и выбирает приоритеты разработки.

Главное рабочий продукт, а не система письменной документации. Т.е. очень скудная тех.документация по проектированию и использованию

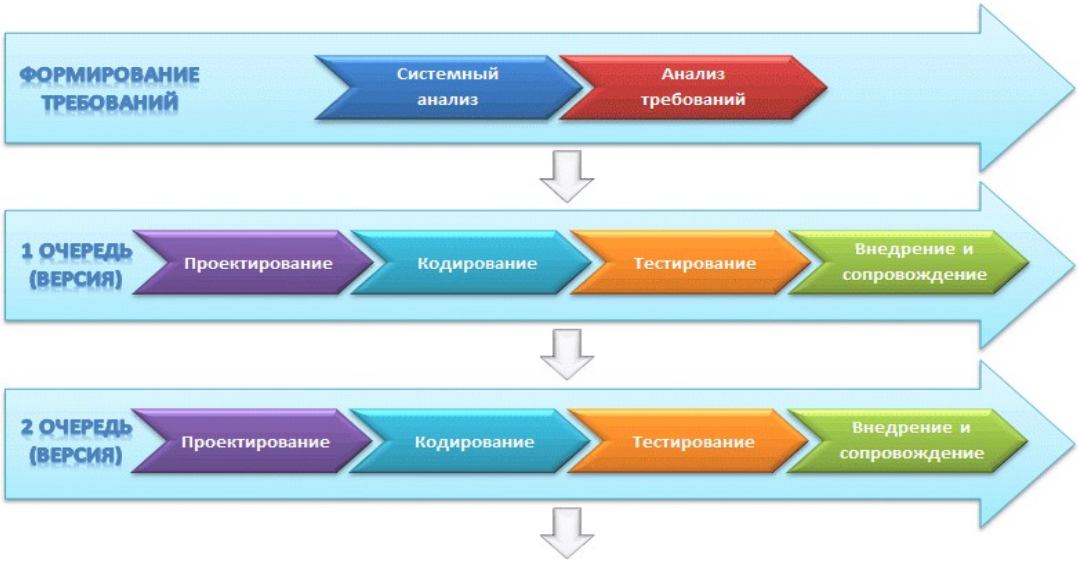


Рис.3.2. Инкрементная стратегия

Принцип работы инкрементной стратегии

В начале работы над проектом определяются все основные требования к системе, после чего выполняется ее разработка в виде последовательности версий. При этом каждая версия является законченным и работоспособным продуктом. Первая версия реализует часть запланированных возможностей, следующая версия реализует дополнительные возможности и т. д., пока не будет получена полная система.

Данная модель жизненного цикла характерна при разработке комплексных систем, для которых имеется четкое видение (как со стороны заказчика, так и со стороны разработчика) того, что собой должен представлять конечный результат (информационная система).

Разработка версиями ведется в силу разного рода причин:

1. -отсутствия у заказчика возможности сразу профинансировать весь дорогостоящий проект;

не достаточно денежных средств

2. - отсутствия у разработчика необходимых ресурсов для реализации сложного проекта в сжатые сроки;

маленький коллектив разработчиков

3. - требований поэтапного внедрения и освоения продукта конечными пользователями. Внедрение всей системы сразу может вызвать у ее пользователей неприятие и только «затормозить» процесс перехода на новые технологии.

Выдают проект частями, так как клиента может не устроить версия ПП

Достоинства и недостатки этой стратегии такие же, как и у классической. Но в отличие от классической стратегии заказчик может раньше увидеть результаты. Уже по результатам разработки и внедрения первой версии он может незначительно изменить требования к разработке, отказаться от нее или предложить разработку более совершенного продукта с заключением нового договора.

Достоинства модели:

- на каждой стадии формируется законченный набор документации;
- выполняемые в четкой последовательности стадии позволяют уверенно планировать сроки выполнения работ и соответствующие ресурсы (денежные, материальные и людские).

Недостатки модели:

1. требования заказчика к информационной системе в начале проекта определены лишь частично;
2. существенная задержка в получении результатов;
3. Возникновение конфликтов между разработчиками.
4. высокий уровень риска и ненадежность инвестиций.
5. сложность управления проектом;
6. чрезмерная информационная перенасыщенность каждого из этапов;
7. сложность параллельного ведения работ по проекту
8. ошибки и недоработки на любом из этапов проявляются, как правило, на последующих этапах работ, что приводит к необходимости возврата;

RAD — Rapid Application Development (быстрая разработка приложений) разновидность инкрементной модели или макетирование

RAD — Rapid Application Development (быстрая разработка приложений), которая стала основой технологий создания и развертывания программных продуктов.

Данная модель позволяет быстро, качественно и с контролем рисков создать и выпустить проект, отличается от модели инкрементной, тем, что возможен возврат и доработка версии.

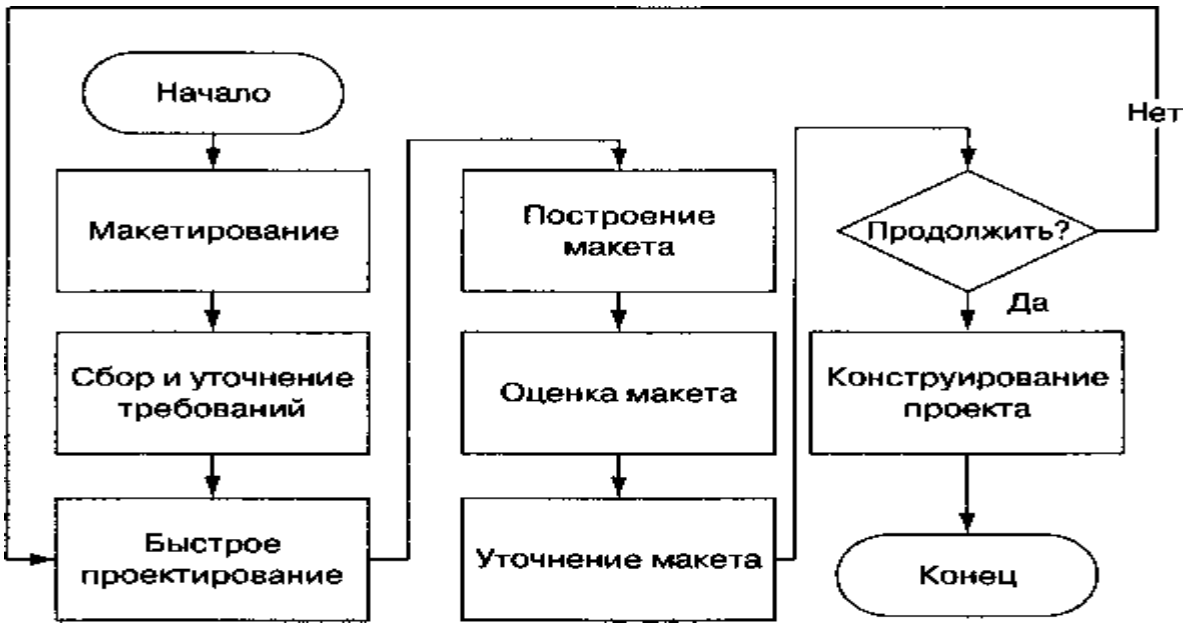


Схема выполнения проектных работ RAD

Макетирование основывается на многократном повторении итераций, в которых участвуют заказчик и разработчик, и начинается со сбора и уточнения требований к создаваемому программному продукту.

Эта модель предусматривает:

1. инструментальную поддержку процесса разработки, минимизацию времени и трудозатрат;

2. использование прототипа для уточнения требований заказчика;

3. цикличность разработки — каждая новая версия продукта основывается на оценке результата работы предыдущей версии заказчиком;

4. постепенное расширение функциональности;

5. распределение ролей в команде разработчика, возможность их совмещения;

6. управление проектом создания программного продукта.

Недостатки Rad модели:

1. Скудный дизайн ПП
2. Отсутствие масштабируемости используется маленькими и средними проектными командами.
3. Нужно выбирать между гибкостью проекта или контролем качества

Случая выбора инкрементной, RAD разработки:

1. для него важна скорость и простота разработки
2. четко определены приоритетные направления разработки проекта
3. разработать приложение нужно в сжатые сроки
4. проект выполняется в условиях ограниченного бюджета
5. главный критерий — интерфейс пользователя
6. есть возможность разбить проект на функциональные компоненты

Контрольные вопросы:

80. что такое экстремальная разработка?
81. Чем отличается инкрементная модель от Rad модели?
82. В каких случаях используют инкрементную и Rad разработку?
83. Принцип работы инкрементной модели
84. Достоинства и недостатки инкрементного проектирования ПП
85. Схема разработки по Rad модели
86. Для чего используют версии проекта?
87. Чем отличается от модели водопада?

Список использованных источников:

- Гагарина, Л.Г. Разработка и эксплуатация автоматизированных информационных систем: учебное пособие для студ. учрежд. СПО/
- Технологии разработки программного обеспечения С.А. Орлов
- Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
- Project Management For Dummies / Управление проектами для "чайников"
- Л. Н. Боронина З. В. Сенук основы управления проектами
- https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema3#p32
- http://www.computer-museum.ru/books/n_collection/models.htm
- https://worksection.com/blog/rapid-application-development.html

ЛЕКЦИЯ 30-31

Темы:

30.Спиральная стратегия. Процесс разработки, поддержка АИС по спиральной схеме

31. Выбор модели под конкретные параметры. Сравнение всех моделей ЖЦ

Цель: изучить спиральную модель АИС. Понять принцип работы, в каких случаях ее применяют на практике. Понять достоинства и недостатки данной стратегии

Модели жизненного цикла информационной системы:

9. **каскадная модель** 1970-80гг
10. **инкрементная модель,RAD**
11. **спиральная модель** 1986г
12. **Компонентно-ориентированная модель**

Спиральная стратегия (эволюционная или итерационная модель)подразумевает разработку в виде последовательности версий, но в начале проекта определены не все требования. Требования уточняются в результате разработки версий.

Спиральная модель представляет шаблон процесса разработки ПО, который сочетает идеи итеративной и каскадной моделей.

Суть метода, что весь процесс создания конечного продукта представлен в виде условной плоскости, разбитой на 4 сектора, каждый из которых представляет отдельные этапы его разработки: определение целей, оценка рисков, разработка и тестирование, планирование новой итерации.

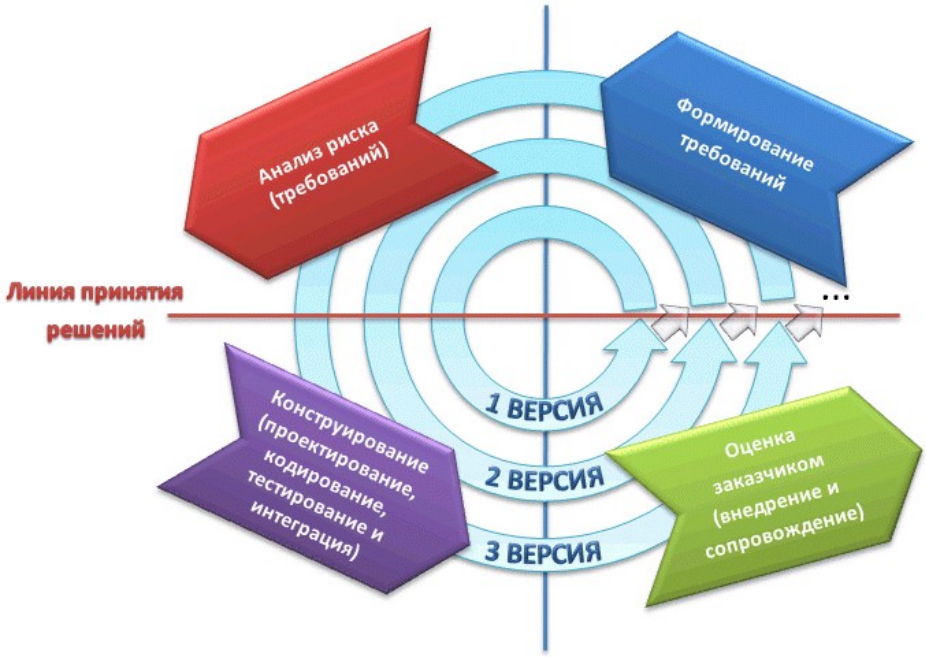
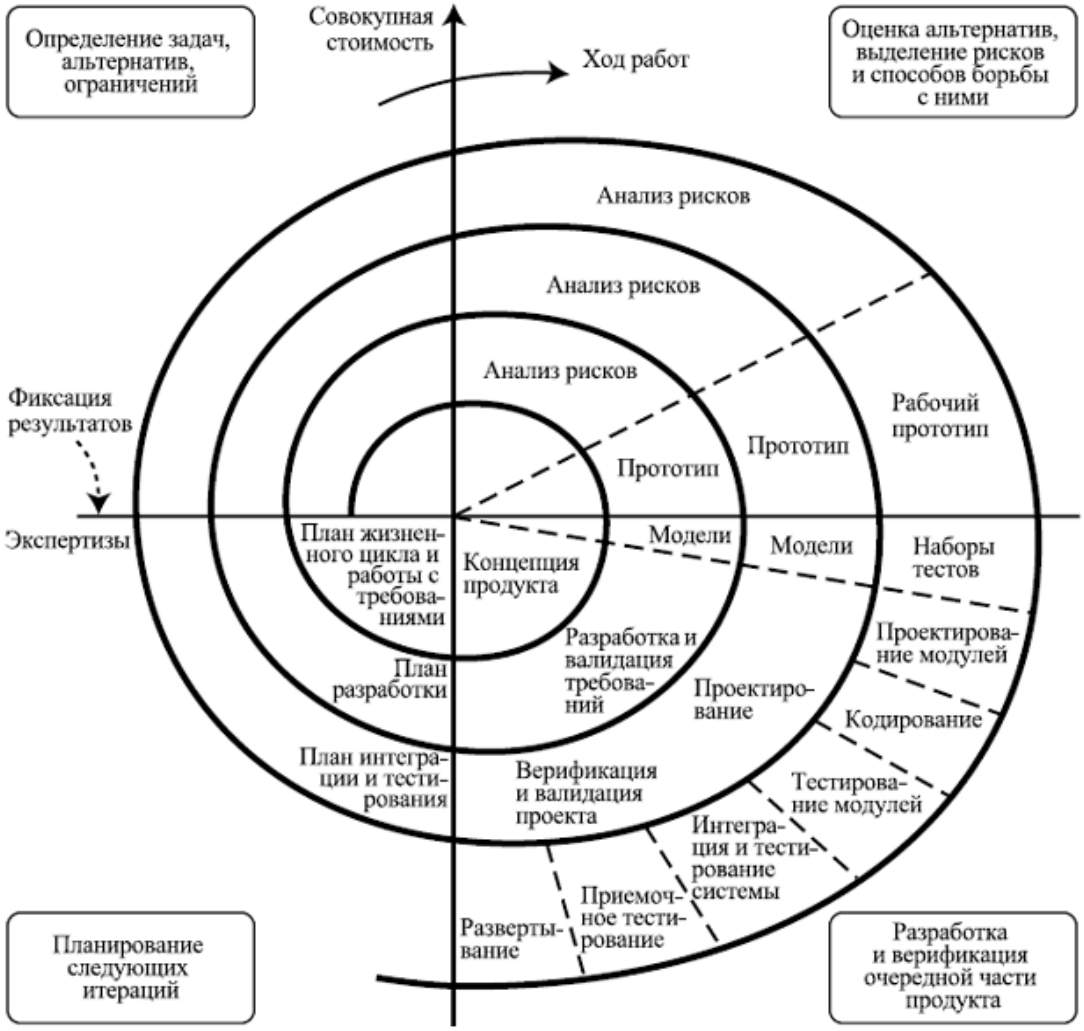


Рис. 3.3. Спиральная стратегия



1. Анализ рисков (**анализ** вероятности того, что произойдут определенные нежелательные события и отрицательно повлияют на достижение целей проекта)
2. валидация требований (подтверждение на основе представления объективных *свидетельств* того, что требования, предназначенные для конкретного использования или применения, выполнены)
3. верификация требований (подтверждение на основе представления объективных свидетельств того, что установленные требования были выполнены

Главная особенность спиральной модели – концентрация на возможных рисках. Для их оценки даже выделена соответствующая стадия.

Основные типы рисков, которые могут возникнуть в процессе разработки ПО:

— Нереалистичный бюджет и сроки;

— Дефицит специалистов;

— Частые изменения требований;

— Чрезмерная оптимизация;

— Низкая производительность системы;

— Несоответствие уровня квалификации специалистов разных отделов.

Принцип работы

Данная модель жизненного цикла характерна при разработке новаторских (нетиповых) систем. В начале работы над проектом у заказчика и разработчика нет четкого видения итогового продукта (требования не могут быть четко определены) или стопроцентной уверенности в успешной реализации проекта (риски очень велики). В связи с этим принимается решение разработки системы по частям с возможностью изменения требований или отказа от ее дальнейшего развития. Как видно из рис.3.3, развитие проекта может быть завершено не только после стадии внедрения, но и после стадии анализа риска.

Достоинства спиральной модели:

1. улучшенный анализ рисков;
2. хорошая документация процесса разработки;
3. гибкость – возможность внесения изменений и добавления новой функциональности даже на относительно поздних этапах;
4. раннее создание рабочих прототипов.

Недостатки спиральной модели

1. может быть достаточно дорогой в использовании;
2. управление рисками требует привлечения высококлассных специалистов;
3. успех процесса в большой степени зависит от стадии анализа рисков;
4. не подходит для небольших проектов.

Когда использовать спиральную модель:

1. когда важен анализ рисков и затрат;
2. крупные долгосрочные проекты с отсутствием четких требований или вероятностью их динамического изменения;
3. при разработке новой линейки продуктов.

Каждая из моделей имеет свои достоинства и недостатки, а также сферы применения в зависимости от специфики разрабатываемой системы, возможностей заказчика и разработчика

Сравнение моделей жизненного цикла

Характеристика проекта	Модель (стратегия)		
	Каскадная	Инкрементная	Спиральная
Новизна разработки и обеспеченность ресурсами	Типовой. Хорошо проработаны технология и методы решения задачи		Нетиповой (новаторский). Нетрадиционный для разработчика
	Ресурсов заказчика и разработчика хватает для	Ресурсов заказчика или разработчика не хватает для реализации проекта в	

	реализации проекта в сжатые сроки	сжатые сроки	
Масштаб проекта	Малые и средние проекты	Средние и крупные проекты	Любые проекты
Сроки выполнения проекта	До года	До нескольких лет. Разработка одной версии может занимать срок от нескольких недель до года	
Заключение отдельных договоров на отдельные версии	Заключается один договор. Версия и есть итоговый результат проекта	На отдельную версию или несколько последовательных версий обычно заключается отдельный договор	
Определение основных требований в начале проекта	Да	Да	Нет
Изменение требований по мере развития проекта	Нет	Незначительное	Да
Разработка итерациями (версиями)	Нет	Да	Да
Распространение промежуточного ПО	Нет	Может быть	Да

Контрольные вопросы:

88. суть спиральной модели
89. что такое спиральная модель ПП?
90. Достоинства и недостатки спиральной модели
91. Когда используют спиральную модель
92. Принцип работы спиральной модели
93. На сколько секторов разделена спиральная модель, скажите основные ее возможности
94. Проведите сравнительный анализ всех моделей разработки ПП
95.

Список использованных источников:

- Гагарина, Л.Г. Разработка и эксплуатация автоматизированных информационных систем: учебное пособие для студ. учрежд. СПО/
- Технологии разработки программного обеспечения С.А. Орлов
- Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
- Project Management For Dummies / Управление проектами для "чайников"
- Л. Н. Боронина З. В. Сенук основы управления проектами
- https://sites.google.com/site/anisimovkhv/learning/pris/lecture/tema3#p32
- http://www.computer-museum.ru/books/n_collection/models.htm
- https://worksection.com/blog/rapid-application-development.html
- https://qalight.com.ua/baza-znaniy/spiralnaya-model-spiral-model/

ЛЕКЦИЯ 32

Тема: Компонентно-ориентированная модель. Экстремальное программирование XP.
Цель: изучить стратегии проектирования программного продукта
Компонентно-ориентированная модель является развитием спиральной модели.

Программные компоненты, созданные в реализованных программных проектах, хранятся в библиотеке. В новом программном проекте, исходя из требований заказчика, выявляются кандидаты в компоненты. Далее проверяется наличие этих кандидатов в библиотеке. Если они найдены, то компоненты извлекаются из библиотеки и используются повторно. В противном случае создаются новые компоненты, они применяются в проекте и включаются в библиотеку.

Рис. 7. Компонентно-ориентированная модель

Достоинства компонентно-ориентированной модели:

1. уменьшает на 30% время разработки программного продукта;
2. уменьшает стоимость программной разработки до 70%;
3. увеличивает в полтора раза производительность разработки.

ЭКСТРЕМАЛЬНОЕ ПРОГРАММИРОВАНИЕ XP.

Экстремальное программирование (eXtreme Programming, XP) — облегченный (подвижный) процесс (или методология).

Данная методология отличается от остальных моделей, что она применяется только для разработки ПО, оно не может использоваться в другом бизнесе, как КанБан или Scrum методологии.

В экстремальном программировании смысл в том, что цели задачи заказчика постоянно меняются и разработчикам нужно менять свои подходы и методы разработки, при этом не теряя качества и скорости разработки.

Цель методики XP — *справиться с постоянно меняющимися требованиями к программному продукту и повысить качество разработки.*

XP-процесс ориентирован на группы малого и среднего размера, строящие программное обеспечение в условиях неопределенных или быстро изменяющихся требований. *XP-группу образуют* до **10 сотрудников**, которые размещаются в одном помещении.

Основная идея XP — устранить высокую стоимость изменения, характерную для приложений с использованием объектов, паттернов* и реляционных баз данных.

XP-группа имеет дело с изменениями требований на всем протяжении итерационного цикла разработки, причем *цикл состоит из очень коротких итераций*.

Один виток (цикл) разработки короткий, чтобы можно было максимально подстраиваться под новые требования. Скорость итерации обеспечивается с помощью : кодирования, тестирования, общения с заказчиком и проектирования.

Четырьмя базовыми действиями в XP-цикле являются:

1. кодирование,
2. тестирование,
3. общения с заказчика
4. проектирование.

Динамизм обеспечивается с помощью четырех характеристик: непрерывной связи с заказчиком (и в пределах группы), простоты (всегда выбирается минимальное решение), быстрой обратной связи (с помощью модульного и функционального тестирования), смелости в проведении профилактики возможных проблем.

структуру «идеального» XP-процесса.

Основная структура заложена на итерации разработки ПО. Это наш цикл повторов определенных действий с достижение каких либо результатов.

А в реализацию входит 3 фазы итерации: исследование, блокировка и регулирование.

Основным структурным элементом процесса является XP-реализация, в которую многократно вкладывается базовый элемент — XP-***итерация***.

В состав XP-реализации и XP-итерации входят три фазы :

1. исследование,

2. блокировка,
3. регулирование.

Исследование — это поиск новых требований (историй, задач), которые должна выполнять система.

Блокировка — выбор для реализации конкретного подмножества из всех возможных требований (иными словами, планирование).

Какие результаты мы планируем получить, с чем сравнивать, те результаты, которые получаем в процессе разработки. Т.е. идеальные параметры, которые мы хотим достичь.

Регулирование — *проведение разработки*, воплощение плана в жизнь.

Преимущества экстремального программирования имеют смысл, когда команда полноценно использует хотя бы одну из практик XP.

1. *заказчик получает именно тот продукт, который ему нужен*, даже если в начале разработки сам точно не представляет его конечный вид

2. *команда быстро вносит изменения в код* и добавляет новую функциональность за счет простого дизайна кода, частого планирования и релизов

3. код всегда работает за счет постоянного тестирования и непрерывной интеграции

4. *команда легко поддерживает код*, т.к. он написан по единому стандарту и постоянно рефакторится

5. *быстрый темп разработки за счет парного программирования*, отсутствия переработок, присутствия заказчика в команде

6. *высокое качество кода*

7. *снижаются риски, связанные с разработкой*, т.к. ответственность за проект распределяется равномерно и уход/приход члена команды не разрушит процесс

8. затраты на разработку ниже, т.к. *команда ориентирована на код, а не на документацию и собрания*

Несмотря на все плюсы, XP не всегда работает и имеет ряд слабых мест. Итак, экстремальное программирование — **недостатки**:

1. *успех проекта зависит от вовлеченности заказчика*, которой не так просто добиться

2. *трудно предугадать затраты времени на проект*, т.к. в начале никто не знает полного списка требований

3. успех XP сильно зависит от уровня программистов, *методология работает только с senior специалистами*

4. менеджмент негативно относится к парному программированию, не понимая, почему он должен *оплачивать двух программистов вместо одного*

5. регулярные *встречи с программистами дорого обходятся заказчикам*

6. из-за недостатка структуры и документации *не подходит для крупных проектов*

Контрольные вопросы:

96. что такое рефакторинг?
97. Расскажите про компонентно ориентированную модель
98. Что такое XP модель?
99. В чём суть метода экстремального программирования?
100. Базовые действия XP ?
101. Цель методики XP?
102. Назовите и опишите достоинства и недостатки

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. https://finswin.com/projects/osnovnye/sroki-realizacii-proekta.html

ЛЕКЦИЯ 33

Тема: Основные методы применения практики в XP разработке
Цель: изучить стратегии проектирования программного продукта

13 ПРАКТИК ЭКСТРЕМАЛЬНОГО ПРОГРАММИРОВАНИЯ

XP — строго упорядоченный процесс.

Простые решения, имеющие высший приоритет, в отличие от проектных решений, которые пока не нужны, а могут (в условиях изменения требований и операционной среды) и вообще не понадобиться.

1. **Вся команда-** обязательно входит представитель заказчика. В нее обязательно входит представитель заказчика. Заказчик выдвигает требования к продукту и расставляет приоритеты в реализации функциональности. Со стороны исполнителей в команду входят разработчики и тестировщики, направляющий команду, и менеджер, который обеспечивает команду ресурсами.

Все участники проекта с применением XP работают как одна команда. Заказчики которые ставят задачи(что нужно выполнить) и исполнители, которые реализуют эти задачи с применением метода экстремального программирования.

2. **Игра планирования** — быстрое определение области действия следующей реализации путем объединения деловых приоритетов и технических оценок. Заказчик формирует область действия, приоритетность и сроки с точки зрения бизнеса, а разработчики оценивают и прослеживают продвижение (прогресс).

Планирование в XP проводят в два этапа — *планирование релиза и планирование итераций*. Если команда не успевает выполнить все задачи к дате релиза, то релиз не отодвигается, а режется часть функционала, наименее важная для заказчика т.е. просто сокращается часть ф-й.

3. **Маленькие релизы версий** (Small releases) — быстрый запуск в производство простой системы. Новые версии реализуются в очень коротком (двухнедельном) цикле.

В XP версии выпускаются часто, но с небольшим функционалом.

Во-первых, маленький объем функциональности легко тестировать и сохранять работоспособность всей системы.

Во-вторых, каждую итерацию заказчик получает часть функционала, несущую бизнес-ценность.

4. **Пользовательские тесты**- Заказчик сам определяет автоматизированные приемочные тесты, чтобы проверить работоспособность очередной функции продукта. Команда пишет эти тесты и использует их для тестирования готового кода.

Заказчик сам решает, что он хочет тестировать и как. Т.е. он может потребовать тесты в определяет объёме только по функционалу основных компонентов или только к пользовательскому интерфейсу. А уже разработчики реализуют эти тесты

5. **Коллективное владение кодом** — любой разработчик может улучшать любой код системы в любое время.

В XP любой разработчик может править любой кусок кода, т.к. код не закреплен за своим автором. Кодом владеет вся команда.

6. **Непрерывная интеграция кода**— система интегрируется и строится много раз в день, по мере завершения каждой задачи. Непрерывное регрессионное тестирование, то есть повторение предыдущих тестов, гарантирует, что изменения требований не приведут к регрессу функциональности.

Это значит, что новые части кода сразу же встраиваются в систему — команды XP заливают в репозиторий новый билд каждые несколько часов и чаще.

Во-первых, сразу видно, как последние изменения влияют на систему. Если новый кусок кода что-то сломал, то ошибку найти и исправить в разы проще, чем спустя неделю.

Во-вторых, команда всегда работает с последней версией системы.

7. **Стандарты кодирования** — должны выдерживаться правила, обеспечивающие одинаковое представление программного кода во всех частях программной системы. Можно создать свои стандарты или использовать готовые.

Когда кодом владеют все, важно принять единые стандарты оформления, чтобы код выглядел так, как будто он написан одним профессионалом. Можно выработать свои стандарты или принять готовые.

8. **Метафора системы** — вся разработка проводится на основе простой, общедоступной истории о том, как работает вся система.

Метафора системы — это ее сравнение с чем-то знакомым, чтобы сформировать у команды общее видение.

Обычно метафору системы продумывает тот, кто разрабатывает архитектуру и представляет систему целиком.

9. **Устойчивый темп** (40-часовая неделя) — как правило, работают не более 40 часов в неделю. Нельзя удваивать рабочую неделю за счет сверхурочных работ.

XP команды работают на максимуме продуктивности, сохраняя устойчивый темп.

При этом экстремальное программирование *негативно относится к переработкам* и пропагандирует 40-часовую рабочую неделю. Т.е. в день не более 8 часов

10. **Разработка основанная на Тестирование** — непрерывное написание тестов самими программистами для определенных модулей, до написания самого кода; функционал покрывается тестами на 100%. заказчики пишут тесты для демонстрации законченности функций.

Самая сложная часть. В XP тесты пишутся самими программистами, причем ДО написания кода, который нужно протестировать.

При таком подходе каждый кусок функционала будет покрыт тестами на 100%.

Когда пара программистов заливают код в репозиторий, сразу запускаются модульные тесты. И ВСЕ они должны сработать. Тогда разработчики будут уверены, что движутся в правильном направлении.

11. **Парное программирование** — весь код пишется двумя программистами, работающими на одном компьютере.

Представьте двух разработчиков за одним компьютером, работающих над одним куском функциональности продукта.

Из двух вариантов решения проблемы выбирается лучший, код оптимизируется сразу же, ошибки отлавливаются еще до их совершения.

В итоге имеем чистый код, в котором хорошо разбираются сразу двое разработчиков.

12. **Простой дизайн**— проектирование выполняется настолько просто, насколько это возможно в данный момент, не пытаемся предугадать будущий функционал.

делать только то, что нужно сейчас, не пытаясь угадать будущую функциональность.

13. **рефакторинг** — это процесс постоянного реконструирование дизайна системы, чтобы привести его в соответствие новым требованиям. Рефакторинг включает удаление дублей кода, повышение связности и снижение сопряжения.

Постоянное улучшение дизайна т.е. постоянные рефакторинги, поэтому дизайн кода всегда остается простым.

Простой дизайн и непрерывный рефакторинг дают синергетический эффект — когда код простой, его легко оптимизировать.

Контрольные вопросы:

103. что такое рефакторинг?
104. Что такое игра планирования?
105. Что такое стандарты кодирования?
106. Метафора системы?
107. Что подразумевается под устойчивым темпом?
108. Что подразумевает разработка основанная на тестировании?
109. Как происходит парное программирование?
110. **Что такое пользовательские тесты?**
111. **Что подразумевается под маленькими релизами версий программы?**
112. Практики экстремального программирования, опишите

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Искусство IT-проектирования Скотт Беркун
4. https://finswin.com/projects/osnovnye/sroki-realizacii-proekta.html

ЛЕКЦИЯ 34-37

Темы:

34Структурное и объектно-ориентированное программирование в проектировании программного обеспечения распределенных информационных систем.

35 Структурный подход к проектированию информационных систем подходов к проектированию программного обеспечения

36 Проектирование информационных систем на основе объектно-ориентированного подхода

37 Сопоставление и взаимосвязь структурного и объектно-ориентированного подходов

Цель: изучить основные методы проектирования.

Методики, используемые при проектировании, сначала программ, а затем и систем в целом, формировались в течение длительного промежутка времени.

Необходимость таких методик проявлялась при разработке сложных программных систем в условиях дефицита времени на разработку.

Большинство методик сначала были внутренними стандартами больших корпорации, а потом перешли на международный стандарт.

В основе наиболее известных методик проектирования ИС лежат два подхода:

1. **структурный**
2. **объектно-ориентированный.**

1. **Структурные методы анализа и проектирования** используют иерархические структуры для моделирования объекта исследования.

Структурное проектирование основано на алгоритмической декомпозиции, особое внимание в которой уделяется порядку происходящих событий.

Эти методы предназначены, в основном, для построения функциональных моделей и моделей данных разного уровня.

Как уже рассказывалось, есть главная цель и есть ее под разделы. Мы разделяем все части на под части.

2.Объектно-ориентированный подход основан на выделении агентов, которые являются либо субъектами действий, либо объектами действий. При объектно-

ориентированной декомпозиции каждый объект обладает своим собственным поведением и каждый из них моделирует некоторый объект реального мира.

Выделяются актеры, описываются их свойства, действия которые они выполняют и уже потом с их помощью моделируют взаимосвязь всех объектов.

вряд ли удастся спроектировать сложную систему одновременно двумя способами, но можно применить их последовательно.

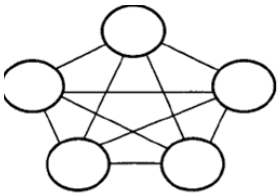
СТРУКТУРНЫЙ ПОДХОД

Структурный подход состоит в декомпозиции (разбиении) *системы на функциональные подсистемы, которые в свою очередь делятся на подфункции, подразделяемые на задачи*, и т.д. Процесс разбиения продолжается вплоть до конкретных процедур.

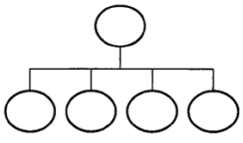
Все наиболее распространенные **структурные методы базируются на следующих принципах:**

1. - принцип разбиения сложной проблемы на множество меньших независимых задач, легких для понимания и решения;

Задачу разбиваем на не зависимые подзадачи которые можно выполнять параллельно или они не взаимосвязаны с друг другом логической цепочкой .

Вывести диалоговое окно приветствия, один разрабатывает структуру окна, другой рисует окно. 

2. - принцип организации составных частей в иерархические структуры.

Это когда есть разделение на модули не зависимые между собой, но они все идут от главной задачи. 

Модели используемые в структурном подходе::

1. **SADT** (Structured Analysis and Design Technique) – ***метод структурного анализа и проектирования***

модели и соответствующие функциональные диаграммы, объединенные данным названием;

2. **DFD** (Data Flow Diagrams) – ***диаграммы потоков данных***-

используются для описания структуры проектируемой системы

ERD (Entity-Relationship Diagrams) – *диаграммы "**сущность-связь**"*

описания модели данных логического и физического уровней.

3 Структурное и объектно-ориентированное программирование в проектировании программного обеспечения распределенных информационных систем

3.1 Проектирование программного обеспечения распределенных информационных систем

Современные крупные проекты информационных систем характеризуются, следующими особенностями:

1. сложность описания (достаточно большое количество функций, процессов, элементов данных и сложные взаимосвязи между ними), требующая тщательного моделирования и анализа данных и процессов;

2. наличие совокупности тесно взаимодействующих компонентов (подсистем), имеющих свои локальные задачи и цели функционирования (например, традиционных приложений, связанных с обработкой транзакций и решением регламентных задач, и приложений аналитической обработки (поддержки принятия решений), использующих нерегламентированные запросы к данным большого объема);

3. отсутствие прямых аналогов, ограничивающее возможность использования каких-либо типовых проектных решений и прикладных систем;

4. необходимость интеграции существующих и вновь разрабатываемых приложений;

5. функционирование в неоднородной среде на нескольких аппаратных платформах;

6. разобщенность и разнородность отдельных групп разработчиков по уровню квалификации и сложившимся традициям использования тех или иных инструментальных средств;

7. существенная временная протяженность проекта, обусловленная, с одной стороны, ограниченными возможностями коллектива разработчиков, и, с другой стороны, масштабами организации-заказчика

Для успешной реализации проекта объект проектирования должен быть прежде всего адекватно описан, должны быть построены полные и непротиворечивые функциональные и информационные модели информационной системы.

Однако до недавнего времени проектирование информационных систем выполнялось в основном на интуитивном уровне с применением неформализованных методов, основанных

на искусстве, практическом опыте, экспертных оценках и дорогостоящих экспериментальных проверках качества функционирования информационных систем.

Кроме того, в процессе создания и функционирования информационных систем информационные потребности пользователей могут изменяться или уточняться, что еще более усложняет разработку и сопровождение таких систем.

Для целенаправленного выполнения проекта должен быть выполнен ряд работ, различных как по своему назначению, так и по квалификационным требованиям, предъявляемым к разработчикам. Иными словами, в ходе развития проекта командой разработчиков выполняются те или иные функции.

Функции, выполняемые разработчиками, — понятие неформализованное. В разных проектах оно может обретать свое содержание. 1)*функции кодирования*
т.е. записи программы на алгоритмическом языке по имеющимся спецификациям, анализа требований, *2)тестирования 3)отладки.*

В рамках деятельности менеджера любого проекта необходимо организовать распределение функций проекта между исполнителями.

считают эти действия одной из функций менеджера. В результате ее выполнения члены команды, выполняющей проект, начинают играть соответствующие роли.

Разработка современного программного комплекса представляет собой сложный и длительный процесс, который состоит из следующих **этапов:**

1. Предпроектные исследования, или анализ;
2. Проектирование и подготовка спецификаций;
3. Программирование, или кодирование;
4. Отладка;
5. Тестирование.

Во время предпроектного исследования собирается информация о предметной области, подбираются исходные данные и определяются функциональные требования к системе. Изучаются необходимые потребительские характеристики разрабатываемого программного обеспечения.

На этапе проектирования широко используется литература, стандарты и нормативные документы. Рассматриваются различные варианты реализации тех или иных функций и выбираются оптимальные. Здесь же разрабатывается архитектура, программный комплекс разбивается на подсистемы, определяются способы их взаимодействия. *Результатом этого*

этапа является документация, с которой можно приступать к программированию, а именно: блок-схемы, SDL-диаграммы, функциональные схемы, описания алгоритмов.

На стадии программирования, или кодирования, происходит воплощение всех наработок, сделанных на предыдущем этапе в работающую систему. Все действия фиксируются как на уровне комментариев в исходных кодах программ, так и в проектной документации. Необходимо организовать процесс таким образом, чтобы в любой момент времени был возможен откат на несколько шагов назад.

Отладка тесно связана с программированием, эти этапы разделяются весьма условно. Отладка может производиться либо непосредственно на оборудовании, для которого предназначено разрабатываемое программное обеспечение, либо на различных эмуляторах. *После программирования и отладки должна быть получена законченная рабочая система.*

Тестирование представляет собой проверку всех функциональных возможностей разработанного программного обеспечения на том оборудовании, для которого оно разрабатывалось и в условиях, максимально приближенных к реальным.

На каждом этапе применяются специализированные средства разработки: во время разработки алгоритмов и архитектуры в основном используются различные редакторы; при программировании применяются компиляторы и линковщики, при отладке – отладчики, при тестировании может использоваться специальное тестовое оборудование. В самом простом случае, без какой-либо автоматизации, все перечисленные средства являются автономными и никак не связанны между собой.

3.2 Структурный подход к проектированию информационных систем

Сущность структурного подхода к разработке информационных систем заключается в ее 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 взаимосвязаны. *При разработке системы «снизу-вверх» от отдельных задач ко всей системе целостность теряется, возникают проблемы при информационной стыковке отдельных компонентов.*

Структурный подход к программированию представляет собой методологию создания программ. **Его внедрение обеспечивает:**

1. повышение производительности труда программистов при написании и контроле программ;

2. получение программ, которые более пригодны для сопровождения, так как состоят из отдельных модулей;

3. создание программ коллективом разработчиков;

4. окончание создания программ в заданный срок.

В структурированных программах обычно легко прослеживается основной алгоритм, они удобнее в отладке и менее чувствительны к ошибкам программирования. Эти свойства являются следствием важной особенности подпрограмм, каждая из которых представляет собой во многом самостоятельный фрагмент программы, связанный с основной программой лишь с помощью нескольких параметров.

Все наиболее распространенные **методологии структурного подхода базируются на ряде общих принципов**

1. **принцип «разделяй и властвуй»** - принцип решения сложных проблем путем их разбиения на множество меньших независимых задач, легких для понимания и решения;

2. **принцип иерархического упорядочивания** - принцип организации составных частей проблемы в иерархические древовидные структуры с добавлением новых деталей на каждом уровне.

Выделение двух базовых принципов не означает, что остальные принципы являются второстепенными, поскольку игнорирование любого из них может привести к непредсказуемым последствиям (в том числе и к провалу всего проекта). Основными из них являются следующие принципы:

5. **принцип абстрагирования** - заключается в выделении существенных аспектов системы и отвлечения от несущественных аспектов;

6. **принцип формализации** - заключается в необходимости строгого методического подхода к решению проблемы;

7. **принцип непротиворечивости** - заключается в обоснованности и согласованности элементов;

8. **принцип структурирования данных** - заключается в том, что данные должны быть структурированы и иерархически организованы.

В структурном анализе используются в основном две группы средств, иллюстрирующих функции, выполняемые системой и отношения между данными. Каждой группе средств

соответствуют определенные виды моделей (диаграмм), наиболее распространенными среди которых, являются следующие:

- SADT (Structured Analysis and Design Technique) модели и соответствующие функциональные диаграммы;
- DFD (Data Flow Diagrams) диаграммы потоков данных;
- ERD (Entity-Relationship Diagrams) диаграммы «сущность-связь».

На стадии проектирования информационной системы модели расширяются, уточняются и дополняются диаграммами, отражающими структуру программного обеспечения: архитектуру программного обеспечения, структурные схемы программ и диаграммы экранных форм.

Структурное проектирование позволяет одновременно сосредотачиваться на меньшем количестве деталей.

Нисходящее проектирование хорошо работает , когда проблема имеет ясно выраженный иерархический характер.

3.3 Проектирование информационных систем на основе объектно-ориентированного подхода

Отличия структурного о объектно ориентированного подхода:

1.Первое отличие структурного подхода от объектно-ориентированного подхода *заключается в принципах декомпозиции и структурной организации элементов* (компонентов, модулей) системы. Согласно этим принципам система представляет собой структуру, состоящую из четко выраженных модулей, связанных между собой определенными отношениями.

Виды декомпозиции:

*1) **структурного подхода*** (первый вид декомпозиции) выполняется функциональная (процедурная, алгоритмическая) декомпозиция системы,

т. е. она представляется в виде иерархии (дерева) взаимосвязанных функций. На высшем уровне система представляется единым целым с наивысшей степенью абстракции и по мере детализации (добавления уровней) разбивается на функциональные компоненты с более конкретным содержанием.

2)***объектно-ориентированный***. В рамках этого подхода система разбивается на набор объектов, соответствующих объектам реального мира, взаимодействующих между собой путем посылки сообщений.

***2.*Вторым отличием является** *объединение в объекте как атрибутивных данных (характеристики, свойства), так и поведения (функции, методы). В функционально-ориентированных системах функции и данные хранятся (существуют) отдельно.*

*3.**Третье отличие*** двух подходов заключается *в структурной организации внутри модулей системы. В структурном подходе модуль состоит из функций, иерархически связанных между собой отношением композиции,*

т. е. функция состоит из подфункций, подфункция из подподфункций и т.д.

В объектно-ориентированном подходе иерархия выстраивается с использованием двух отношений: композиции и наследования.

При этом в объектно-ориентированном подходе «объект-часть» может включаться сразу в несколько «объектов-целое».

Таким образом, модуль в структурном подходе представляется в виде дерева, а в объектно-ориентированном подходе – в виде ориентированного графа, т. е. с помощью более общей структуры.

Объектно-ориентированное проектирование – это конструирование программных систем как структурных коллекций, реализующих абстрактные типы данных.

Объектно-ориентированное проектирование и объектно-ориентированное программирование улучшают возможности нисходящего проектирования, концентрируя больше внимание на данных системы, а не на том, что система делает.

Это подход позволяет создавать системы, которые легче сопровождать, они более гибкие, более устойчивые и более приспособлены к многократному использованию, чем создаваемые при нисходящем структурном подходе.

Преимущества объектно-ориентированного метода:

1. - работают на более высоком уровне абстракции;
2. - нет «прыжков» между фазами;
3. - поддерживают данные, которые имеют тенденцию, к большей стабильности, чем функции;
4. - поощряют и поддерживают классические достоинства хорошего программирования и проектирования;
5. - сопровождаются инструментами, обеспечивающими поддержку повторного использование кода.

Объектно-ориентированный подход имеет два аспекта:

1. объектно-ориентированная разработка программного обеспечения;

2. объектно-ориентированная реализация программного обеспечения.

1.Объектно-ориентированная разработка программного обеспечения связана с применением объектно-ориентированных моделей при разработке программных систем и их компонентов. **К объектно-ориентированной разработке относятся:**

- объектно-ориентированные технологии разработки программных систем;

- инструментальные средства, поддерживающие эти технологии.

Можно выделить следующие объектно-ориентированные методологии разработки программного обеспечения:

- RUP (Rational Unified Process);

- OMT (Object Modeling Technique);

- SA/SD (Structured Analysis/Structured Design);

- JSD (Jackson Structured Development);

- OSA (Object-Oriented System Analysis)

2.Объектно-ориентированный подход в проектировании, предполагает декомпозицию информационных систем. объектно-ориентированном подходе декомпозиции подлежат объекты, которые характеризуются определенной структурой данных. Здесь декомпозиция идет от данных. В объектно-ориентированном подходе выделяют классы объектов. Каждый класс содержит однородные объекты. Объектам одного класса присуще одинаковое множество методов реагирования на внешние сообщения.

Иерархическая декомпозиция системы представляется в виде иерархии классов объектов, а функционирование системы – в виде взаимодействия объектов, обменивающихся сообщениями.

Среди свойств объектов в объектно-ориентированном подходе можно отметить:

- инкапсуляция, что означает скрытие информации. Смысл этого свойства в том, что состав и структура атрибутов объекта не зависит от сообщений, поступающих извне;

- наследование – это свойство, связанное с выделением иерархических классов объектов, то есть существуют родительские и дочерние классы. Оно проявляется в том, что, методы реагирования объекта, предусмотренные родительским классом, автоматически присваивают объектам дочерних классов, то есть родительские классы имеют общие методы, а дочерние – как общие, так и частные;

- полиморфизм – возможность выбора объектом в ответ на получаемые им сообщения какого-либо метода из множества методов в зависимости от того, какое пришло сообщение.

Наличие этих свойств у объекта позволяет в объектно-ориентированном подходе добиться параллельности и автономности разработки отдельных компонент системы,

т.е. возможно создание прототипов с дальнейшей интеграцией отдельных прототипов в единую систему и использование итерационного подхода к разработке информационных систем.

достоинство объектно-ориентированного подхода

1)состоит в упрощении накопления типовых проектных решений с тем, чтобы в дальнейших разработках новых информационных систем осуществить сбор новой системы из готовых компонент.

Эта особенность связана с тем, что классы объектов повторяются в определенной мере при переходе от одной информационной системы к другой, а для повторяющихся классов уже запрограммированы методы, разработаны и описаны структуры объектов данных.

2) является отсутствие строгой последовательности в выполнении стадий как в прямом, так и в обратном направлениях процесса проектирования по отдельным компонентам.

3) в упрощении проектирования информационных систем, при наличии типовых проектных решений по отдельным компонентам, а также легкости модификации, поскольку модификация касается лишь отдельных компонент.

3.4. Сопоставление и взаимосвязь структурного и объектно-ориентированного подходов

Основой взаимосвязи между структурным и объектно-ориентированным подходами является общность ряда категорий и понятий обоих подходов (процесс и вариант использования, сущность и класс и др.). Эта взаимосвязь может проявляться в различных формах.

Взаимосвязь между структурным и объектно-ориентированным подходами достаточно четко просматривается в различных технологиях создания программного обеспечения.

Главный недостаток структурного подхода заключается в следующем: процессы и данные существуют отдельно друг от друга (как в модели деятельности организации, так и в модели программной системы), причем проектирование ведется от процессов к данным. Таким образом, помимо функциональной декомпозиции, существует также структура данных, находящаяся на втором плане.

В объектно-ориентированном подходе основная категория объектной модели - класс - объединяет в себе на элементарном уровне как данные, так и операции, которые над ними выполняются (методы). Именно с этой точки зрения изменения, связанные с переходом от структурного к объектно-ориентированному подходу, являются наиболее заметными. Разделение процессов и данных преодолено, однако остается проблема преодоления сложности системы, которая решается путем использования механизма компонентов.

Данные по сравнению с процессами являются более стабильной и относительно редко изменяющейся частью системы. Отсюда следует главное достоинство объектно-ориентированного подхода: объектно-ориентированные системы более открыты и легче поддаются внесению изменений, поскольку их конструкция базируется на устойчивых формах.

объектно-ориентированная модель наиболее адекватно отражает реальный мир, представляющий собой совокупность взаимодействующих объектов. Но на практике в настоящий момент продолжается формирование UML, и количество CASE-средств, поддерживающих объектно-ориентированный подход, невелико по сравнению с поддерживающими структурный подход.

Кроме того, *диаграммы, отражающие специфику объектного подхода (диаграммы классов и т.п.), гораздо менее наглядны и плохо понимаемы непрофессионалами. Поэтому одна из главных целей внедрения CASE-технологии, а именно снабжение всех участников проекта (в том числе и заказчика) общим языком «для передачи понимания», обеспечивается на сегодняшний день только структурными методами.*

При переходе от структурного подхода к объектному, как при всякой смене технологии, необходимо вкладывать деньги в приобретение новых инструментальных средств.

Здесь следует учесть и расходы на обучение (овладение методом, инструментальными средствами и языком программирования). Для некоторых организаций эти обстоятельства могут стать серьезными препятствиями.

Несмотря на отдельные критические замечания в адрес ООП, в настоящее время именно эта парадигма используется в подавляющем большинстве промышленных проектов. Однако, нельзя считать, что ООП является наилучшей из методик программирования во всех случаях.

Процедурное программирование лучше подходит для случаев, когда важны быстродействие и используемые программой ресурсы, но требует большего времени для разработки.

Объектное программирование подходит когда важна управляемость проекта и его модифицируемость, а также скорость разработки.

При классификации критических высказываний в адрес объектно-ориентированного программирования, можно выделить несколько аспектов критики данного подхода к программированию:

1. Критика рекламы объектно-ориентированного программирования.

2. Оспаривание эффективности разработки методами объектно-ориентированного программирования.

3. Производительность объектно-ориентированных программ.

4. Критика отдельных технологических решений в объектно-ориентированных-языках и библиотеках.

5.*снижению производительности программ из-за использования объектно-ориентированных средств:*

1.Динамическое связывание методов.

Обеспечение полиморфного поведения объектов приводит к необходимости связывать методы, вызываемые программой (то есть определять, какой конкретно метод будет вызываться) не на этапе компиляции, а в процессе исполнения программы, на что тратится дополнительное время. При этом реально динамическое связывание требуется не более чем для 20 % вызовов, но некоторые объектно-ориентированные-языки используют его постоянно.

2.Значительная глубина абстракции.

3.Наследование «размывает» код.

4. Инкапсуляция снижает скорость доступа к данным.

Запрет на прямой доступ к полям класса извне приводит к необходимости создания и использования методов доступа. И написание, и компиляция, и исполнение методов доступа сопряжено с дополнительными расходами.

5.Динамическое создание и уничтожение объектов.

Динамически создаваемые объекты, как правило, размещаются в куче, что менее эффективно, чем размещение их на стеке и, тем более, статическое выделение памяти под них на этапе компиляции.

Декомпозиция системы (в частности- программной системы - ПС)

- **Функциональная** – на основе потока данных с выделением обрабатывающих функций

- **Объектная** – на основе выделения сущностей, обладающих собственными наборами данных, состояниями и наборами операций

отличия =

- В первом случае внимание концентрируется на порядке происходящих событий (действиях)
- Во втором – на агентах, являющихся либо объектами, либо субъектами действий

Структурные единицы

Основной структурной единицей при функциональной декомпозиции является процедура как программная реализация алгоритма

Основной структурной единицей при объектно-ориентированной декомпозиции является объект как объединение данных и действий над ними

Сущность объектного подхода к разработке программных средств , состоит в том, что вся разработка системы основывает на выделении в объектов , их взаимосвязей, функций и свойств в той предметной области и среде исполнения , для работы в которых предназначается данное программное средство , а также в связи с теми задачами, которое данное ПС средство должно решать.

Контрольные вопросы:

113. Какие две методики проектирования вы знаете?
114. Что такое сущность ?
115. Что такое атрибут?
116. Какие сущности виды вы знаете?
117. Какие модели проектирования в структурном подходе вы знаете? Опишите их
118. Что представляет собой ER-диаграмма?
119. Какие типы связей вы знаете?
120. Какие графические примитивы вы знаете ?

Список использованных источников:

1. Технологии разработки программного обеспечения С.А. Орлов
2. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
3. Project Management For Dummies / Управление проектами для "чайников"
4. Л. Н. Боронина З. В. Сенук основы управления проектами

5. https://studopedia.su/3_11546_metodi-strukturnogo-proektirovaniya.html
6. https://studfiles.net/preview/3828360/page:7/
7. https://studfiles.net/preview/3828360/page:6/

ЛЕКЦИЯ 38

Тема: Типы информационных потоков. Case-средства проектирования информационных потоков данных

Цель: изучить основные методы проектирования потоков.

3. **SADT** (Structured Analysis and Design Technique) – *моделировании процессов в различных нотациях IDEF0*
4. **DFD** (Data Flow Diagrams) – *диаграммы потоков данных-*

используются для описания структуры проектируемой системы

ERD (Entity-Relationship Diagrams) – *диаграммы "сущность-связь"*

описания модели данных логического и физического уровней.

Результат структурного проектирования — *иерархическая структура* ПС.

Действия структурного проектирования зависят от типа информационного потока в модели анализа.

ТИПЫ ИНФОРМАЦИОННЫХ ПОТОКОВ

Типы информационных потоков бывают 2 типов:

1) **ПОТОК ПРЕОБРАЗОВАНИЙ**;

Рис. 5.1. Элементы потока преобразований

2) **ПОТОК ЗАПРОСОВ**.

Рис. 5.2. Структура потока запроса

1.в потоке преобразований *выделяют* *3 элемента*:

-Входящий поток,

-Преобразуемый поток

- Выходящий поток.

2.Потоки запросов имеют в своем составе особые элементы — запросы. Назначение элемента-запроса состоит в том, чтобы запустить поток данных по одному из нескольких путей. Анализ запроса и переключение потока данных на один из путей действий происходит в центре запросов

Проектирование для потока данных типа преобразование

Шаг 1. ***Проверка основной системной модели***. Модель включает: контекстную диаграмму IDEF0, словарь данных и спецификации процессов. Оценивается их согласованность с системной спецификацией.

Проверяем каркас входных и выходных данных которые мы хотим получить в результате ,анализируя IDEF0

Шаг 2. ***Проверки и уточнения диаграмм потоков данных уровней 1 и 2***. Оценивается согласованность диаграмм, достаточность детализации преобразователей.

Проверяем достаточность детализации 1-го и 2-го уровня IDEF0, если плохая детализация расширяем функционал

Шаг 3. ***Определение типа основного потока диаграммы*** потоков данных. Основной признак потока преобразований — отсутствие переключения по путям действий.

это тот тип потоков, которой у нас переходит из диаграммы в диаграмму детализации

Шаг 4. ***Определение границ входящего и выходящего потоков***, отделение центра преобразований.

Входящий поток — отрезок, на котором информация преобразуется из внешнего во внутренний формат представления.

Выходящий поток обеспечивает обратное преобразование — из внутреннего формата во внешний. Границы входящего и выходящего потоков достаточно условны.

Мы должны определится с результатом того, что мы хотим видеть на выходе,какеи ф-и, данные, объекты.

Шаг 5. ***Определение иерархическая структура*** ПС формируется нисходящим распространением управления.

В иерархической структуре:

1. модули верхнего уровня принимают решения;
2. модули нижнего уровня выполняют работу по вводу, обработке и выводу;
3. модули среднего уровня реализуют как функции управления, так и функции обработки.

Каждый модуль имеет контролирующий блок

1. ***Контроллер входящего потока*** (контролирует получение входных данных).
2. ***Контроллер преобразуемого потока*** (управляет операциями над данными во внутреннем формате).
3. ***Контроллер выходящего потока*** (управляет получением выходных данных).

Шаг 6. ***Детализация структуры ПС***. После прохождения преобразуемого поток слева направо. Возможны следующие варианты отображения:

---*1 преобразователь отображается в 1 модуль*;

Один главный блок 2-го уровня декомпозируется в один блок нижнего уровня детализации

--- *2-3 преобразователя отображаются в 1 модуль;*

Если у нас 3 основных блока 2-го уровня взаимосвязаны так, что выходящий поток имеет только 1блок декомпозицию нижнего уровня

---*1 преобразователь отображается в 2-3 модуля.*

Если у нас диаграмма 2-го уровня состоит из одного основного блока, а декомпозируется на 3 ф-и

Шаг 7. ***Уточнение иерархической структуры*** ПС. Модули разделяются и объединяются для:

1) повышения связности и уменьшения сцепления;
2) упрощения реализации;
3) упрощения тестирования;
4) повышения удобства сопровождения

CASE-СИСТЕМЫ

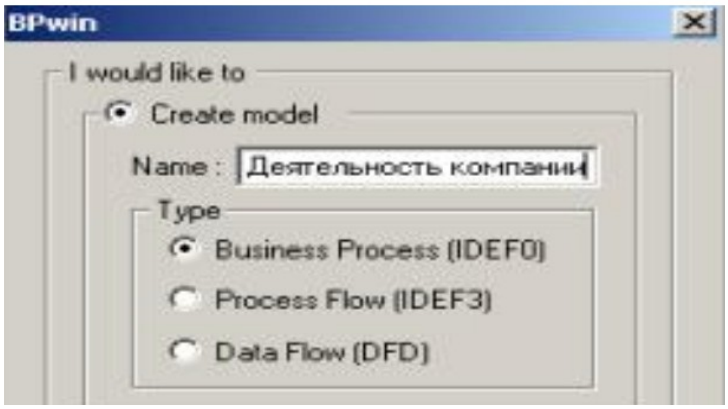
Имеющиеся на рынке программных продуктов CASE-системы для концептуального проектирования АИС чаще всего поддерживают методологию IDEF. В России широко известны продукты BPWin, ERWin, biasness modeler, OOWin фирмы Logic Works, Design/IDEF фирмы Meta Software, Silverrun фирмы CSA и др.

Например, пакет BPWin поддерживает работу с IDEF0, IDEF3, DFD моделями.

CASE- ситема(*Computer Aided Software/Sistem Engineering)* автоматизированное проектирование программ/ информационных систем.

ЭТО РАЗНОВИДНОСТИ БИЗНЕС МОДЕЛИРОВАНИЯ:

1. Моделирование бизнес процессов,
2. Технологического процесса ,
3. Поток данных



Два направления использования CASE-систем.

1. Как средство повышения эффективности в разработке сложного программного обеспечения, - соответствующие CASE-системы часто называют инструментальными средами разработки ПО.

2. Поддержка концептуального проектирования сложных систем.

моделирования деятельности предприятий, позволяя осуществлять проектирование и перепроектирование (реинжиниринг) бизнес-процессов. Такие CASE-системы часто называют системами BPR (Business Process Reengineering).

ХАРАКТЕРИСТИКИ CASE СРЕДСТВ

Основными характеристиками CASE средств:

1. **Наличие графического интерфейса**. CASE-технологии обеспечивают всех участников проекта, включая заказчиков, единым строгим, наглядным и интуитивно понятным графическим языком, позволяющим получать обозримые компоненты с простой и ясной структурой.

Понятный инерефейс позволяет заказчику участвовать в процессе разработки, а разработчикам - общаться с экспертами предметной области, разделять деятельность системных аналитиков, проектировщиков и программистов.

2. **Наличие репозитория**. Основа CASE-технологии - использование базы данных проекта (репозитория) для хранения всей информации о проекте, которая может разделяться между разработчиками в соответствии с их правами доступа.

позволяет раздавать права доступа разработчиками от БД и настраивать под каждого

3. **Гибкость применения**. CASE средства должны позволять проводить анализ процессов и создавать модели, сфокусированные на различных аспектах деятельности предприятия.

Мы можем видеть все процессы и анализируя их можем объединять в одну модель, а потом готовые модули использовать повторна в разработке

4. **Возможность коллективной работы**. поддерживает групповую работу над проектом, обеспечивая возможность работы в сети, экспорт-импорт любых фрагментов проекта для их развития и/или модификации, а также планирование, контроль, руководство и взаимодействие.

Разработчики могут параллельно смотреть разные часть модели проектирования и выгружать себе отдельные ее части.

5. **Построение прототипов**. CASE-технология дает возможность быстро строить макеты (прототипы) будущей системы, что позволяет заказчику на ранних этапах разработки оценить, насколько она приемлема.

Мы можем быстро спроектировать любой процесс и оценить его стоимость и кадровую востребованность

6. **Построение отчетов**. Вся документация по проекту генерируется автоматически на базе репозитория (как правило, в соответствии с требованиями действующих стандартов). При этом документация всегда отвечает текущему состоянию дел, поскольку любые изменения в проекте автоматически отражаются в репозитории.

Автоматически генерируется отчетность по добавлению каких либо моделей или процессов,вснесение изменений в проектирование.

Контрольные вопросы:

121. Из чего состоит поток преобразований?
122. Что такое поток запросов?
123. Какие направления case систем вы знаете?
124. Назовите и опишите основные характеристики case средств
125. Что такое входящий и выходящий поток?
126. Из каких шагов состоит проектирование потоков данных.
127. Что такое case-система ?
128. Какие два направления case-система вы знаете?
129. Назовите и опешите характеристики case-chtlcnd

Список использованных источников:

8. Технологии разработки программного обеспечения С.А. Орлов
9. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
10. Project Management For Dummies / Управление проектами для "чайников"
11. Л. Н. Боронина З. В. Сенук основы управления проектами
12. http://www.kpms.ru/Automatization/CASE_tools.htm
13. https://studfiles.net/preview/3828360/page:10/

ЛЕКЦИЯ 39

Тема: Объектно-ориентированная информационная система, моделирование UML диаграмм. Use-case диаграммы.

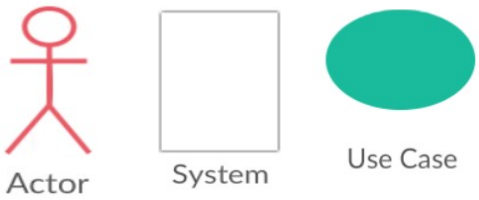
Цель: научиться строить UML диаграммы

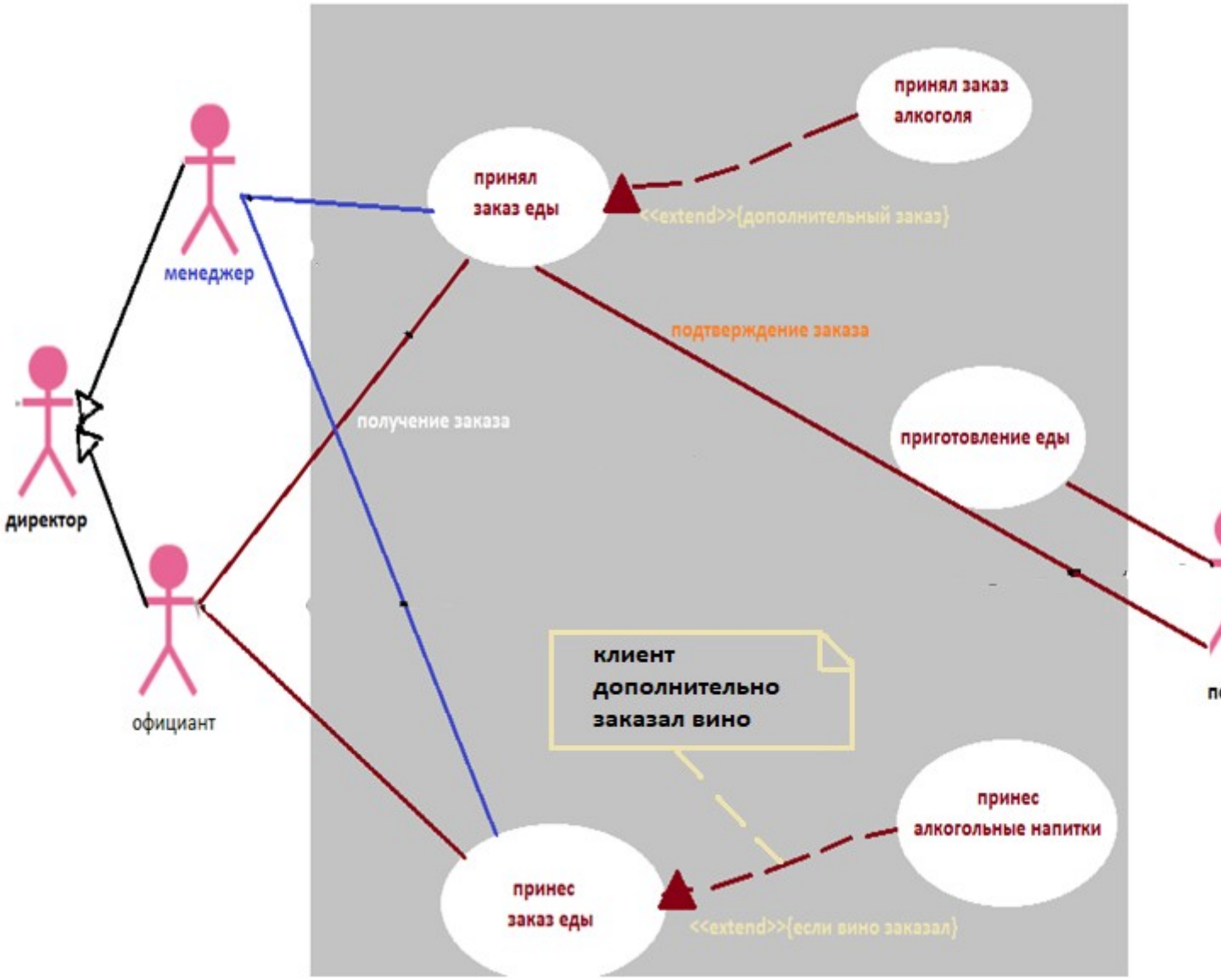
К числу средств *визуального моделирования объектно-ориентированных информационных систем (ИС) относится* **Rational Rose**.

Идея унификации привела к появлению языков 3-го поколения. В качестве стандартного языка третьего поколения был принят Unified Modeling Language (UML)

рецедент (Use case) - это описание последовательности выполняемых системой действий, которая производит наблюдаемый результат, значимый для какого-то определенного актера (Actor).

USE-CASE состоит из:





Use Case (диаграмма прецедентов)

UML — стандартный язык для написания моделей анализа, проектирования и реализации объектно-ориентированных программных систем UML может использоваться для визуализации, спецификации, конструирования и документирования результатов программных проектов.

UML — это не визуальный язык программирования, но его модели прямо транслируются в текст на языках программирования и даже в таблицы для реляционной БД.

Словарь UML образуют три разновидности строительных блоков: предметы, отношения, диаграммы. Предметы — это абстракции, которые являются основными элементами в модели, отношения

Предметы в UML

В UML имеются четыре разновидности предметов: ❱

1. структурные предметы; ❱
2. предметы поведения; ❱
3. группирующие предметы; ❱

4. поясняющие предметы.

Эти предметы являются базовыми объектно-ориентированными строительными блоками. Они используются для написания моделей.

Структурные предметы являются существительными в UML-моделях.

Они представляют статические части модели — понятийные или физические элементы.

Перечислим восемь разновидностей структурных предметов.

1. **Класс** — описание множества объектов, которые разделяют одинаковые свойства, операции, отношения и семантику (смысл).

Класс реализует один или несколько интерфейсов.

графически *класс отображается в* ***виде прямоугольника***, обычно включающего секции с именем, свойствами (атрибутами) и операциями.

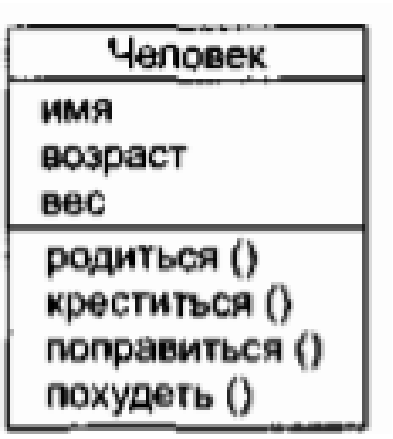


Рис. Классы

2. **Интерфейс**.— набор операций, которые определяют услуги класса или компонента описывает поведение элемента, видимое извне.

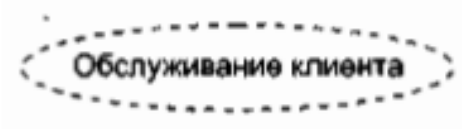
конструкция, определяющая методы и свойства, предоставляемые классом.

Имя интерфейса обычно начинается с буквы «I».

Интерфейс редко показывают самостоятельно. Обычно его присоединяют к классу или компоненту, который реализует интерфейс.

3. **Кооперация** (сотрудничество) определяет взаимодействие и является совокупностью ролей и других элементов, которые работают вместе для обеспечения коллективного поведения более сложного, чем простая сумма всех элементов.

Конкретный класс может участвовать в нескольких кооперациях. Эти кооперации представляют реализацию паттернов (образцов), которые формируют систему.



4. **Актер** — набор согласованных ролей, которые могут играть пользователи при взаимодействии с системой (ее элементами Use Case). Каждая *роль требует от системы определенного поведения*.

5. **Элемент Use Case** (Прецедент) — описание последовательности действий, выполняемых системой в интересах отдельного актера и производящих видимый для актера результат

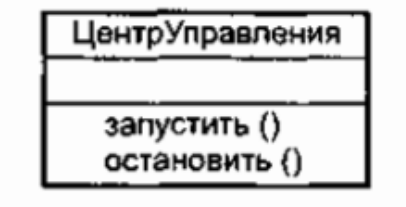
6. 

Элемент Use Case — это последовательность взаимодействий в диалоге, выполняемом актером и системой. Запускается элемент Use Case актером, поэтому удобно выявлять элементы Use Case с помощью актеров.

7. **Активный класс** — класс, чьи объекты имеют один или несколько процессов (или потоков) и поэтому могут инициировать управляющую деятельность.

Активный класс похож на обычный класс за исключением того, что *его объекты действуют одновременно с объектами других классов*.

активный класс изображается как утолщенный прямоугольник, обычно включающий имя, свойства (атрибуты) и операции.

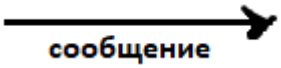


Предметы поведения — динамические части UML.

То что взаимодействует на диаграмме

. **Существует две основные разновидности предметов поведения**.

1. Взаимодействие — поведение, заключающее в себе набор сообщений, которыми обменивается набор объектов в конкретном контексте для достижения определенной цели.



2. **Конечный автомат** — поведение, которое определяет последовательность состояний объекта или взаимодействия, выполняемые в ходе его существования в ответ на события

С помощью конечного автомата может определяться поведение индивидуального класса или кооперации классов.

Элементами конечного автомата являются состояния, переходы (от состояния к состоянию), события (предметы, вызывающие переходы) и действия (реакции на переход).

прямоугольник, обычно включающий его им и его подсостояния (если они есть).

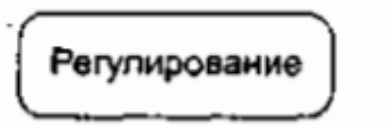


Рис.. Состояния

2.1 Поясняющие предметы — разъясняющие части UML-моделей, сообщения,примечания.

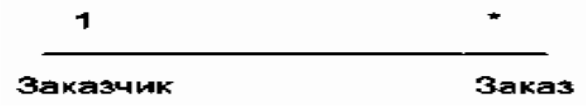
ОТНОШЕНИЯ В UML

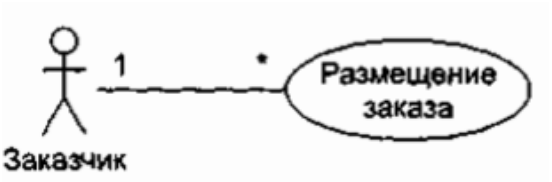
Между актером и элементом Use Case возможен только один вид отношения — ассоциация, отображающая их взаимодействие

В UML имеются четыре разновидности отношений:

1. Ассоциация
2. Зависимость
3. Обобщение
4. реализация

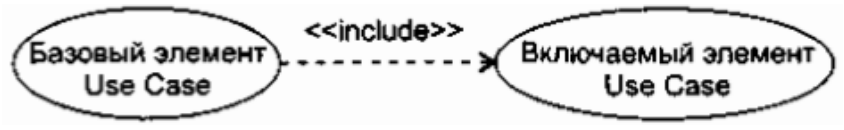
1) АССОЦИАЦИЯ;

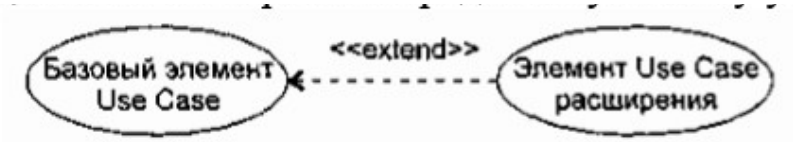




Между актерами допустимо отношение обобщения , означающее, что экземпляр потомка может взаимодействовать с такими же разновидностями экземпляров элементов Use Case, что и экземпляр родителя.

2) ЗАВИСИМОСТЬ;



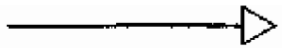


Между элементами Use Case определены отношение обобщения и две разновидности отношения зависимости — **включения** <<include>> **(**уточнение элемента Use Case) и **расширения**<<extend>>.

расширения <<extend>>. *используют*: ❱

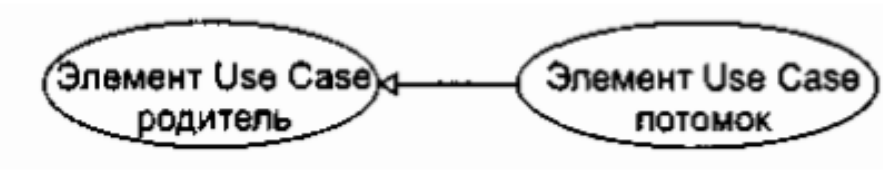
1. для моделирования вариантных частей элементов Use Case; ❱
2. для моделирования сложных и редко выполняемых альтернативных последовательностей;
3. ❱ для моделирования подчиненных последовательностей, которые выполняются только в определенных случаях;
4. ❱ для моделирования систем с выбором на основе меню.

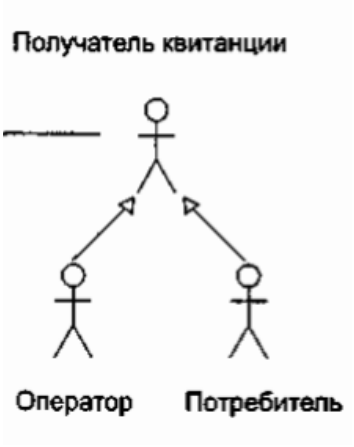
включения <<include>> Уточнение модели сводится к выявлению одинаковых частей в элементах Use Case и извлечению этих частей. Любые изменения в такой части, выделенной в отдельный элемент Use Case, будут автоматически влиять на все элементы Use Case, которые используют ее совместно.

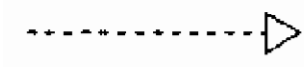
3) ОБОБЩЕНИЕ; 

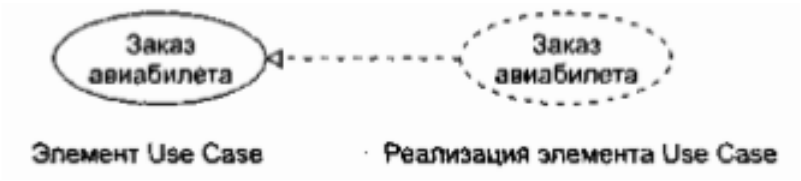
абстрактные элементы Use Case могут использоваться другими абстрактными элементами Use Case. Так образуется иерархия. *При построении иерархии абстрактных элементов Use Case руководствуются правилом: выделение элементов Use Case прекращается при достижении уровня отдельных операций над объектами*.

Абстрактный актер — это общий фрагмент роли в нескольких конкретных актерах.





4) **РЕАЛИЗАЦИЯ**. 



кооперация реализует конкретный элемент Use Case.

требования к информационной системе авиакассы задаются множеством элементов Use Case, каждый из которых реализуется отдельной кооперацией. *Все кооперации применяют одни и те же классы, но все же имеют разную функциональную организацию*.

Эти отношения являются базовыми строительными блоками отношений. Они используются при написании моделей.

1. **Зависимость** — семантическое отношение между двумя предметами, в котором изменение в одном предмете (независимом предмете) может влиять на семантику другого предмета (зависимого предмета).

2. **Ассоциация** — структурное отношение, которое набор связей, являющихся соединением между двумя

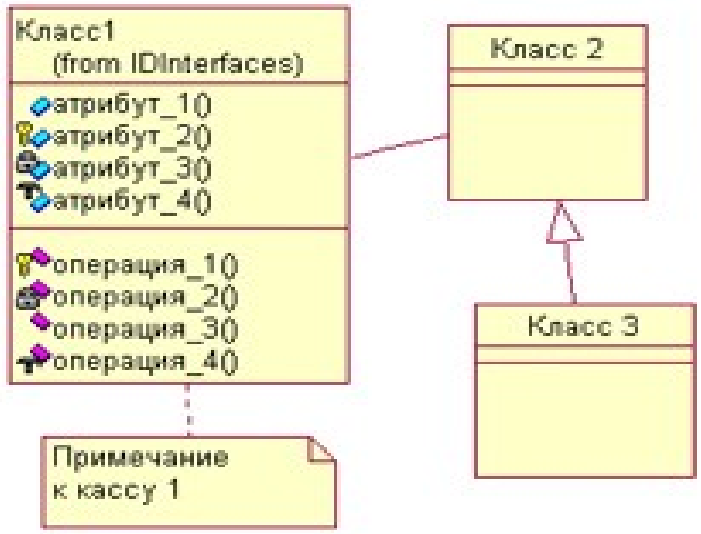
3. **Обобщение** — отношение специализации/обобщения, в котором объекты специализированного элемента (потомка, ребенка) могут заменять объекты обобщенного элемента (предка, родителя).

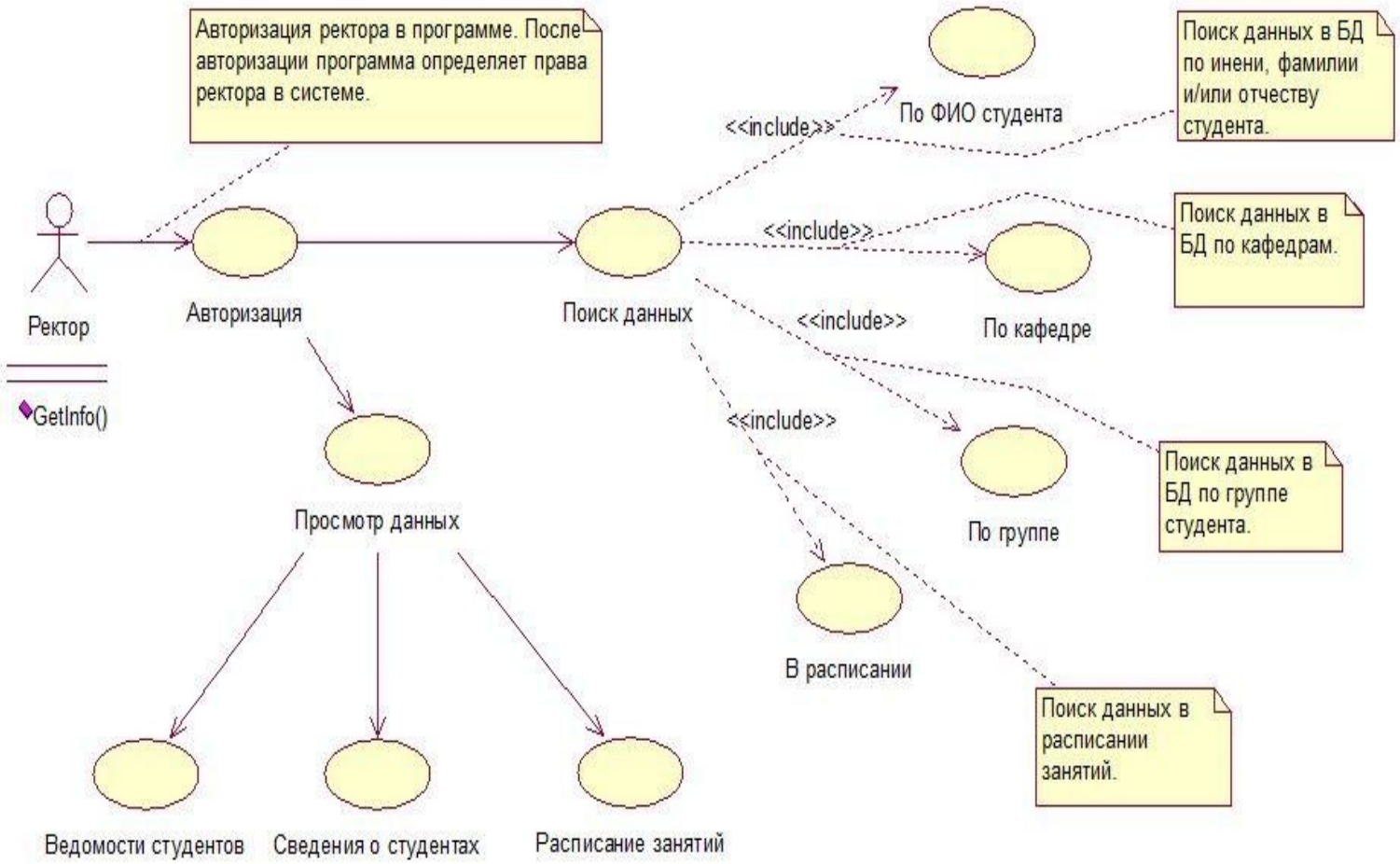
Иначе говоря, потомок разделяет структуру и поведение родителя.

4. **Реализация** — семантическое отношение между классификаторами, где один классификатор определяет контракт, который другой классификатор обязуется выполнять (к классификаторам относят классы, интерфейсы, компоненты, элементы Use Case, кооперации).

Отношения реализации применяют в двух случаях: между интерфейсами и классами (или компонентами), реализующими их; между элементами Use Case и кооперациями, которые реализуют их.

связывают эти предметы, диаграммы группируют коллекции предметов.





Контрольные вопросы:

130. Из чего состоит use-case?

131. Что такое UML?
132. Что такое use-case?
133. Назовите и опишите разновидности предметов в UML
134. Какие отношения в UML вы знаете, как они обозначаются?

Список использованных источников:

14. Технологии разработки программного обеспечения С.А. Орлов
15. Технологии разработки программного обеспечения В.В. Бахтизин, Л.А. Глухова
16. Project Management For Dummies / Управление проектами для "чайников"
17. Л. Н. Боронина З. В. Сенук основы управления проектами
18. http://www.kpms.ru/Automatization/CASE_tools.htm
19. https://studfiles.net/preview/3828360/page:10/

ЛЕКЦИЯ 40

Тема: Мониторинг и оценка качества работоспособности программного обеспечения.
Цель: понять когда нужно мониторинг и оценка, как проводится мониторинг ПО.

Мы поговорим о мониторинге и оценке ПО. Чем отличается и зачем нужно мониторить и оценивать ПО на стадиях разработки.

Мониторинг – лишь отслеживание текущей «картинки».
Оценка предполагает глубокий анализ,

МОНИТОРИНГ

мониторинг – это систематический сбор информации о значениях заранее выбранных индикаторов для обеспечения руководителей и других заинтересованных сторон сведениями о том, насколько успешно выполняется программа, в какой степени достигаются поставленные цели и как используются фонды, выделенные на данную программу .

Другими словами, мониторинг отвечает на вопрос «Как идут дела?» и позволяет оперативно отслеживать ход программы по сравнению с планом.

Это информация которая предоставляется или заказчику или руководителю проекта о том, как протекает процесс разработки и вкладывается процесс разработки в график, бюджет..

Принципиальное отличие мониторинга от оценки – глубина анализа.

Оценка предполагает глубокий анализ,

Задача выполнена или нет, кем выполнена в какие сроки, с тестирование или нет, на каких моментах были затруднения в разработки

Мониторинг – лишь отслеживание текущей «картинки».

На данный момент задача выполнена или нет

С технической точки зрения главные отличия между мониторингом и оценкой заключаются в том, что

мониторинг	**оценка**
постоянно действующая система	проводится время от времени
основана исключительно на замерах значений индикаторов	проводится с учетом данных мониторинга (значений индикаторов), но не ограничивается ими.

Именно поэтому системы мониторинга и оценки являются взаимодополняющими, но никак не могут заменить друг друга.

Для построения системы мониторинга программы необходимо сделать следующее:

1. Определить, какие характеристики программы нам необходимо отслеживать.
2. Определить измеряемые показатели (индикаторы), по которым можно будет отслеживать эти характеристики.
3. Установить источники информации для проведения мониторинга (организации, отделы, отдельные люди или группы людей, документы и т.д.).
4. Выбрать методы сбора информации.
5. Определить периодичность и график сбора информации (замера значений индикаторов).
6. Назначить ответственных за получение необходимой информации и договориться с теми, кто эту информацию будет предоставлять.
7. Определить технологию обработки и анализа получаемой информации.
8. Спланировать, как и кому будут переданы данные мониторинга, а также кто и как будет их использовать.
9. Учесть в бюджете программы расходы, необходимые для проведения мониторинга.

Работа любой подсистемы в программе требует ресурсов. Ресурсы стоят денег. Если не учесть расходы на мониторинг программы на стадии проектирования, то при внедрении этой системы могут возникнуть большие трудности.

К сожалению, именно такую ошибку зачастую совершают авторы проектов и программ: подробно *рассчитывают все статьи бюджета, связанные с реализацией программы, но не учитывают расходы на мониторинг.*

Разрабатывать систему мониторинга нужно на стадии создания программы. Основой для построения качественной системы мониторинга является тщательная проработка логики программы, корректная формулировка ее целей.

ОЦЕНКА ПРОГРАММЫ

Основные этапы проведения оценки включают:

1. *Возникновение потребности в информации.*

Это когда нам не хватает информации для реализации подавленных задач.

Пример: выведите мне статистику за каждый месяц (какой период?, какую именно статистику? По пользователям или внутреннюю разработку?)

2. *Постановку задачи.*

Что мы должны оценить, в какой срок и какую информацию должны предоставить на выходе

3. *Планирование оценки.*

Расписываем по датам, когда проводится глубокая оценка сбора данных, и кем она будет проводится, какими подразделениями.

4. *Сбор данных.*

Собираем данные с подразделений

5. *Анализ данных.*

Получили данные, мы должны их обработать, что нам показывают определенные показатели

6. *Подготовку отчета.*

Готовим отчет: по срокам, по задачам, по бюджету

7. *«Обратную связь» по результатам оценки.*

Уточняем причины, если есть отклонения от нормы показателей.

8. *Принятие решения.*

Ставим новые задачи для решения этих проблем или принимаем решения нужны нам вообще эти задачи или не, расширить бюджет по задаче или сократить.

Строго говоря, первый и последний этапы выходят за рамки оценки программы. Собственно *оценка начинается с постановки задачи и заканчивается «обратной связью».* Однако мы рассматриваем оценку как одну из функций управления, поэтому считаем важным увязывать этапы проведения оценки с жизнью программы.

Описание этапов оценки :

1 этап. Возникает проблема что-то произошло, что требует анализа и изучения.

2 этап. Формулировка Задач оценки – собрать сведения, которые необходимы для принятия управленческих решений. Поэтому отправной точкой для проведения оценки является возникновение потребности в информации для принятия решений.

Естественно, эта потребность возникает у вполне конкретных людей, принимающих решения. Оценка будет максимально полезной, если она сориентирована на этих людей и проводится с учетом того, как они будут пользоваться полученными результатами .

преобразование возникшей потребности в задание на проведение оценки. Главным *компонентом задания являются вопросы, на которые нужно ответить, чтобы получить необходимую для принятия решений информацию.* Кроме того, в задании оговариваются детали проведения оценки:

-источники и методы сбора информации,

-требования к отчету, кто будет участвовать в проведении оценки,

- график работ, организационные вопросы,

-условия использования и распространения полученной информации,

-необходимые для проведения оценки ресурсы.

3 этап: планирование и подготовка к проведению оценки – разрабатывают детали проведения работ и подготавливают необходимые инструменты. На этом этапе могут быть разработаны анкеты, списки вопросов для интервью, методики проведения наблюдения и т.д.

4 этап: сбор данных. Для этого используют интервьюирование (индивидуальное и групповое), анкетирование, наблюдение, различные методы анализа документации. При сборе данных важно беспристрастно фиксировать факты, не пытаясь их интерпретировать.

5 этап: Анализ Время для обобщений и выводов наступит, когда будут собраны мнения нескольких участников, клиентов, представителей организации. **анализа** полученной информации заключается в том, что выводы должны базироваться не на отдельных фактах, а на их совокупности.

Например, один из участников проекта может очень эмоционально и убедительно доказывать бесперспективность выполненной работы.

Человек, проводящий оценку, должен воспринимать это лишь как мнение одного из участников, несмотря на убедительность приведенных доводов.

6 этап: Форма и содержание отчета о проведенной оценке определяются при согласовании задания. Главное на этом этапе – помнить о том, что отчет – важнейшая часть оценки. Отчет предназначен для тех, кто инициировал оценку, и должен быть удобен, в первую очередь, для них. Отчет, написанный с мыслью о будущем читателе, обречен на успех.

7 этап: «Обратная связь» по результатам оценки – эта наиболее волнительная для всех сторон фаза проведения оценки.

1.Для тех, кто *инициировал оценку*, – это своего рода заключение об успешности работы, которой они руководили или которую финансировали.

2.Для тех, *кто проводил оценку*, это экзамен на профессионализм и проверка качества выполнения задания.

3.Для тех, *чья программа оценивалась*, это прелюдия к важным решениям, которые могут повлиять на будущее их организации.

Оценка призвана показать программу такой, как она есть. Это значит, что сообщение о результатах может содержать и положительные, и отрицательные выводы.

Как рассказать о них, не обидев людей? Как выслушать сообщение о результатах оценки без обид на тех, кто «принес плохие новости»? Все стороны должны тщательно подготовиться к этой встрече.

8 этап: принятие решений с учетом полученной информации. Здесь главная роль принадлежит людям, у которых возникла потребность в информации и которые инициировали проведение оценки. Принятие решений – это их компетенция.

Результатом оценки является информация.

Контрольные вопросы:
1. Что такое мониторинг?
2. Что такое оценка?
3. Сравнительный анализ мониторинга и оценки?
4. Этапы проведения мониторинга ПО?
5. Что нужно для проведения монитроинга

Список использованных источников:

21. Государственный стандарт Российской Федерации. Информационная технология. Сопровождение программных средств
22. Основы маркетинга учебное пособие Суркова Е.В.
23. http://www.processconsulting.ru/korotko_o_glavno/ocenka_programmy/

ЛЕКЦИЯ 41-42

Темы:

41Фундаментальный процесс тестирования программного продукта. Методы тестирования.

42 Основные принципы тестирования ПО.

Цель: изучить основные методы тестирования. Научиться применять основные шаги тестирования.

планирование и документирование процесса.

В обязанности тестировщиков входит разработка тестовых сценариев, а также подготовка тестирования и оценка его результатов

Тести́рование ПО— процесс исследования, испытания, имеющий своей целью проверку соответствия между реальным поведением программы и её ожидаемым поведением на конечном наборе тестов, выбранных определенным образом (ISO/IEC TR 19759:2005)

. В рамках тестирования процесса можно выделить ключевые шаги:

1. – планирование и управление;
2. – анализ и проектирование;
3. – внедрение и реализация;
4. – оценка критериев выхода и написание отчетов;
5. – действия по завершению тестирования.

Здесь действия описаны в логической последовательности, но в условиях реального проекта они могут накладываться, происходить одновременно или даже повторяться. Обычно, происходит адаптация этих шагов под нужды конкретной системы или проекта. Рассмотрим их

1. ПЛАНИРОВАНИЕ И УПРАВЛЕНИЕ

Планирование тестирования включает действия, направленные на определение основных целей тестирования и задач, выполнение которых необходимо для достижения этих целей.

Мы определяем сановные действия, что мы должны сделать, чтобы получить результат. И по каждому действию смотрим что в результате должно получиться и составляем для этого тесты.

действие заказа товара: зарегистрироваться, найти товар, добавить в корзину количество товара, оплатить….результат на руках документ подтверждающий оплату товара с его кодом и стоимостью.

В процессе планирования мы убеждаемся в том, что мы правильно поняли цели и пожелания заказчика и объективно оценили уровень риска для проекта, после чего ставим цели и задачи для, собственно, тестирования.

Для более ясного описания целей и задач тестирования составляются такие документы как тест-политика, тест-стратегия и тест-план.

Тест-политика – высокоуровневый документ, описывающий принципы, подходы и основные цели компании в сфере тестирования.

Описываются общие принципы тестирования(раннее тестирование, исчерпывающее тестирование, тестирование показывающее наличие дефектов.)

основные цели указываются, что должно показать тестирование(юзабилити, тестирование на отказ и восстановлении системы, конфигурационное тестирование(клиентский и серверный), тестирование документации…)

Тест-стратегия – высокоуровневый документ, содержащий описание уровней тестирования и подходов к тестированию в пределах этих уровней. Действует на уровне компании или программы (одного или больше проектов).

Определяем сферу для тестирования(торговля, образование, спорт..) и в зависимости от сферы выбираем виды и принципы тестирования

Для спорта нужно стрессовое тестирование, так как пользователь может быть под впечатлением и заполнить не верную ставку на спорт.

Тест-план – документ, описывающий средства, подходы, график работ и ресурсы, необходимые для проведения тестирования. Помимо прочего, определяет инструменты тестирования, функциональность, которую требуется протестировать, распределение ролей в команде, тестовое окружение, используемые техники тест-дизайна, критерии начала и

окончания тестирования и риски. То есть, это подробное описание всего процесса тестирования.

Это документ, где описывается даты выполнения тестов с их результатами на каждом этапе, инструменты с помощью чего мы будем тестировать, описывается весь процесс детально.

В любой деятельности, управление не заканчивается планированием. Нам нужно контролировать и измерять прогресс. Именно поэтому управление тестированием – непрерывный процесс.

Управление тестированием – сопоставление текущей ситуации в процессе тестирования с планом и составление отчетности.

Мы придерживаемся сценария тестирования и по результатам прохождения каждого этапа заполняем документацию с показателями тестирования

В свою очередь, данные, полученные в ходе контроля над процессом, учитываются при планировании дальнейших действий.

2. АНАЛИЗ И ПРОЕКТИРОВАНИЕ

Анализ и проектирование тестов – это процесс написания тестовых сценариев и условий на основе общих целей тестирования.

Это сам процесс написание всех сценарием тестирования для выбранной сферы деятельности.

В процессе анализа и проектирования мы разрабатываем тестовые сценарии на основании общих целей тестирования, определенных во время планирования.

Тестовый сценарий – документ, определяющий установленную последовательность действий при выполнении тестирования.

3. ВНЕДРЕНИЕ И РЕАЛИЗАЦИЯ

Во время выполнения тестирования происходит написание *тест-кейсов*, на основе написанных ранее тестовых сценариев, собирается необходимая для проведения тестов информация, подготавливается *тестовое окружение* и запускаются тесты.

Когда мы уже описали все сценарии мы создаем Тест кейсы -это набор данных, сценариев которые будет поступать для входа тестирования(0,1, даты, запятые, символы) исключительные ситуации и запускается тест.

А сам тест запускаться может с помощью эмуляции различных ОС, экранов и т.д.

Тест-кейс— это профессиональная документация тестировщика, последовательность действий направленная на проверку какого-либо функционала, описывающая как прийти к фактическому результату

Что нужно сделать чтобы проверить юзабилити сайта (скорость поиска главных элементов, методы их выбора ориентированность в функционале)

Тестовое окружение – аппаратное и программное обеспечение и другие средства, необходимые для выполнения тестов.

4. ОЦЕНКА КРИТЕРИЕВ ВЫХОДА И НАПИСАНИЕ ОТЧЕТОВ

Критерии выхода определяют, когда можно завершать тестирование. Они необходимы для каждого уровня тестирования, поскольку нам необходимо знать, достаточно ли было проведено тестов.

Есть оценочные таблицы, где указываются когда можно завершать тестирование

К примеру: основные методы тестирования проверили на корректность ввода логина:

Проверить на ввод –символа @. Т.к. если его не указать-это не будет адресом почты.

При оценке критериев выхода необходимо:

1. –проверить, было ли проведено достаточное количество тестов, достигнута ли нужная степень обеспечения качества системы.

Проверить обязательно основные сценарии, по которым происходит к примеру регистрации, если основные сценарии прошли тестирование-переходим к другому сценарию.

2. –убедится в том, что нет необходимости проводить дополнительные тесты. Если все же такая необходимость есть, возможно, потребуется изменить установленный критерий выхода.

Если появились спорные ситуации-это значит нужно проводить доп.тестирование или параметры входных данных расширить.

Пример: нужно ли реагировать на заглавную букву в поиске или нет?

У клиенту вся документация к примеру капс лукам написана.

После окончания тестирования происходит написание отчета, который будет доступен всем заинтересованным сторонам.

Ведь не только тестировщики должны знать результаты выполнения тестов, – эта информация может быть необходима многим участникам процесса создания ПО.

5. ДЕЙСТВИЯ ПО ЗАВЕРШЕНИЮ ТЕСТИРОВАНИЯ

При завершении тестирования мы собираем, систематизируем и анализируем информацию о его результатах. Она может пригодиться позже – при выпуске готового продукта.

Все тесты систематизируем по датам и сценариям их проработки, чтобы в нужный момент можно было найти определенные тесты.

Основные цели этого этапа:

1. убедиться, что вся запланированная функциональность действительно была реализована;

проверяем основные сценарии работы сайта, чтобы убедиться, что при разработке ничего не забыли от входа, до заказа продукта.

2. проверить, что все отчеты об ошибках, поданные ранее, были, так или иначе, закрыты;

ранее найденные ошибки исправлены-это отдельные сценарии с пометкой исправить!,они проверяются повторно

3. завершение работы тестового обеспечения, тестового окружения и инфраструктуры;

проверяются на использование различных ОС, и размеров экрана

4. оценить общие результаты тестирования и проанализировать опыт, полученный в его процессе.

Даем выводы, показываем клиенту, что тестирование было необходимо + показываем его целесообразность различных методик +накапливаем опыт для следующих тестов.

Т.к. входные данные основные у нас уже будут, повторно их создавать не обязательно, их просто расширим если нужно.

ОСНОВЫНЕ ПРИНЦИПЫ ТЕСТИРОВАНИЯ:

1. Тестирование показывает наличие дефектов

Тестирование может показать наличие дефектов в программе, но не доказать их отсутствие.

Т.е. нельзя говорить, что дефектов нет на юзабилити, мы можем сказать, что они есть в определенном месте т.к. при написании кода могут появится новые, и тогда наше заключение, что их нет будет ложным.

Тем не менее, важно составлять тест-кейсы, которые будут находить как можно больше багов. Таким образом, при должном тестовом покрытии, тестирование позволяет снизить вероятность наличия дефектов в программном обеспечении.

2. Исчерпывающее тестирование невозможно

Невозможно провести тестирование, которое бы покрывало все комбинации пользовательского ввода и состояний системы, за исключениям совсем уж примитивных случаев. Всегда есть шанс пропустить определенную комбинацию

необходимо использовать анализ рисков и расстановку приоритетов, что позволит более эффективно распределять усилия по обеспечению качества ПО.

3. Раннее тестирование

Тестирование должно начинаться как можно раньше в жизненном цикле разработки программного обеспечения, и его усилия должны быть сконцентрированы на определенных целях.

4. Скопление дефектов

В основном, большую часть критических дефектов находят в ограниченном количестве модулей. Это проявление принципа Парето: 80% проблем содержатся в 20% модулей.

Модуль работы с текстом

5. Парадокс пестицида

Прогоняя одни и те же тесты вновь и вновь, Вы столкнетесь с тем, что они находят все меньше новых ошибок. Поскольку система эволюционирует, многие из ранее найденных дефектов исправляют и старые тест-кейсы больше не срабатывают.

Изменяйте входные параметры для тестирования.

6. Тестирование зависит от контекста

Выбор методологии, техники и типа тестирования будет напрямую зависеть от природы самой программы. Например, программное обеспечение для медицинских нужд требует гораздо более строгой и тщательной проверки, чем, скажем, компьютерная игра.

Из тех же соображений, сайт с большой посещаемостью должен пройти через серьезное тестирование производительности, чтобы показать возможность работы в условиях высокой нагрузки.

7. Заблуждение об отсутствии ошибок.

Тот факт, что тестирование не обнаружило дефектов, еще не значит, что программа готова к релизу. Нахождение и исправление дефектов будут не важны, если система окажется неудобной в использовании, и не будет удовлетворять ожиданиям и потребностям пользователя.

И еще несколько важных принципов:

— тестируйте как позитивные, так и негативные сценарии;

— указания ожидаемого результата выполнения тестов.

Контрольные вопросы:

6. что такое тестирование ПО?
7. Ключевые шаги тестирования?
8. Опишите этап планирована и управления?
9. Опишите этап анализа и проектирования?
10. Опишите этап внедрения и реализации?
11. Опишите этап оценки критериев выхода и написание отчетов?
12. Опишите этап действия по завершению тестирования?
13. Основные принципы тестирования.

Список использованных источников:

24. Государственный стандарт Российской Федерации. Информационная технология. Сопровождение программных средств
25. Основы маркетинга учебное пособие Суркова Е.В.
26. http://www.processconsulting.ru/korotko_o_glavno/ocenka_programmy/

ЛЕКЦИЯ 43-44

Темы:

43тестирование Usability программного продукта с учетом интернационализации и локализации

44 Тестирование Usability интерфейса, определения показателя потерянности фокуса

Цель: изучить основные методы тестирования юзабилити и понять принципы интернационализации и локализации.

Виды тестирования:

1. *Юзабилити*
2. *Интернационализация и локализация*

ЮЗАБИЛИТИ

Юзабилити-тестирование (Usability – удобство) – это проверка программного продукта на соответствие с требованиями в плане удобности использования приложения. Таким образом, с помощью юзабилити-тестирования мы можем определить эргономичность (приспособленность к использовании) программы.

Проверка юзабилити приложения заключается в:

1. *Оценка соответствия дизайна приложения к его функциональности, заданной заказчиком.*

Кнопка добавить не должна удалять обьект

2. *Анализ используемых графических элементов, цветового оформления с точки зрения восприятия.*

Есть принятые цвета удалять, запрет –красны, скачать, добавить, успешно-зеленый, предупреждения, напоминания-желтый, оранжевый.

Основные элементы должны выделяться ,а не прописываться серым

3. *Оценке удобства навигации и ссылочной структуре.*

Списком, строкой , навигация собранного списка, двойная навигация(основная трока всегда закреплена, вложенная при пролистывании страницы сворачивается)

4. *Анализ текстового наполнения сайта.*

Сокращения должны быть обще принятыми или с расшифровкой.

Блоки должны быть логически завершенными перетекающими по логическому смыслу.

5. *Оценка удобства использования функциями приложения (сервисами, если это сайт).*

W3C mobileOK Checker — онлайн-сервис от W3C проверки сайта для **мобильных устройств**.
W3C Links Validator — этот инструмент анализирует гиперлинки и якоря (ancor) в HTML/XHTML документах, что очень важно при тестировании сайта на нерабочие, «битые» линки.
Pingdom Tools — полезный онлайн-сервис для проверки **скорости загрузки сайта** и его элементов. Очень наглядный отчет, где видно какие элементы сайта перегружены или создают проблемы при загрузке, а страница «Page Analsis» дает развернутые отчеты оскорости отклика сервера, ошибок, отказов и пр.

6. *Анализ шрифтового оформления текста.*

Текст должен быть читабельный, менее чем за 0,5 сек. Человек не должен догадываться, что буква значит и как ее правильно интерпретировать.

Также, полагаю, было бы полезно рассмотреть 10 правил проектирования пользовательского интерфейса, составленных Якобом Нильсеном (один из основателей компании «Nielsen Norman Group», которая занимается проектированием пользовательских интерфейсов.

ПРАВИЛА ПРОЕКТИРОВАНИЯ ИНТЕРФЕЙСА:

1) **Информативность системы** – пользователь всегда должен знать текущий статус приложения.

Онлайн, офлайн, зарегистрированный, не зарегистрированный

2) **Приближенность приложения к реальному миру** – диалог с пользователем должен вестись на понятному ему языку, остерегаясь использования непонятной терминологии.

Визуализация информации должна быть понятной с определенной смысловой нагрузкой

3) **Система должна иметь выходы** – приложение всегда должно иметь «запасные выходы» из любой функциональности, которые пользователь по ошибке запустил.

Окна должны быть всегда закрываемые и с возвратами на предыдущие шаги

4) **Однозначность**. Все термины, функции и понятия должны описываться в едином толковании – у пользователя не должно возникнуть путаницы.

Функционал читабельный только в одном смысле, скачать- значит скачать

5) **Предусмотрительность**. Система должна всячески «оберегать» пользователя от возможных ошибок.

Если создаете диалоговые окна, они должны быть максимально последовательны без точек перехода и заполнения других документов.

6) **Наглядность**. Пользователь не должен ломать голову в попытках понять, что ему нужно делать или пытаясь вспомнить, как он достиг того или иного состояния системы. Возможные манипуляции с программой должны быть постоянно наглядными.

Максимально запоминаемые истории событий, как куда перейти (попасть в добавленные товары, нажать корзину)

7) **Гибкость и эффективность**. Предоставляйте опытным пользователям возможность избегать рутинных действий, и в то же самое время, необходимо скрывать расширение функционала от неопытных.

Расширенный функционал, должен быть сворачиваемый, доп.ф-и можно убрать из основного сценария.

8) **Лаконичность и точность**. Диалоги должны содержать только ту информацию, которую необходимо донести до пользователя, ничего лишнего.

Если мы ожидаем действия от пользователя, ему нужно предоставить как можно меньше выбора т.к. пользователь просто потеряется и не сможет выбрать нужный вариант

9) **Лояльность к ошибкам**. Информация об ошибках должна быть понятной и содержать подсказки к дальнейшим действиям.

Если была ошибка, должно указываться в чем и подсвечиваться где исправить

10) **Постоянная справка**. Как бы информативно не была спроектирована система – она всегда должна содержать раздел справки и документации.

ИНТЕРНАЦИОНАЛИЗАЦИЯ И ЛОКАЛИЗАЦИЯ

«**интернационализация и локализация**» – это процесс придания продукту свойств определенной народности, местности, расположения.

Для успешной реализации продукта во все необходимые нам страны, нужно уделять внимание *не только техническому переводу, но и элементов интерфейса на язык страны-покупателя.*

Пример: В Америке принято сначала отображать значке валюты, а потом само значение, у нас сначала само значение а потом значке валюты.

Само значение слов, технологий, размещения кнопок, текстовых полей, изображений не должно хоть как-то затрагивать чью либо религию или культуру.

Локализация – процесс адаптации программного продукта к языку и культуре клиента.

Данный процесс адаптации включает в себя:

1. Перевод пользовательского интерфейса.
2. Перевод документации.
3. Контроль формата даты и времени.
4. Внимание к денежным единицам.
5. Внимание к правовым особенностям.
6. Раскладка клавиатуры пользователя.
7. Контроль символики и цветов.
8. Толкование текста, символов, знаков.
9. И прочие подобные аспекты.

Что же тогда интернационализация?

Всю что связно языком, переводом. Символов, форматов данных

Интернационализация – более обобщенное понятие, подразумевающее проектирование и реализацию программного продукта или документации таким образом, который максимально упростит локализацию приложения.

Термин интернационализации не обязывает переводить текст программ или документацию на другой язык, *он подразумевает разработку приложений таким образом, который сделает локализацию максимально простой и удобной, а также позволит избежать проблем при интеграции продукта для стран с отличающейся культурой.*

Интернационализация включает в себя:

1. *Создание продукта с учетом возможности кодировки Unicode* (стандарт кодирования, поддерживающий практически все языки мира).
2. *Создание в приложении возможности поддержки элементов, которые невозможно локализовать обычным образом* (вертикальный текст азиатских стран, чтение с права на лево арабских стран и т.д.).

3. *Возможность загрузки локализированных элементов в будущем при желании пользователя.*

как определить функционально эффективный ваш дизайн сайта или нет?

Создайте количественный отчет из юзабилити-тестов

Для начала определитесь с метриками, релевантными для вашего проекта и заказчиков отчета. Их можно разделить на две группы:

- Метрики на уровне системы
- Метрики на уровне задач

Рассмотрим каждую из этих групп.

1. Метрики на уровне системы

Шкала юзабилити системы SUS — это стандартная техника для измерения юзабилити, обладающая двумя преимуществами:

- Надежность. Это стандартизированный опрос, выдержавший проверку временем.
- Юзабилити видно как на ладони. Его легко измерить, понять и передать.

Оценка рассчитывается на основе ответов участников на 10 вопросов. Шкала ответов варьируется от 1 (абсолютно не согласен) до 5 (полностью согласен):

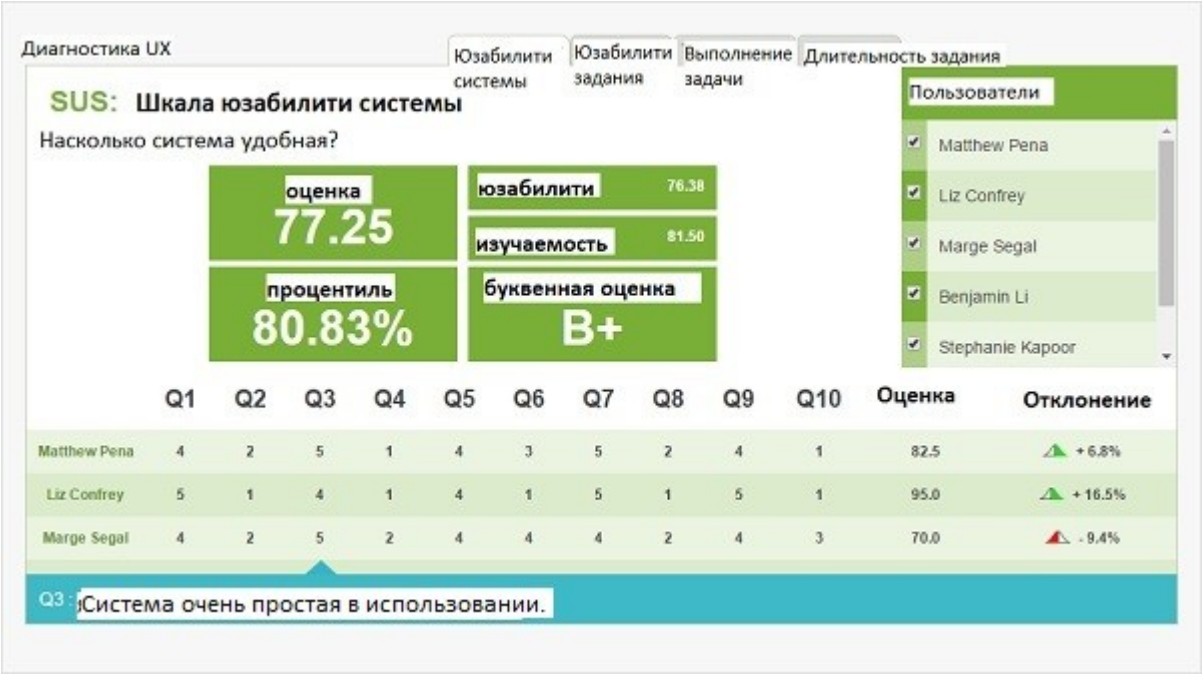
1. Думаю, что часто использовал бы этот сайт/продукт.
2. Сайт/продукт показался мне слишком сложной.
3. Сайт/продукт очень простой в использовании.
4. Думаю, что мне понадобится поддержка технического специалиста, чтобы использовать этот сайт/продукт.
5. Все функции на этом сайте/продукте хорошо интегрированы.
6. На этом сайте/продукте слишком много несоответствий.
7. Большинство людей очень быстро научатся пользоваться этим сайтом/продуктом.
8. Сайт/продукт очень громоздкий в использовании.
9. Я чувствовал себя очень уверенно, используя эту систему.
10. Мне нужно было многое изучить, прежде чем я смог начать работать с этим сайтом/продуктом.

Используйте стандартный инструмент для проведения опроса (SurveyMonkey, Typeform, Survicate, Qualtrics), а затем экспортируйте результаты в электронную таблицу и вычислите среднюю оценку по следующей формуле:

- Для нечетных пунктов — вычтите 1 из ответа участника.
- Для четных пунктов — вычтите ответ участника из 5.
- Суммируйте измененные ответы участника и умножьте сумму на 2,5.

- Вы получите оценку от 0 (F, или плохо) до 100 (A, или хорошо).
- Вычислите среднюю оценку по всем участникам.

Многие инструменты юзабилити-тестирования, такие как, например, TryMyUI, автоматически считают оценку:



Матрица опыта и ожиданий

Еще один способ получить представление о юзабилити на уровне системы — оценить разрыв между тем, насколько легкими представляются участникам задания до выполнения, и тем, какими они оказываются для них по факту.

Для этого попросите участников оценить задание до и после его выполнения по шкале от 1 (очень сложное) до 7 (очень легкое). Если оценка меньше 5, задание считается «не простым».

Сравните ответы «до» (ожидание участника) и «после» (опыт участника) и отобразите их на точечной диаграмме.

Как анализировать ответы:

1. Более легкие, чем ожидалось, задания могут стать сравнительным преимуществом вашего продукта при продвижении его в маркетинговых материалах.

2. Более сложные, чем ожидалось, задания должны получить приоритет в списке изменений.

3. Задания, которые, по ожиданиям участников, должны были быть трудными, и они такими и оказались, дают возможность удивить пользователей.

4. Задания, которые, по ожиданиям участников, должны были быть легкими, и они такими и оказались, следует оставить в покое: они выполняют свою работу.

Для сбора ответов участников используйте стандартные инструменты опроса, а для их визуализации — электронную таблицу. Также вы можете прибегнуть к специальным инструментам, таким как TryMyUI или Searchness, которые сделают все это автоматически.

Вот пример отчета, выполненного с помощью инструмента TryMyUI:

2. Метрики на уровне задач

Показатели выполненных работ и провалов (completion and failure rates)

Есть три типа показателей выполненных работ:

1. Общий показатель выполненных работ. Это процент участников, выполнивших задачу тем или иным образом.

2. Показатель выполненных работ ожидаемым путем. Процент участников, выполнивших задачу, следуя ожидаемому пути.

3. Показатель выполненных работ неожиданным путем. Процент участников, выполнивших задание, используя шаги, которые вы не ожидали или о которых не знали. Это отличный способ раскрыть проблемы юзабилити или их возможности.

Вручную определить шаги на бумаге, просмотреть записи и вычислить все — непростая задача. Используйте для этого инструменты.

Например, в UserTesting.com вы можете использовать функцию Пути клика (click paths), а в Searchness для получения метрик вы визуально определяете шаги:

Противоположность показателя выполненных работ — это частота провалов (failure rate). Посмотрите на частоту провалов на каждом из ваших ожидаемых путей, чтобы определить, где люди испытывают больше всего трудностей.

Первым делом вам следует научиться быстро делать записи, преследуя три цели:

1. Сосредоточить свое внимание на том, что делает и говорит участник, чтобы распознать важное.

2. Зафиксировать основные моменты, чтобы позже можно было легко восстановить их в памяти.

3. Минимизировать объем работ, который необходимо выполнить по завершении сессии (если это модерируемое тестирование) или после просмотра видео (если это немодерируемое тестирование).

Для достижения этих целей используйте технику «логи данных» (data logging).

Для каждой подтемы придумайте свой буквенный код, например:

Б — для багов

Л — для изменений в лице

Н — для негативных высказываний участника

О — для отказа от задания или действия

П — для положительных высказываний участника

Р — для рекомендаций участника

Ю — для проблем с юзабилити, с которыми столкнулся участник

На этом этапе вы также можете добавить второй уровень категоризации. Например, текущая задача или вопрос может стать темой для добавления к вашей записи.

Таким образом, «1Б» будет означать «баг, возникший во время выполнения задачи 1». Наконец, каждая запись должна сопровождаться временной меткой, чтобы потом можно было легко вернуться к нужному моменту в видео.

Потерянность (Lostness)

1. Показатель потерянности измеряет, насколько участник чувствует себя потерянным во время выполнения задания.
2. Разработанная Патрицией Смит (Patricia Smith) в 1996 году метрика учитывает минимальное количество страниц, необходимых для выполнения задачи, а затем сравнивает его с фактическим количеством посещенных участником страниц.
3. Результатом является оценка от 0 до 1. Чем ближе оценка к 1, тем больше участник был потерян (и наоборот). Участник считается потерянным, если оценка равна или выше 0,4.
4. Точная формула выглядит так:

$$L = \sqrt{\left(\frac{N}{S} - 1\right)^2 + \left(\frac{R}{N} - 1\right)^2}$$

L — показатель потерянности

N — количество посещенных уникальных страниц

S — общее количество посещенных страниц (включая

несколько посещений одной и той же страницы). R — минимальное количество страниц, необходимое для успешного выполнения задачи

Заключение

Ценность юзабилити-тестирования частично зависит от того, насколько быстро вы переводите тесты в убедительные данные. Инструменты помогут ускорить этот процесс, но важно знать, как эти инструменты вписываются в него:

1. Используйте технику «логи данных», чтобы ускорить ведение заметок.
2. Держите заметки и ссылки под рукой и делитесь ими в своих качественных отчетах.
3. Автоматизируйте вычисление метрик на уровне системы и задач для оперативного создания количественных отчетов.
4. Проведите краткие индивидуализированные презентации, вдохновляющие на изменения.

Передерживание общепринятых стандартов проектирования

Контрольные вопросы:
14. что такое юзабилити тестирование?
15. Из каких элементов состоит юзабилити тестирование?
16. Проверка юзабилити в чем заключается?
17. Перечислите правила проектирования интерфейса
18. Что такое интернационализация и локализация?
19. Что включает в себя локализация?
20. Что включает в себя интернационализация?

Список использованных источников:

27. Государственный стандарт Российской Федерации. Информационная технология. Сопровождение программных средств
28. Основы маркетинга учебное пособие Суркова Е.В.
29. https://qalight.com.ua/baza-znaniy/internatsionalizatsiya-i-lokalizatsiya/
30. https://qalight.com.ua/baza-znaniy/yuzabiliti/

ЛЕКЦИЯ 45

Тема: Тестирование документации. Структура мануалов. IEEE Standard SRS Template. Введение

Цель: понять основные методы и принцип тестирования документации изучить спецификацию требований к ПО

Виды тестирования:

ТЕСТИРОВАНИЕ ДОКУМЕНТАЦИИ

не все организации уделяют достаточное количество времени разработке стоящей документации… Очень часто нам не счастливится иметь дело с толковым программным продуктом и невзрачным, непонятным и беспомощным документным сопровождением.

Техническая документация–набор документов, используемых при проектировании (конструировании), создании (изготовлении) и использовании (эксплуатации) программного и аппаратного обеспечения.

В целом, документация создается для возможности грамотно и без паники найти выход или решение из любой возникшей проблемной ситуации человеку из самым низким знанием

принципов разработки программного обеспечения. От этого принципа необходимо отталкиваться, продумывая содержание и структуру наших мануалов.

Структура мануалов:

1. ***Полнота и соответствие действительности***. Любая документация должна содержать описание именно той функциональности, которая присутствует в приложении. И данное описание должно касаться абсолютно всей функциональности, а не только наиболее значимой.

Каждая функция, которая прорабатывается на системе должна быть описана, как правильно ее выполнить, в плоть до нажатии скачать документ

Мы должны проверить все заявленные разработчиком ф-и программы.

2. ***Навигация***. И не просто навигация, а удобная навигация. У пользователя никогда не должно возникать проблем с поиском необходимой ему информации. Все деревья файлов, закладки и прочее должны быть на видном месте сразу, как пользователь открывает документ. Алфавитный указатель, поиск – должно присутствовать все, что поможет найти решение или описание проблемы.

Это основные разделы сайта

Если сайт-хранилище данных , то должно быть максимально расширенный функционал поиска, алфавитный, страничный, по авторам, по названию, по новизне…

И это должно быть максимально понятно

3. От пункта выше, вытекает ***структурированность документации***. Все документы должны находится в полном порядке, по разделах. Текст должен быть также с четкой структурой – чтобы можно было в любой момент вспомнить, где остановился или понять, в каком абзаце содержится именно та информация, которая нам необходима.

4. ***Инструкции должны присутствовать везде***. Даже при выполнении абсолютно одинаковых манипуляций с программой – необходимо пошаговое описание действий во всех случаях. Это может быть, как и прямое повторение инструкций, так и ссылка на уже существующие.

Одинаковые действия тоже документируются или ссылаются на уже описанные действия.

5. ***Термины и их значение***. В любой документации может использоваться масса терминов, аббревиатур и сокращений. Каждый из этих сущностей должно иметь свое значение и расшифровку.

6. ***Доступность пользователю***. Документация должна быть максимально понятной для любой целевой аудитории.

В пользовательской документации не должно быть реплик от разработчиков, их специфике разговора.

7. ***документация с учетом иностранных пользователей*** – необходимо привлечение специалистов данного лингвистического сектора, вплоть до носителей языка.

Программный документ — документ, содержащий в зависимости от назначения данные, необходимые для разработки, производства, эксплуатации, сопровождения программы или программного средства

основных типа документации на ПО:

1. архитектурная/проектная — обзор программного обеспечения, включающий описание рабочей среды и принципов, которые должны быть использованы при создании ПО

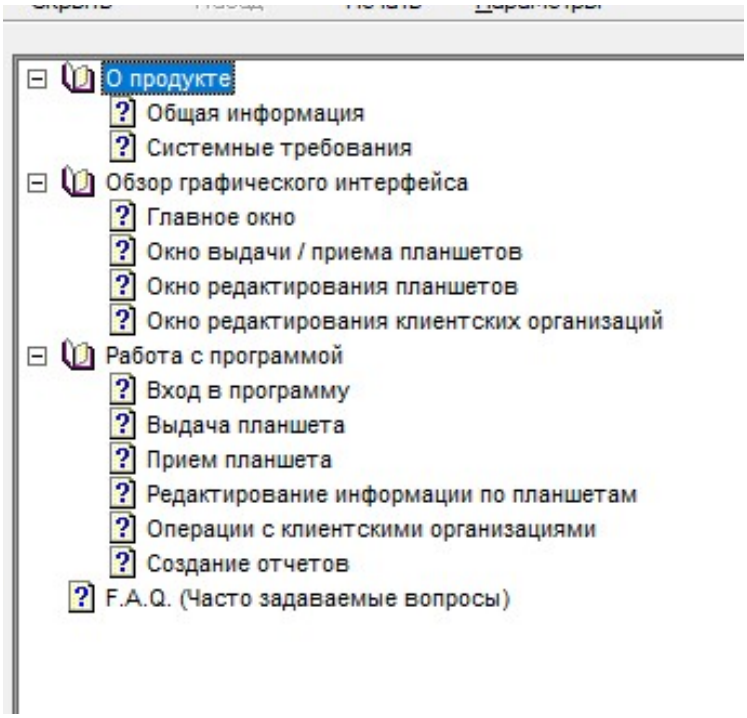
какое ПО и среды будут использоваться при создании определённого проекта

2. техническая — документация на код, алгоритмы, интерфейсы,API

документация для разработчиков. Возможность читать документацию по проекту для сопровождения и разработки, чужим разработчиком

3.пользовательская — руководства для конечных пользователей, администраторов системы и другого персонала

документация для персонала, пользователей



СПЕЦИФИКАЦИЯ ТРЕБОВАНИЙ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

IEEE Standard SRS Template

***Software Requirements Specification*, SRS**

В этом разделе обсуждается каждая основная часть SRS. Эти части упорядочены ниже в виде плана, который может использоваться как образец при написании SRS.

Хотя SRS не обязана в точности следовать этому плану или использовать такие же заголовки частей, хорошая SRS должна включать в себя всю приведенную информацию.

Содержание SRS:

1. Введение
    1. Назначение
    2. Область применения
    3. Определения, акронимы и сокращения
    4. Ссылки
    5. Обзор
2. Общее описание
    1. Позиционирование продукта
    2. Функции продукта
    3. Пользовательские характеристики
    4. Ограничения
    5. Предположения и зависимости
3. Специфические требования
4. Приложения
5. Индекс

. Назначение (1.1 SRS)

В этом подразделе следует:

1. Определить назначение SRS;

2. Задать целевую аудиторию SRS.

Область применения (1.2 SRS)

Этот подраздел должен:

1. Идентифицировать производимый продукт по имени (например, Host DBMS, Report Generator и т.д.);

2. Пояснять, что должен делать программный продукт, а также, при необходимости, чего он не должен делать;

3. Описать применение программного обеспечения, включая выгоды, намерения и цели;

4. Согласовываться со сходными положениями спецификаций верхнего уровня (например, спецификацией требований к системе), если они существуют.

Определения, акронимы и сокращения (1.3 SRS)

Этот подраздел должен представлять определения всех терминов, акронимов и сокращений, необходимых для правильной интерпретации SRS. Эта информация может быть представлена в виде ссылок на одно или более приложений к SRS либо на другие документы.

Ссылки (1.4 SRS)

Данный подраздел должен:

1. Представлять полный перечень документов, на которые есть ссылки где-либо в SRS;
2. Идентифицировать каждый документ по названию, отчетному номеру (если применимо), дате и опубликовавшей организации;
3. Задавать источники, из которых могут быть получены документы, на которые имеются ссылки.

Эта информация может быть представлена в виде ссылки на приложение или другой документ.

Обзор (1.5 SRS)

Данный подраздел должен:

1. Описывать содержимое остальной части SRS;
2. Пояснять организацию SRS.

Контрольные вопросы:
21. что такое техническая документация?
22. Структура мануалов
23. Основные типы документации на ПО?
24. Спецификации требований программного обеспечения
25. Что включает в себя раздел введние?

Список использованных источников:

31. Государственный стандарт Российской Федерации. Информационная технология. Сопровождение программных средств
32. Основы маркетинга учебное пособие Суркова Е.В.
33. https://www.drexplain.ru/help/getting_started.php?ms=AAAA&st=MA%3D%3D&sct=MA%3D%3D&mw=MjUw
34. https://www.elma-bpm.ru/kb/help/Platform/content/Configs_Vkladka_Menu_index.html?ms=AgAAAAAAAAAAAAAAAAAAQ%3D%3D&st=MA%3D%3D&sct=MA%3D%3D&mw=MjQw

ЛЕКЦИЯ 46-51

Темы:

46 IEEE Standard SRS Template. Общее описание: Позиционирование продукта. Функции продукта. Пользовательские характеристики

47 IEEE Standard SRS Template. Общее описание: Ограничения. Предположения и зависимости

48 IEEE Standard SRS Template Специфические требования: Внешние интерфейсы. Функции.

49 Требования к производительности. Логические требования к БД

50 IEEE Standard SRS Template Специфические требования: Атрибуты программной системы. Организация специфических требований.

51 IEEE Standard SRS Template Специфические требования: Дополнительные комментарии. Вспомогательная информация.

Содержание SRS:

6. Введение
    1. Назначение
    2. Область применения
    3. Определения, акронимы и сокращения
    4. Ссылки
    5. Обзор
7. Общее описание
    1. Позиционирование продукта
    2. Функции продукта
    3. Пользовательские характеристики
    4. Ограничения
    5. Предположения и зависимости
8. Специфические требования
9. Приложения
10. Индекс

ОБЩЕЕ ОПИСАНИЕ (РАЗДЕЛ 2 SRS)

Этот раздел SRS должен описывать общие факторы, оказывающие влияние на продукт и требования к нему. В этом разделе не приводятся специфические требования. В нем подготавливается основа для требований, которые детально определяются в Разделе 3 SRS, и приводится информация, облегчающая их понимание.
Этот раздел обычно состоит из шести подразделов:

1. Позиционирование продукта;
2. Функции продукта;
3. Пользовательские характеристики;
4. Ограничения;
5. Предположения и зависимости;
6. Распределение требований.

Позиционирование продукта (2.1 SRS)

Этот подраздел позиционирует продукт среди других связанных продуктов. Если продукт является независимым и полностью самодостаточным, это следует отразить здесь. Если SRS определяет продукт, который является компонентом большей системы, как это часто бывает, то данный подраздел должен связать требования к большей системе с функциональностью программного обеспечения и определить интерфейсы между системой и программным обеспечением.

Могут быть полезными блок-схемы, показывающие основные компоненты большей системы, связи между ними и внешние интерфейсы.
Этот подраздел должен также описывать, как программное обеспечение работает под действием различных ограничений. Например, эти ограничения могут включать:

1. Системные интерфейсы;
2. Пользовательские интерфейсы;
3. Аппаратные интерфейсы;
4. Программные интерфейсы;
5. Коммуникационные интерфейсы;
6. Память;
7. Операции;
8. Требования к адаптируемости на месте.

2.1.1. Системные интерфейсы

Следует перечислить все системные интерфейсы и идентифицировать функциональность программного обеспечения для выполнения требований к системе, а также описание интерфейса для соответствия системе.

2.1.2. Пользовательские интерфейсы

Следует задать:

1. Логические характеристики каждого интерфейса между программным продуктом и его пользователями. Сюда входят конфигурационные характеристики (например, требуемые экранные формы, страницы или раскладки окон, содержимое всех отчетов или меню, а также доступность программируемых функциональных клавиш), необходимые для выполнения программных требований.
2. Все аспекты оптимизации интерфейса с персоналом, который должен использовать систему. Они могут просто включать список, как система должна выглядеть с точки зрения пользователя, а как не должна. Например, может быть требование наличия опции полных или кратких сообщений об ошибках. Подобно всем остальным, эти требования должны быть верифицируемыми, например: «оператор 4-го разряда должен научиться выполнять действие X за Z минут после часового обучения», а не «оператор должен выполнять действие X». (Это также можно задать в «Системных атрибутах программного обеспечения» в разделе под названием «Простота использования»).

2.1.3. Аппаратные интерфейсы

Здесь следует задать логические характеристики каждого интерфейса между программным продуктом и аппаратными компонентами системы. Сюда входят конфигурационные характеристики (число портов, набор инструкций и т.д.). Также освещаются такие моменты, как: какие устройства поддерживаются, как они поддерживаются, а также протоколы. Например, поддержка терминала может задавать поддержку полноэкранного интерфейса, в противоположность построчному вводу.

2.1.4. Программные интерфейсы

Следует задать использование прочих необходимых программных продуктов (например, система управления данными, операционная система или математический пакет), а также интерфейсы с другими прикладными системами (например, связь между системой приема оплаты счетов и общей бухгалтерской системой). Для каждого необходимого программного продукта следует предоставить следующую информацию:

- Название;
- Мнемоника;
- Номер спецификации;
- Номер версии;
- Источник.

Для каждого интерфейса следует предоставить следующую информацию:

- Обсуждение назначения интерфейсного программного обеспечения с точки зрения программного продукта.
- Определение интерфейса в терминах содержания и формата сообщений. Не обязательно детально описывать хорошо документированные интерфейсы, но требуется ссылка на документ, определяющий требуемый интерфейс.

2.1.5. Коммуникационные интерфейсы

Следует задать различные интерфейсы коммуникаций: протоколы локальных сетей

2.1.6. Ограничения по памяти

Следует задать все значимые характеристики и ограничения, касающиеся первичной и вторичной памяти.

2.1.7. Операции

Следует задать обычные и специфические операции, которые требуются пользователю, например:

1. Различные модели операций в организации пользователя (например, операции, инициируемые пользователем);
2. Периоды интерактивных операций и периоды операций, не требующих ручного вмешательства;
3. Функции поддержки обработки данных;
4. Операции резервного копирования и восстановления.

Примечание. Иногда они задаются как часть раздела «Пользовательские Интерфейсы».

2.1.8. Требования к адаптации на мест

Следует:

1. Определить требования ко всем данным или последовательностям инициализации, специфичным для данного места, задачи или режима работы (например, таблицы значений, безопасные пределы и т.д.);
2. Задать особенности, относящиеся к месту или задаче, которые следует модифицировать с целью адаптации программного обеспечения к конкретной инсталляции.

Функции продукта (2.2 SRS)

В этом подразделе SRS следует представить сводку основных функций, выполняемых системой. Например, SRS для бухгалтерской программы может посвятить эту часть работе со

счетами, обслуживанию клиентов и подготовке платежных поручений, не вдаваясь в обширную детализацию этих функций.
Иногда сводка функций, необходимых для данной части, берется прямо из соответствующего раздела спецификации верхнего уровня (если она есть), которая размещает некоторые функции в программном продукте. Заметим для ясности, что:

1. Функции должны быть организованы таким образом, чтобы сделать перечень функций понятным потребителю или другим читателям при первом прочтении документа.
2. Можно использовать текстовые или графические методики для представления различных функций и отношений между ними. Подобные диаграммы не должны представлять реализацию продукта, а лишь показывать логические взаимосвязи между переменными.

Пользовательские характеристики (2.3 SRS)

Этот подраздел SRS должен описывать общие характеристики предполагаемых пользователей, включая уровень образования, опыт и техническую грамотность. В нем не следует устанавливать специфические требования, но следует привести причины, по которым некоторые специфические требования заданы далее в Разделе 3 SRS.

Ограничения (2.4 SRS)

В этом подразделе должны быть приведены общие описания всего того, что может ограничить действия разработчика. Они включают:

1. Правовые вопросы;
2. Аппаратные ограничения (например, требования к длительности сигналов);
3. Интерфейсы с другими приложениями;
4. Параллельные операции;
5. Функции аудита;
6. Функции управления;
7. Языковые ограничения высшего порядка;
8. Протоколы синхронизации сигналов (например, XON-XOFF, ACK-NACK);
9. Требования к надежности;
10. Критичность приложения;
11. Соображения безопасности и секретности.

Предположения и зависимости (2.5 SRS)

В этом подразделе следует перечислить все факторы, которые влияют на требования, устанавливаемые SRS. Эти факторы не являются проектными ограничениями, а скорее относятся к их изменениям, которые могут повлиять на требования SRS. Например, может быть сделано предположение, что некоторая операционная система будет доступна для оборудования, на которое ориентирован программный продукт. Если в действительности операционная система недоступна, потребуется соответствующее изменение SRS.

Распределение требований (2.6 SRS)

Этот подраздел должен представлять требования, которые могут быть отложены до будущих версий системы.

СПЕЦИФИЧЕСКИЕ ТРЕБОВАНИЯ (РАЗДЕЛ 3 SRS)

Этот раздел SRS должен содержать все требования к программному обеспечению на уровне детализации, достаточном, чтобы позволить разработчикам создать систему, удовлетворяющую этим ограничениям, а тестерам – проверить, удовлетворяет ли система этим ограничениям. В данном разделе каждое требование должно быть ориентировано на пользователей, операторов или другие системы, внешние по отношению к данной. Эти требования должны включать минимальное описание для каждого ввода в систему, каждого

ответа системы, а также всех функций, выполняемых системой в ответ на ввод или для поддержки вывода. Поскольку зачастую эта часть SRS является самой большой и наиболее важной, применяются следующие принципы:

1. Специфические требования должны соответствовать характеристикам, описанным в 4.3.
2. Специфические требования должны снабжаться перекрестными ссылками на ранние документы.
3. Все требования должны быть уникально идентифицируемыми.
4. Следует уделить особое внимание организации требований с целью достижения максимальной читабельности.

Перед изучением способов организации требований полезно рассмотреть различные элементы, включающие в себя требования, которые описаны в подразделах с 3.1 по 3.7.

3.1.Внешние интерфейсы

Здесь должно быть детальное описание всех входных и выходных данных программной системы. Оно должно дополнять описания интерфейсов из 5.2 и не содержать повторяющейся информации.

Оно должно иметь формат и содержание, как указано ниже:

1. Имя элемента;
2. Описание назначения;
3. Источник входных данных или получатель выходных;
4. допустимый диапазон, точность и отклонения;
5. Единицы измерения;
6. Временные диаграммы;
7. Взаимосвязи с другими входами/выходами;
8. Форматы и организация экранов;
9. Форматы и организация окон;
10. Форматы данных;
11. Форматы команд;
12. Завершающие сообщения.

3.2. Функции

Функциональные требования должны определять фундаментальные действия, которые должны выполняться программным обеспечением для приема и обработки входных данных, а также обработки и вывода выходных данных. Они обычно перечисляются в виде предложений, начинающихся со слов «Система должна…».

Они включают:

1. Проверку допустимости входных значений;
2. Точный порядок действий;
3. Реакцию на нештатные ситуации, включающие:
    1. Переполнение;
    2. Коммуникационные проблемы;
    3. Обработку ошибок и восстановление;
4. Влияние параметров;
5. Взаимосвязь между входными и выходными данными, включая:
    1. Порядок ввода/вывода;
    2. Формулы преобразования входных данных в выходные

Может оказаться удобным разделение функциональных требований на подфункции или подпроцессы. Это не значит, что программный проект будет разделен таким же образом.

3.3. Требования к производительности

Этот подраздел должен задавать как статические, так и динамические численные требования, предъявляемые в целом к программному обеспечению или к взаимодействию человека с программой. Статические численные требования могут включать следующие:

1. Число поддерживаемых терминалов;
2. Число одновременно поддерживаемых пользователей;
3. Объем и тип обрабатываемой информации.

Статические численные требования иногда оформляются в виде отдельного раздела.
Динамические численные требования могут включать, например, число транзакций и задач, или объем данных, обрабатываемых в некоторый период данных в условиях как нормальной, так и пиковой нагрузки.
Все эти требования следует формулировать в терминах измеримых величин.
Например:
95% транзакций должны обрабатываться менее чем за 1 секунду
вместо:
Оператор не должен ждать, пока завершится транзакция.
Примечание. Численные границы, применимые к отдельной функции, обычно задаются в описании данной функции, в подразделе обработки.

3.4. Логические требования к базе данных

Здесь следует задать логические требования к информации, которая должна размещаться в базе данных. Они могут включать следующие:

- Типы информации, используемой различными функциями;
- Частоту использования;
- Возможность доступа;
- Сущности и отношения между ними;
- Ограничения целостности;
- Требования к хранению данных.

3.5. Ограничения проектирования

Здесь задаются ограничения проектирования, налагаемые другими стандартами, аппаратурой и т.д.

3.5.1. Соответствие стандартам

Этот подраздел должен задавать ограничения, вытекающие из существующих стандартов и правил. Они могут включать следующее:

1. Форматы отчетов;
2. Именование данных;
3. Бухгалтерские процедуры;
4. Протоколирование работы.

Например, требования к программному обеспечению по трассировке вычислительных действий. Подобные трассировки необходимы для некоторых приложений для выполнения требований правовых или финансовых стандартов. Требование протоколирования работы может, например, устанавливать, что при изменениях в платежной ведомости прежнее и новое значения должны записываться в файл трассировки.

3.6. Атрибуты программной системы

Некоторые атрибуты программного обеспечения могут служить требованиями. Важно, чтобы требуемые атрибуты были заданы таким образом, чтобы можно было объективно проверить выполнение данного требования. В подразделах с 3.6.1 по 3.6.5 приведен список примеров.

3.6.1. Надежность

Следует перечислить факторы, необходимые для установления требуемого уровня надежности программного обеспечения.

3.6.2. Доступность

Следует перечислить факторы, призванные гарантировать определенный уровень доступности системы, такие как контрольные точки, восстановление и перезапуск.

3.6.3. Безопасность

Следует перечислить факторы, защищающие программное обеспечение от случайного или злонамеренного доступа, использования, модификации, разрушения или разглашения. Специфические требования в этой области включают необходимость:

1. Использования криптографии;
2. Хранение логов или истории;
3. Назначать некоторые функции различным модулям;
4. Ограничивать коммуникации между некоторыми областями программы;
5. Проверять целостность данных для критических переменных.

3.6.4. Поддерживаемость

Следует перечислить атрибуты программного обеспечения, относящиеся к легкости поддержки самого программного обеспечения. Это могут быть некоторые требования, относящиеся к модульности, интерфейсам, сложности и т.д. Не следует помещать здесь требования лишь потому, что принято считать их хорошим тоном разработки.

3.6.5. Переносимость

Следует перечислить атрибуты программного обеспечения, касающиеся легкости переноса программного обеспечения на другие компьютеры и/или операционные системы. Они могут включать следующее:

1. Доля компонентов с машинно-зависимым кодом;
2. Доля машинно-зависимого кода;
3. Использование испытанного переносимого языка;
4. Использование особенного компилятора или подмножества языка;
5. Использование особенной операционной системы.

3.7. Организация специфических требований

Для любой нетривиальной системы детальные требования обширно разрастаются. Поэтому рекомендуется тщательный подход к их организации в виде, оптимальном для понимания. Не существует единой организации, оптимальной для всех систем. Различные классы систем придерживаются различной организации требований в Разделе 3 SRS. Некоторые способы организации описаны в подразделах с 3.7.1 по 3.7.7.

3.7.1. Режим системы

Поведение некоторых систем кардинально различается в зависимости от режима системы. Например, система управления может иметь различные наборы функций для разных режимов: обучение, нормальный или аварийный. При организации этого раздела по режимам, можно использовать планы A.1 или A.2. Выбор определяется тем, зависят ли от режима интерфейсы и производительность.

3.7.2. Класс пользователя

Некоторые системы обеспечивают различные наборы функций для разных классов пользователей. Например, система управления лифтом предоставляет различные возможности пассажирам, обслуживающему персоналу и пожарным. При организации этого раздела по классам пользователей следует использовать план A.3.

3.7.3. Объекты

Объекты – это сущности реального мира, которые имеют соответствие в системе. Например, в системе мониторинга пациентов в число объектов входят пациенты, датчики, медсестры, кабинеты, врачи, медикаменты и т.д. С каждым объектом связаны наборы атрибутов (данного объекта) и функций (выполняемых данным объектом). Эти функции

называются также сервисами, методами или процессами. При организации этого раздела по объектам следует использовать план A.4. Заметим, что наборы объектов могут иметь общие атрибуты и сервисы. Они группируются вместе как классы.

3.7.4. Функциональные возможности

Функциональная возможность – это внешний сервис системы, который может потребовать последовательность ввода данных для достижения желаемого результата. Например, в телефонной системе функциональные возможности включают местный вызов, переадресацию звонка и конференцию. Обычно функциональные возможности описывают в виде последовательности пар запрос-ответ. При организации этого раздела в соответствии с функциональными возможностями следует использовать план A.5.

3.7.5. Внешние воздействия

Некоторые системы могут быть лучше организованы путем описания их функций в терминах внешних воздействий. Например, функции системы автоматического приземления самолета могут быть разбиты на секции для потери тяги, порывов ветра, неожиданной смены курса, избыточной вертикальной скорости и т.д. При организации этого раздела в соответствии с внешними воздействиями следует использовать план A.6.

3.7.6. Отклики

Некоторые системы могут быть лучше организованы путем описания всех функций, поддерживающих генерацию откликов. Например, функции системы управления персоналом могут быть разбиты на секции, соответствующие всем функциям, связанным с генерацией чеков оплаты, всем функциям, связанным с генерацией текущего списка сотрудников, и т.д. Следует использовать план A.6, в котором все внешние воздействия заменены откликами.

3.7.7. Функциональная иерархия

Если ни одна из вышеперечисленных организационных схем не подходит, общая функциональность может быть организована в виде иерархии функций, организованных в соответствии с входными данными, выходными данными или внутренними данными. Могут быть использованы диаграммы потоков данных и словари данных, чтобы показать взаимосвязи между функциями и данными. При организации этого раздела в соответствии с функциональной иерархией следует использовать план A.7.

3.8. Дополнительные комментарии

В процессе работы над SRS могут оказаться применимыми более одной организационной схемы из перечисленных в 3.7.7. В таких случаях организуйте специфические требования по нескольким иерархиям, приспособленным под специфические нужды системы, для которой разрабатываются спецификации. Например, см. A.8 как пример организации, комбинирующей классы пользователей и функциональные возможности. Все дополнительные требования могут быть размещены в отдельном разделе в конце SRS.

Имеется множество нотаций, методик и автоматизированных инструментов для поддержки документирования требований. В основном они полезны в части организационной функции. Например, при организации по режимам могут оказаться полезными конечные автоматы диаграммы состояний; при организации по объектам может оказаться полезным объектно-ориентированный анализ; при организации по функциональным возможностям могут оказаться полезными последовательности запрос-ответ, а при организации по функциональной иерархии могут пригодиться диаграммы потоков данных и словари данных.

В любом из планов, описанных с A.1 по A.8, разделы с именем «Функциональное требование i» могут быть описаны на естественном языке, на псевдокоде, на языке описания систем либо в виде четырех подразделов с именами «Введение», «Входные данные», «Обработка» и «Выходные данные».

4. ВСПОМОГАТЕЛЬНАЯ ИНФОРМАЦИЯ

Вспомогательная информация делает SRS более удобочитаемой. Она включает следующее:

- Оглавление;
- Индекс;
- Приложения.

4.1. Оглавление и индекс

Оглавление и индекс крайне важны и должны следовать общим принципам композиции.

4.2. Приложения

Приложения не всегда рассматриваются как часть SRS и не всегда необходимы. Они могут включать:

- Примеры форматов входных и выходных данных, описания примеров ценового анализа или результаты пользовательских обзоров;
- Вспомогательная и дополнительная информация, которая может помочь читателю SRS;
- Описание проблемы, решаемой программным обеспечением;
- Специальные инструкции по упаковке для кода и носителей для соответствия требованиям по безопасности, экспорту начальной загрузке и т.д.

Если приложения включены в состав SRS, следует явно указать, являются ли они частью требований.

Контрольные вопросы:

26. Опишите из чего состоит общий раздел документации?
27. Опешите специфический раздел требований
28. Опешите раздел приложений
29. Опишите раздел индекса

Список использованных источников:

35. Государственный стандарт Российской Федерации. Информационная технология. Сопровождение программных средств
36. Основы маркетинга учебное пособие Суркова Е.В.
37. Рекомендуемая стандартом IEEE 830структура SRS