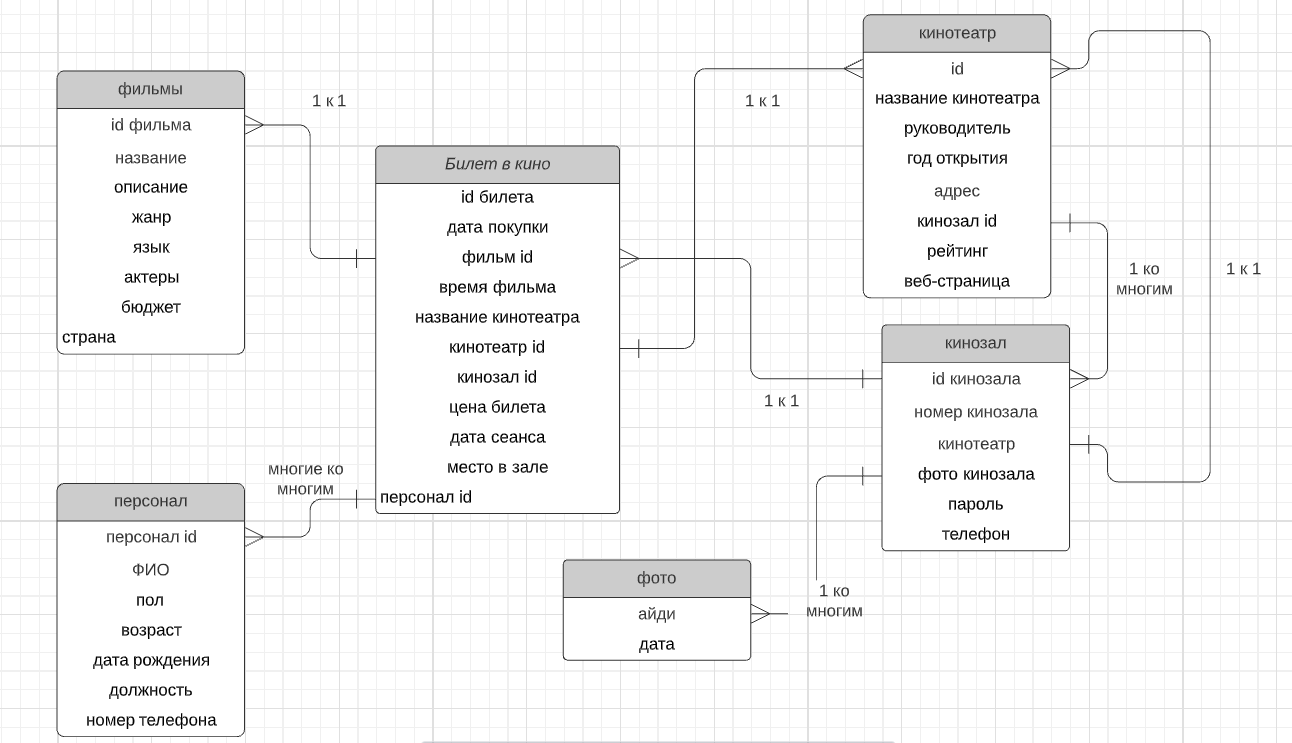
**№1**

**1 вопрос (20 баллов): Рассмотрим проект веб сайта по продаже билетов.**

**Вы участник проектной команды отвечающий за разработку схемы**

**данных. Необходимо составить ER-диаграмму, не менее 6 сущностей, не**

**менее 40 атрибутов, минимум один пример связи 1:1.**

****

**2 вопрос (20 баллов): Из чего состоит и зачем нужна стратегия тестирования. Приведите несколько примеров**

Зачем?

* Для организации процесса в условиях ограниченных ресурсов.
* Для того, чтобы все участники проекта понимали роль тестирования, что оно может дать. Чтобы у всех были равные ожидания и понимание, что происходит в области контроля качества, а также для выявления возможных проблем, которые неизбежно станут очевидными в процессе обсуждения

Из чего состоит?

* Технологии и инструменты тестирования.
* Условия завершения и успешного выполнения тестов.

Например, можно решить, что после успешного выполнения 95% тестов можно перейти к приемке продукта. Другим примером условия может служить охват кода. Например, в системах безопасности может требоваться стопроцентный охват кода при тестировании.

* Особые обстоятельства, влияющие на требования к ресурсам и график тестирования, например:
  + тестирование всех интерфейсов ко внешним системам
  + имитация угрозы безопасности или физического повреждения системы

**3 вопрос (20 баллов): Паттерны проектирования (GoF Patterns). Общие**

**понятия, назначение, плюсы и минусы.**

Общие понятия

Паттерн проектирования — это часто встречаемое решение определённой проблемы при проектировании архитектуры программ.

Зачем?

* проверенные решения
* стандартизация кода
* общий программистский стиль

Виды паттернов + назначение:

* поведенческие - Эти паттерны решают задачи эффективного и безопасного взаимодействия между объектами программы
* структурные - Эти паттерны отвечают за построение удобных в поддержке иерархий классов
* порождающие - Эти паттерны отвечают за удобное и безопасное создание новых объектов или даже целых семейств объектов

Преимущества от применения паттернов:

* **повышает эффективность труда** отдельных исполнителей и всей группы в целом.
* позволяет также создавать **более модифицируемое и гибкое** программное обеспечение
* помогают **общему пониманию** основных принципов объектно-ориентированного проектирования
* позволяют разработчику находить проектные решения для сложных проблем, **не создавая громоздкой иерархии** наследования классов.

Недостатки паттернов:

* **неэффективные решения -** разработчики используют паттерны как догмы и не подстраивают под проект
* **неоправданное применение** - неопытные прогеры начинают их использовать везде, даже там, где можно было обойтись кодом попроще
* **“костыль” для слабых ЯП** - применяют паттерны для тех проектов, которые написаны на языке программирования с низким уровнем абстракции

**№ 2**

**1 вопрос (20 баллов): Написать 15 требований к веб-сайту заказа билетов**

**в кино. Разделить на функциональные и качественные, дополнительно**

**разделить согласно модели Кано. Требования должны удовлетворять**

**критериям качества.**

Функциональные требования:

* Покупка билетов через веб-сайт
* Отправка чека на электронную почту после покупки билета
* Возможность регистрации и входа в учетную запись для просмотра истории заказов
* Возможность забронировать кинозал для компании на определенное время
* Просмотр расписания фильмов в кинотеатрах на требуемую дату
* Получение информации о проводимых акциях и закрытых показах
* Оформление подарочной карты онлайн
* Возможность оставлять отзывы на фильм и кинотеатр
* Возможность обсуждать фильмы с другими зарегистрированными участниками
* Получение ответов на вопросы в онлайн режиме

Качественные требования:

* Адаптивность

Сайт должен корректно отображаться во всех основных браузерах. Сайт должен быть адаптивным, т.е. доступным к удобному просмотру с любых экранов (ПК, ноутбук, смартфон и т. д.). Сайт должен быть оптимизирован для просмотра с основными типами разрешений. Элементы управления должны быть сгруппированы однотипно – горизонтально либо вертикально – на всех страницах.

* Отказоустойчивость

Дублирование всех критичных подсистем, что позволяет приложению функционировать, даже если из строя выходит один из компонентов.

* Дизайн

При разработке сайта должны быть использованы преимущественно светлые и контрастные цветовые решения.

В дизайне сайта не должны присутствовать:

- мелькающие баннеры;

- много сливающегося текста;

- тёмные и агрессивные цветовые сочетания и графические решения

* Требования к безопасности

Требуется защитить исходный код общей части сайта. Требуется разграничение доступа. Перехват данных на уровне протокола tcp возможен. На уровне СУБД должно быть реализовано разграничение доступа к данным в БД

* Требования к надежности

Система может быть недоступна не более чем 24 часа в год. Резервирование данных осуществляет хостинг-провайдер. У администратора сайта должна быть возможность выгрузить и загрузить копию сайта.

### Основные:

* Покупка билетов через веб-сайт
* Возможность регистрации и входа в учетную запись для просмотра истории заказов
* Отправка чека на электронную почту после покупки билета
* Просмотр расписания фильмов в кинотеатрах на требуемую дату
* Получение информации о проводимых акциях и закрытых показах

### Желаемые:

* Возможность забронировать кинозал для компании на определенное время
* Оформление подарочной карты онлайн
* Возможность оставлять отзывы на фильм и кинотеатр

### Восхищающие:

* Возможность обсуждать фильмы с другими зарегистрированными участниками
* Получение ответов на вопросы в онлайн режиме

**2 вопрос (20 баллов): Жизненный цикл ПО. Понятие жизненного цикла**

**ПО, этапы, их описание в терминологии различных стандартов (примеры).**

Понятие

Жизненный цикл программного обеспечения (Software Life Cycle Model) — это период времени, который начинается с момента принятия решения о создании программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Все проекты могут иметь следующую структуру жизненного цикла:

* начало проекта;
* организация и подготовка;
* выполнение работ проекта;
* завершение проекта.

Стандарты и их этапы:

Основной нормативный документ, регламентирующий ЖЦ ПО – международный стандарт ***ISO/IEC 12207.*** Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, выполняемые во время создания ПО. В стандарте ISO/IEC 12207 описаны пять основных процессов жизненного цикла программного обеспечения:

1. процесс приобретения определяет действия предприятия - покупателя информационной системы, программного продукта или службы программного обеспечения;
2. процесс поставки определяет действия предприятия-поставщика по снабжению

покупателя информационной системой, программным продуктом или службы программного обеспечения;

1. процесс разработки определяет действия предприятия-разработчика, который разрабатывает принципы построения программного изделия и собственно программный продукт;
2. процесс функционирования определяет действия предприятия-оператора,

обслуживающего систему в целом. Сюда входят консультация пользователей, получение обратной связи и т.д.;

1. процесс сопровождения определяет действия персонала, обеспечивающего

сопровождение программного продукта, т.е. управление модификацией программного

продукта, поддержку текущего состояния и функциональной пригодности, установку и

удаление.

Методика **Oracle CDM**:

1. определение требование - моделирование и анализ процессов, описывающих деятельность организации, для которой создается ИС
2. анализ - проводится для формулирования детальных требований к ИС
3. проектирование - требования преобразуются в формальные спецификации ИС
4. реализация - написание и тестирование приложений
5. эксплуатация

Жизненный цикл процесса создания АСУ согласно **ГОСТ 34** (ГОСТ 34.601-90) включает

следующие стадии:

* Формирование требований к АС
* Разработка концепции АС
* Техническое задание
* Эскизный проект
* Технический проект
* Рабочая документация
* Ввод в действие
* Сопровождение АС

**3 вопрос (20 баллов): Процесс разработки. Архитектура.**

**Документирование и распространение сведений. Группы**

**заинтересованных лиц..**

Процесс разработки

Программным процессом (software process) называются действия по организации, нормированию и разработке программного обеспечения.

Архитектура

Архитектура это объективная реальность. Любое существующее приложение имеет архитектуру. Даже если о ней никто не задумывался. Даже если это выглядит как абсолютный хаос. Даже если все делалось «как получится».

Архитектура это одно из средств организации коммуникаций с заинтересованными лицами

Зачем нужна архитектура?

* Архитектура определяет ограничения реализации
* Архитектура сдерживает или способствует реализации атрибутов качества системы
* Архитектура облегчает анализ изменений и их организацию
* Архитектура позволяет более точно рассчитывать стоимость и сроки

Документирование и распространение сведений

Для того, чтобы архитектура стала основой проекта, ее суть должна быть четко и недвусмысленно донесена до всех заинтересованных лиц.

Для достижения этих целей, документирование архитектуры должно быть информативным и понятным людям различных профессий. Заинтересованные лица преследуют различные цели – требуют разной направленности документов, разных уровней детализации и разных трактовок. Предполагать, что один и тот же документ будет одинаково воспринят всеми «заказчиками», наивно. Необходимо стремиться к тому, чтобы любое заинтересованное лицо смогло быстро найти необходимую именно ему информацию.

Группы заинтересованных лиц

Для того, чтобы понять как структурировать документацию и повысить удобство ее использования, следует выделить **группы заинтересованных лиц**. Для каждого проекта список групп может быть своим, типовые:

* Спонсоры и менеджеры проекта
* Представляющие заказчика разработчики требований
* Архитектор(-ы)
* Тестировщики и сборщики
* Специалисты по сопровождению

**№ 3**

**1 вопрос (20 баллов): Написать 15 требований к веб-сайту заказа билетов**

**в кино. Разделить на функциональные и качественные, дополнительно**

**разделить согласно модели Кано. Требования должны удовлетворять**

**критериям качества.**

**2 вопрос (20 баллов): Опишите отличия между проектным управлением**

**pmbok и работой по agile (scrum)**

Начнем с того, что pmbok и agile сравнить не стоит, так как их инструменты предназначены для разных задач.

Идея создания подхода **pmbok** заложилась в строительной, оборонной и аэрокосмической области. И при работе с физическими объектами результат понятен в самом начале проекта, то есть можно распланировать, выстроить некую виртуальную модель или чертеж, а потом перейти к фактическому строительству. Но когда pmbok начали применять в сфере IT, то поняли, что такой подход не так хорошо работает с виртуальными объектами, потому что базовая аксиома **pmbok** о том, что можно заранее спланировать и перейти к реализации не работает. Так как в самом начале пытаются договориться с заказчиками и выстроить план реализации проекта, но чуть позже сталкиваются с тем, что эти договоренности часто меняются.

В 2001 году появилось понятие agile, в практиках которого отражены идеи, позволяющие непрерывно пересматривать представление о создаваемом продукте и двигать в процессе разработки быстро.

Выбор подхода зависит от степени понятности результата или способа его получения.

преимущества pmbok

* Главное преимущество PMBOK – возможность предвосхитить значительную долю проблем в проекте, решить их на бумаге дешево и быстро еще до выхода в производство.
* С точки зрения сроков реализации, бюджета и качества результатов использование PMBOK дает определенную уверенность и прозрачность. (опять таки там где высокая степень понятности). Agile-практиками тоже можно решить понятную задачу. Но это будет дольше и дороже, а при очередной переработке могут возникнуть еще и проблемы с качеством.

преимущества agile

* Преимущества Agile заключаются в том, что команды в таких проектах гибкие и юркие, не боятся изменчивости, готовы развернуться и быстро перестроиться.
* Одной из самых привлекательных черт в Agile-практиках является отношение к человеку. Оно отличается от классического проектного подхода, где людей считают ресурсом. И в этом – серьезный недостаток PMBOK.

**3 вопрос (20 баллов): Паттерны проектирования. Порождающие паттерны**

**(на выбранном примере). Описать компоненты паттерна, их**

**взаимодействие, привести собственный пример. Желательно**

**проиллюстрировать схемой вызова.**

Паттерн проектирования — это часто встречаемое решение определённой проблемы при проектировании архитектуры программ.

Порождающие паттерны (беспокоятся о гибком создании объектов без внесения в программу лишних зависимостей) - Фабричный метод, абстрактная фабрика, строитель, прототип.

Пример паттерна:

Абстрактная фабрика - паттерн проектирования, который предоставляет интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.

Компоненты и из взаимодействие:

* абстрактный продукт (объявляет интерфейс продуктов) -> конкурентный продукт
* абстрактная фабрика (объявляет интерфейс фабрики) -> конкретная фабрика

Собственный пример:

Допустим, мы это автомобильный холдинг, все, что мы делаем, это скупаем заводы автомобилей и потом выпускаем и продаем машины. Тогда абстрактной фабрикой и будет холдинг, которая будет прдставлена интерфейсом:

***interface CarsFactory()***

а заводы будут классами имплементирующие этот интерфейс:

***class Nissan implements CarsFactory***

***class Lada implements CarsFactory***

Едем дальше, решаем, что наши предприятия будут выпускать машины с двумя типами кузова (седан и купе), оке тогда мы должны прописать эти методы в нашем интерфейсе, чтобы наследующие классы были обязаны реализовать их в себе

***interface CarsFactory {***

***public function createSedan();***

***public function cretaeCoupe();***

***}***

***class Nissan implements CarsFactory {***

***public function createSedan() {***

***return new NissanSedan();***

***}***

***public function createCoupe() {***

***return new NissanCoupe();***

***}***

***}***

***class Lada implements CarsFactory {***

***public function createSedan() {***

***return new LadaSedan();***

***}***

***public function createCoupe() {***

***return new LadaCoupe();***

***}***

***}***

Но при этом типом возвращаемого объекта методов будет тип седана или купе, который должен быть общий, значит будет еще два интерфейса (седан и купе), которые будут наследоваться классами конкретных автомобилей

***interface Coupe {}***

***interface Sedan {}***

***// точно также для лады классы написать***

***class NissanCoupe implements Coupe {***

***// ...***

***}***

***class NissanSedan implements Sedan {***

***// ...***

***}***

Ну и получается, что у нас есть фабрика фабрик, поэтому когда к нам придет клиент и скажет, что хочет машину, мы такие да легко у нас есть переменная factory типа CarsFactory испрашиваем какую марку хотите, клиент такой ладу хочу

***factory = new LadaFactory();***

И уточняем купе или седан, нам говорят купе:

***factory.createCoupe();***

вуа-ля машина готова

**№ 4**

**1 вопрос (20 баллов): Представить планирование на 3 спринта с 15**

**пользовательскими историями. На историях должны быть названия,**

**оценка в стори-пойнтах согласно ряду Фибоначчи. У спринтов должны**

**быть емкость и скорость. Расчет емкости должен быть обоснован.**

**Тематика разработки - веб-сайт заказа билетов в кино.**

Заранее условимся, что 1 storypoint - 1 час работы 1-го разработчика

Каждый спринт длится 2 недели

**1 спринт**

Я как пользователь хочу видеть текущие фильмы, которые идут в кинотеатре и которые выйдут в скором времени (3)

Я как пользователь хочу иметь возможность просматривать расписание фильмов в любое время, в любом кинотеатре Сети (21)

Я как пользователь хочу просмотреть цены на сайте на билеты и еду (5)

Я как пользователь хочу просмотреть контакты, чтобы узнать адрес любого кинотеатра Сети (8)

Я как пользователь хочу купить билет в кино (13)

Всего: 3+5+8+13+21 = 50 часов

**2 спринт**

Я как пользователь хочу иметь возможность заказа еды в кинотеатр и домой через сторонний сервис (21)

Я как пользователь хочу иметь возможность оформления подарочный карты (13)

Я как пользователь хочу просматривать информацию об акциях проводимых кинотеатром (3)

Я как пользователь хочу иметь возможность просматривать баланс подарочной карты (8)

Я как пользователь хочу просматривать информацию о каждом кинотеатре, фотографии залов(5)

Всего: 3+5+8+13+21 = 50 часов

**3 спринт**

Я как пользователь хочу получать уведомления на почту об акциях, закрытых показах (3)

Я как пользователь хочу общаться с другими участниками Сети, обсуждать фильмы прям на платформе (8)

Я как пользователь хочу зарегистрироваться для получения скидок и покупки билетов на аккаунте (13)

Я как пользователь хочу войти в личный кабинет, чтобы иметь возможность смотреть историю своих покупок (21)

Я как пользователь хочу оставлять отзывы просмотренным фильмам, кинотеатрам Сети (5)

Всего: 3+5+8+13+21 = 50 часов

**Нам необходимо 50+50+50 = 150 рабочих часов => что необходимо привлечь двух разработчиков, которые работают по 40 часов в неделю, а так как спринт длится 2 недели, то этого времени должно хватить, так как у нас будет 40\*2\*2 = 160 часов**

**2 вопрос (20 баллов): Жизненный цикл ПО. Понятие жизненного цикла**

**ПО, этапы, их описание в терминологии различных стандартов (примеры).**

Понятие

Жизненный цикл программного обеспечения (Software Life Cycle Model) — это период времени, который начинается с момента принятия решения о создании программного продукта и заканчивается в момент его полного изъятия из эксплуатации.

Все проекты могут иметь следующую структуру жизненного цикла:

* начало проекта;
* организация и подготовка;
* выполнение работ проекта;
* завершение проекта.

Стандарты и их этапы:

Основной нормативный документ, регламентирующий ЖЦ ПО – международный стандарт ***ISO/IEC 12207.*** Он определяет структуру ЖЦ, содержащую процессы, действия и задачи, выполняемые во время создания ПО. В стандарте ISO/IEC 12207 описаны пять основных процессов жизненного цикла программного обеспечения:

1. процесс приобретения определяет действия предприятия - покупателя информационной системы, программного продукта или службы программного обеспечения;
2. процесс поставки определяет действия предприятия-поставщика по снабжению

покупателя информационной системой, программным продуктом или службы программного обеспечения;

1. процесс разработки определяет действия предприятия-разработчика, который разрабатывает принципы построения программного изделия и собственно программный продукт;
2. процесс функционирования определяет действия предприятия-оператора,

обслуживающего систему в целом. Сюда входят консультация пользователей, получение обратной связи и т.д.;

1. процесс сопровождения определяет действия персонала, обеспечивающего

сопровождение программного продукта, т.е. управление модификацией программного

продукта, поддержку текущего состояния и функциональной пригодности, установку и

удаление.

Методика **Oracle CDM**:

1. определение требование - моделирование и анализ процессовБ описывающих деятельность организации, для которой создается ИС
2. анализ - проволится для формулирования детальных требований к ИС
3. проектирование - требования преобразуются в формальные спицификации ИС
4. реализация - написание и тестирование приложений
5. эксплуатация

Жизненный цикл процесса создания АСУ согласно **ГОСТ 34** (ГОСТ 34.601-90) включает

следующие стадии:

* Формирование требований к АС
* Разработка концепции АС
* Техническое задание
* Эскизный проект
* Технический проект
* Рабочая документация
* Ввод в действие
* Сопровождение АС

**3 вопрос (20 баллов): Паттерны проектирования. Поведенческие паттерны**

**(на выбранном примере). Описать компоненты паттерна, их**

**взаимодействие, привести собственный пример. Желательно**

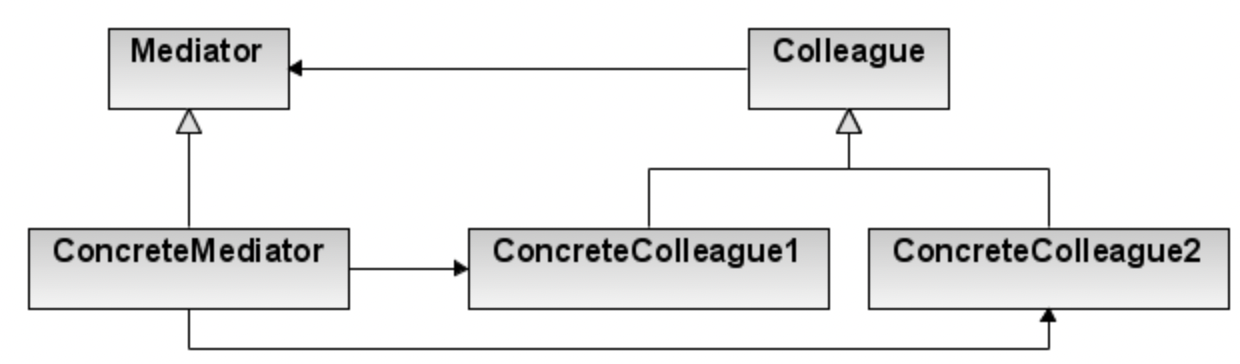
**проиллюстрировать схемой вызова.**

Паттерн проектирования — это часто встречаемое решение определённой проблемы при проектировании архитектуры программ.

Поведенческие паттерны (заботятся об эффективной коммуникации между объектами) - Посредник, наблюдатель, снимок, состояние, итератор, команда, цепочка обязанностей

Рассмотрим на примере посредника:

Паттерн Посредник заставляет объекты общаться не напрямую друг с другом, а через отдельный объект-посредник, который знает, кому нужно перенаправить тот или иной запрос. Благодаря этому, компоненты системы будут зависеть только от посредника, а не от десятков других компонентов.



Допустим у нас есть коллеги и нам нужно реализовать их взаимодействие через посредник.

объяснение на языке людей: есть абстрактный класс посредника, который представляет канал общения между коллегами и определяет, какие методы должен будет реализовать конкретный посредник (). есть класс абстрактного работника, который будет наследоваться классом конкретного работника.

***// Абстрактный канал общения между коллегами***

***abstract class Mediator***

***{***

***abstract public function send($message, Colleague $colleague);***

***}***

***// Абстрактный работник, который не против пообщаться со своими коллегами***

***abstract class Colleague***

***{***

***protected $mediator;***

***public function \_\_construct(Mediator $mediator)***

***{***

***$this->mediator = $mediator;***

***}***

***public function send($message)***

***{***

***$this->mediator->send($message, $this);***

***}***

***/\*\****

***\* Обработка полученного сообщения реализуется каждым конкретным***

***\* наследником***

***\* @param string message получаемое сообщение***

***\*/***

***abstract public function notify($message);***

***}***

***// Конкретный канал общения между коллегами***

***class MSTeams extends Mediator***

***{***

***/\*\****

***\* @var ConcreteColleague1***

***\*/***

***private $colleague1;***

***/\*\****

***\* @var ConcreteColleague2***

***\*/***

***private $colleague2;***

***public function setColleague1(ConcreteColleague1 $colleague)***

***{***

***$this->colleague1 = $colleague;***

***}***

***public function setColleague2(ConcreteColleague2 $colleague)***

***{***

***$this->colleague2 = $colleague;***

***}***

***public function send($message, Colleague $colleague)***

***{***

***switch ($colleague) {***

***case $this->colleague1:***

***$this->colleague2->notify($message);***

***break;***

***case $this->colleague2:***

***$this->colleague1->notify($message);***

***}***

***}***

***}***

***//коллега 1***

***class Tom extends Colleague***

***{***

***public function notify($message)***

***{***

***//…***

***}***

***//коллега 2***

***class Paul extends Colleague***

***{***

***public function notify($message)***

***{***

***//…***

***}***

***}***

***$mediator = new MSTeams();***

***$Tom = new Tom($mediator);***

***$Paul = new Paul($mediator);***

***$mediator->setColleague1($collegue1);***

***$mediator->setColleague2($collegue2);***

***$Tom->send('Пожуй гавна');***

***$Paul->send('сам пожуй');***

**№ 5**

**1 вопрос (20 баллов): Представить планирование на 3 спринта с 15**

**пользовательскими историями. На историях должны быть названия,**

**оценка в стори-пойнтах согласно ряду Фибоначчи. У спринтов должны**

**быть емкость и скорость. Расчет емкости должен быть обоснован.**

**Тематика разработки – портал IT поддержки.**

Заранее условимся, что 1 storypoint - 1 час работы 1-го разработчика

Каждый спринт длится 2 недели

**1 спринт**

Я как пользователь, хочу отправлять запросы на портал ИТ-поддержки (21)

Я как пользователь, хочу получать письмо на почту после отправки заявки, чтобы знать, что моя заявка была принята на рассмотрение (5)

Я как пользователь, хочу при заполнении заявки загружать фото, чтобы в моей заявки была фотография, более подробно описывающая мою проблему/ошибку (3)

Я как пользователь, хочу выбирать причину обращения, для ускорения обработки запроса (8)

Я как пользователь, хочу повысить срочность моего обращения (13)

Всего: 3+5+8+13+21 = 50 часов

**2 спринт**

Я как пользователь, хочу зарегистрироваться в системе, чтобы видеть все мои заявки (21)

Я как пользователь, хочу иметь “плитки” для создания быстрого обращения в системе (5)

Я как пользователь, хочу получать уведомление на почту если система ИТ-поддержки не работает и возможное время решения данной проблемы (3)

Я как пользователь, хочу при заполнении заявки автоматически подтягивались ФИО, рабочий номер и табельный номер (8)

Я как пользователь, хочу чтобы мои заявки автоматически присваивались к УЗ, без авторизации в системе (по имени/фамилии) (13)

Всего: 3+5+8+13+21 = 50 часов

**3 спринт**

Я как пользователь хочу иметь мобильное приложения для создания заявок (21)

Я как пользователь, хочу иметь возможность оформлять пропуска через портал (5)

Я как пользователь, хочу создавать заявки-инциденты, который будут решаться в ускоренном режиме (3)

Я как пользователь, хочу бронировать переговорные комнаты в офисе через ИТ-портал (8)

Я как пользователь, хочу иметь возможность спросить какую-то информацию в онлайн-чате и тут же получить ответ (13)

Всего: 3+5+8+13+21 = 50 часов

**Нам необходимо 50+50+50 = 150 рабочих часов => что необходимо привлечь двух разработчиков, которые работают по 40 часов в неделю, а так как спринт длится 2 недели, то этого времени должно хватить, так как у нас будет 40\*2\*2 = 160 часов**

**2 вопрос (20 баллов): Соглашения об уровне сервиса (SLA). Зачем нужны**

**SLA? Приведите состав SLA.**

Cоглашение об уровне обслуживания (Service Level Agreement или SLA) — внешний документ (существующий между заказчиком и исполнителем), описывающий параметры предоставляемой услуги.

Зачем нужен?

Простыми словами, SLA – это договор, в котором подробно описаны предоставляемые услуги, их качество, время реагирования на заявку и ее исполнение.

Состав SLA

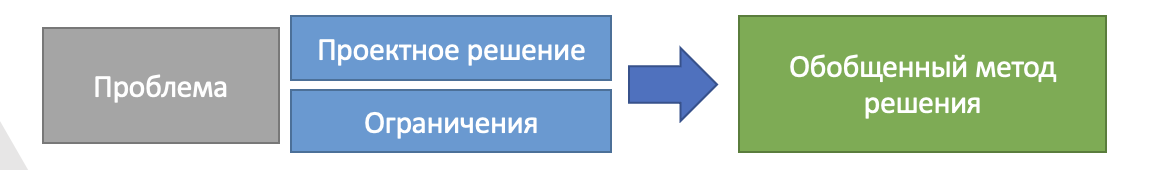
Типовое соглашение об уровне сервиса (SLA) должно содержать следующие разделы:

* Контактные данные сторон, вовлеченных в соглашение, и срок его действия
* Описание услуги
* Техническая информация об услуге
* Поддерживаемое оператором абонентское оборудование
* Уровень и качество обслуживания
* Средства мониторинга и отчеты SLA
* Центр технической поддержки
* Механизм резервирования и восстановления услуги
* Нарушения обслуживания и компенсации
* Тарифы и выставление счетов
* Процесс улучшения SLA
* Порядок прекращения оказания услуги

**3 вопрос (20 баллов): Стили и шаблоны проектирования. Понятие**

**архитектурного стиля и архитектурного шаблона**

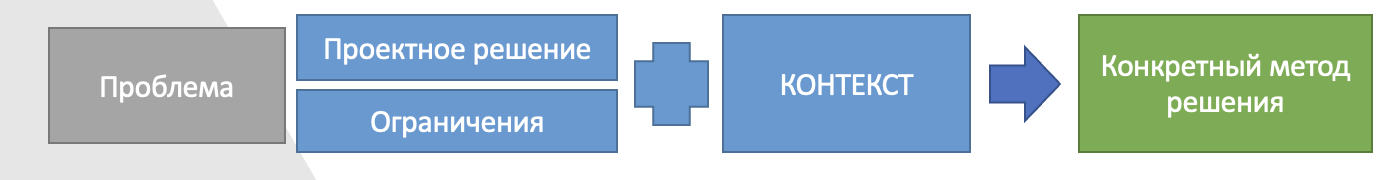
Архитектурный стиль – набор типов элементов и связей между ними, а также ограничений использования, направленный на решение проблемы



Примеры стилей:



Архитектурный шаблон – основа для построения структурных схем программных систем. Описывает набор предопределенных подсистем, их ответственность и правила организации связи между ними

****

**№ 6**

**1 вопрос (20 баллов): Написать 15 пользовательский историй к веб-сайту**

**заказа билетов в кино с указанием критериев приемки и разделением**

**минимум на две роли.**

1. Я как пользователь хочу видеть текущие фильмы, которые идут в кинотеатре и которые выйдут в скором времени

Критерий приемки: после перехода на пункт меню “Главное” будут отображены текущие фильмы, которые идут в кинотеатре и которые выйдут в скором времени

1. Я как пользователь хочу иметь возможность просматривать расписание фильмов в любое время, в любом кинотеатре Сети

критерий приемки: В разделе меню “Расписание” можно посмотреть расписание фильмов на любой день в любом кинотеатре Сети

1. Я как пользователь хочу просмотреть цены на сайте на билеты и еду

критерий приемки: В разделе Цены на билеты и Заказ еды можно просмотреть всю информацию по ценам для билетов и еды

1. Я как пользователь хочу купить билет в кино

критерий приемки: В разделе меню “Покупка билетов” есть возможность купить билет в кино на выбранный сеанс

1. Я как пользователь хочу иметь возможность просматривать баланс подарочной карты

критерий приемки: В разделе меню “Подарочные карты” можно просматривать баланс подарочной карты и использовать в своих целях

1. Я как пользователь хочу войти в личный кабинет, чтобы иметь возможность смотреть историю своих покупок

критерий приемки: Есть возможность регистрации и входа в личный кабинет пользователя для просмотра истории покупок

1. Я как пользователь хочу оставлять отзывы просмотренным фильмам, кинотеатрам Сети

критерий приемки: Есть возможность оставлять отзывы после похода в кино на фильм, кинотеатр.

1. Как администратор сайта, я хочу иметь инструменты по изменению внешнего оформления сайта после входа в УЗ админа

критерий приемки: после входа в УЗ админа на сайте и перехода на любой пункт меню появляются инструменты позволяющие поменять наполнение сайта и внешнее оформление

1. Как администратор сайта, я хочу получать информацию о состоянии веб-сайта. Если происходит сбой, меня уведомляют различными способами.

критерий приемки: если происходит сбой в работе системы, приходит уведомление в системе мониторинга и т.п

1. Как администратор сайта, я хочу получать информацию о количестве посетителей сайта в виде развернутой аналитики.

критерий приемки: Каждую неделю должна приходить информация о количестве посетителей. аналитика должна быть представлена и в текстовом и в графическом виде

1. Как Администратор сайта, я хочу иметь возможность блокировать пользователей системы, если они нарушают пользовательское соглашение.

критерий приемки: Предоставляется “Админская панель” где можно заблокировать пользователя, который нарушил правила сайта

1. Я как пользователь хочу общаться с другими участниками Сети, обсуждать фильмы прям на платформе

критерий приемки: Отдельный пункт меню “Чат пользователей”, где пользователи могут общаться, делиться впечатлениями и т.п.

**еще 3 придумайте сами ыы**

**2 вопрос (20 баллов): Проект. Управление проектами. Понятие проектного офиса, типы проектных офисов. Программы и портфели.**

Проект — это временное предприятие, направленное на создание уникального продукта, услуги или результата.

Проект — это законченное мероприятие по чему-либо

Проект может создать:

* продукт, представляющий собой компонент другого изделия, улучшение изделия или конечное изделие;
* услугу или способность предоставлять услугу (например, бизнес-функция, поддерживающая производство или дистрибуцию);
* улучшение существующей линейки продуктов или услуг (например, проект по методике «шести сигм» (Six Sigma), предпринятый для уменьшения дефектов);
* результат, такой как конечный результат или документ (например, исследовательский проект приносит новые знания, которые можно использовать для определения наличия тенденции или пользы какого-либо нового процесса для общества).

Проектное управление является одним из двух основных видов управления разработкой (наряду с гибким управлением Agile).

Методологии проектного управления:

* PMBOK
* PRINCE2

Не существует идеальной системы управления проектами, подходящей для каждого из видов проектов. Также не существует системы, которая бы подходила каждому руководителю и была удобна для всех членов команды.

Управление проектом - приложение знаний, навыков, инструментов и методов к работам проекта для удовлетворения требований, предъявляемых к проекту.

Управление проектом осуществляется посредством должного применения и интеграции логически сгруппированных 47 процессов управления проектом, объединенных в 5 групп процессов.

• Инициация = идея: процессы, выполняемые для определения нового проекта или новой фазы существующего проекта путем получения авторизации на начало проекта или фазы;

• Планирование: процессы, требуемые для установления содержания работ, уточнения целей и определения направления действий, требуемых для достижения целей проекта;

• Исполнение: процессы, применяемые для выполнения работ, указанных в плане управления проектом, с целью соответствия спецификациям проекта;

• Мониторинг и контроль: процессы, требуемые для отслеживания, анализа, а также регулирования исполнения проекта; выявления областей, требующих внесения изменений в план; и инициирования соответствующих изменений,

• Закрытие: процессы, выполняемые для завершения всех операций в рамках всех групп процессов в целях формального закрытия проекта или фазы.

**Программа** — ряд связанных друг с другом проектов, подпрограмм и операций программы, управление которыми координируется для получения выгод, которые были бы недоступны при управлении ими по отдельности.

**Портфель** — это набор проектов,программ, подпортфелей и элементов операционной деятельности, управляемых как группа с целью достижения стратегических целей.

**Офис управления проектами/Проектный офис —** организационная структура,

стандартизирующая процессы руководства проектами и способствующая обмену

ресурсами, методологиями, инструментами и методами. Сфера ответственности ОУП

может варьироваться от оказания поддержки в управлении проектами до прямого

управления одним или более проектами.

Типы :

• Поддерживающий. Играет консультативную роль, предоставляя шаблоны, лучшие практики, обучение, доступ к информации и уроки, извлеченные из других проектов. Данный тип служит в качестве хранилища проекта. Степень контроля со стороны Проектного офиса низкая.

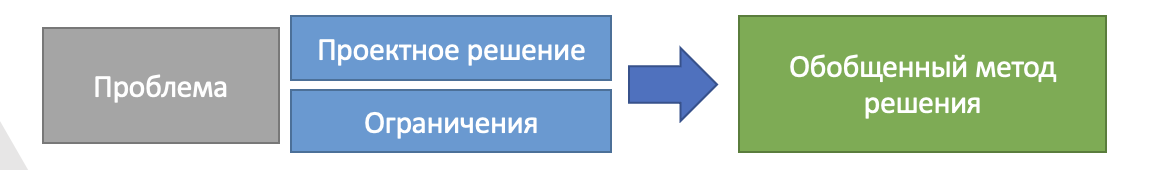
• Контролирующий. Предоставляет поддержку и требует соответствия требованиям с помощью различных средств. Соответствие может предполагать адаптацию структур или методологий управления проектами, использование специфических шаблонов, форм и инструментов или соответствие требованиям руководства. Степень контроля со стороны Проектного офиса средняя.

• . Контролирует проекты путем непосредственного управления данными проектами. Степень контроля со стороны Проектного офиса высокая.

**3 вопрос (20 баллов): Стили и шаблоны проектирования. Понятие**

**архитектурного стиля и архитектурного шаблона.**

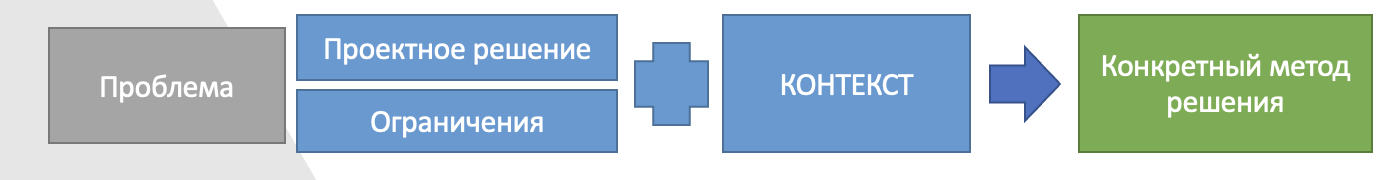
Архитектурный стиль – набор типов элементов и связей между ними, а также ограничений использования, направленный на решение проблемы



Примеры стилей:



Архитектурный шаблон – основа для построения структурных схем программных систем. Описывает набор предопределенных подсистем, их ответственность и правила организации связи между ними

****

**№ 7**

**1 вопрос (20 баллов): Написать базовые положения соглашения об уровне**

**обслуживания портала ИТ-поддержки со стороны внешнего подрядчика.**

**Выделить три уровня приоритета, критерии определения приоритета и**

**временные характеристики.**

Базовые положения

1. название системы

“портал ИТ-поддержки”

1. Стек технологий

php vue.js postgreSQL ElasticSearch (што уходно написать можно)

1. Границы действия SLA
   1. территориальные границы – москва и МО
   2. временные границы - Портал поддерживается в режиме рабочей 40-часовой недели, пока не будет выделено больше ресурсов для круглосуточной поддержки. То есть график рабочих часов следующий: пн-пт 11:00-19:45
   3. Функциональные (какие функции системы обслуживаются) - обслуживаются функции системы, с которыми взаимодействуют клиенты
2. Способ обращения и куда обращаться

HelpDesk (он же 1й уровень поддержки), принимающий обращения от всех пользователей компании по телефону, почте и интернету, оформляющий обращения в инциденты и переводящий инциденты в специализированные группы поддержки 2-го уровня;

Группа поддержки 2-го уровня из разработчиков, которые умеют исправить код/данные в системе и привлечь вендора базового ПО в случае необходимости.

1. Перечень услуг

* Оказание консультаций пользователям ИС Заказчика, решение проблем пользователей, связанных с функционированием ИС Заказчика и ее составных частей;
* Поддержание работоспособности и доступности основных сетевых служб заказчика на уровне, указанном в данном соглашении.
* Проведение мониторинга элементов и подсистем ИС заказчика, а также профилактических работ, направленных на поддержание работоспособности ИС заказчика в целом;

1. Приоритеты

* Высший приоритет (1)

Максимальный срок удаленной реакции не должен превышать четырех часов. Если реакция предполагает проведение работ на площадке заказчика, то срок реакции не должен превышать одного дня.

* Средний приоритет (2)

Максимальный срок удаленной реакции не должен превышать шести часов. Если реакция предполагает проведение работ на площадке заказчика, то срок реакции не должен превышать двух дней.

* Низкий приоритет (3)

Максимальный срок удаленной реакции не должен превышать одного дня. Если реакция предполагает проведение работ на площадке заказчика, то срок реакции не должен превышать 7 дней.

1. Определение приоритета и временные характеристики

| Влияние на работу пользователей | Количество вовлеченных пользователей | | |
| --- | --- | --- | --- |
|  | Один | Менее 10 | Более 10 |
| Полная неработоспособность | средний | высокий | высокий |
| Неработоспособность одного или нескольких приложений | низкий | средний | высокий |
| Неработоспособность некоторых функций приложений | низкий | средний | средний |

**2 вопрос (20 баллов): Гибкие принципы управление. Agile – идеи и принципы.**

Существуют два вида управления:

1. Проектное управление;
2. Гибкое управление (Agile).

Рассматриваемые методологии гибкого управления:

• Scrum;

• Kanban.

Не существует идеальной системы управления проектами, подходящей для каждого из видов проектов. Также не существует системы, которая бы подходила каждому руководителю и была удобна для всех членов команды.

**Идеи:**

* **Люди и взаимодействие** важнее процессов и инструментов
* **Работающий продукт** важнее исчерпывающей документации
* **Сотрудничество с заказчиком** важнее согласования условий контракта
* **Готовность к изменениям** важнее следования первоначальному плану

Не отрицая важности того, что справа, мы всё-таки больше ценим то, что слева.

**Принципы:**

1. Удовлетворение клиента за счет ранней и бесперебойной поставки ценного программного обеспечения;

2. Приветствие изменений требований даже в конце разработки (это может повысить конкурентоспособность полученного продукта);

3. Частая поставка рабочего программного обеспечения (каждый месяц или неделю или ещё чаще);

4. Тесное, ежедневное общение заказчика с разработчиками на протяжении всего проекта;

5. Проектом занимаются мотивированные личности, которые обеспечены нужными условиями работы, поддержкой и доверием;

6. Рекомендуемый метод передачи информации — личный разговор (лицом к лицу);

7. Работающее программное обеспечение — лучший измеритель прогресса;

8. Спонсоры, и пользователи должны иметь возможность поддерживать постоянный темп на неопределённый срок;

9. Постоянное внимание улучшению технического мастерства и удобному дизайну;

10. Простота — искусство не делать лишней работы;

11. Лучшие технические требования, дизайн и архитектура получаются у самоорганизованной команды;

12. Постоянная адаптация к изменяющимся обстоятельствам. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

**3 вопрос (20 баллов): Атрибуты качества и сценарии - подробное описание на примере одного из типа сценариев (Модифицируемость, Ремонтопригодность, Производительность, Безопасность...)**

Причинами, потребовавшими изменения систем чаще всего становятся:

* Трудности сопровождения
* Проблемы масштабирования
* Низкая производительность
* Недостаточная стойкость к атакам
* Неготовность к переносу

Вопросы воплощения атрибутов качества решаются в периоды проектирования, реализации и развертывания. Не существует ни одного атрибута качества, который бы зависел только от какого-то одного этапа программного цикла.

* **Практичность** имеет как архитектурные, так и не архитектурные аспекты. Примеры последних – обеспечение ясности и простоты применения пользовательского интерфейса. Что выбрать: переключатель или флажок? Какая схема размещения на экране наиболее интуитивна? Какая гарнитура шрифта отображается четче других? Эти вещи имеют большое значение для конечного пользователя но при этом являются локальными решениями стадии реализации
* **Модифицируемость** определяется тем, каким образом происходит разделение функциональности (архитектурный аспект) и методик кодирования в рамках отдельного модуля (аспект тактической реализации).
* **Производительность** также зависит от архитектурных и неархитектурных факторов. Она зависит от объемов информации, передающейся между компонентами, от распределения функциональности между компонентами, управления общими ресурсами – все это архитектурные аспекты. Но на производительность влияют и выбранные для реализации функциональности алгоритмы, и качество их реализации (кодирования) – а это неархитектурные аспекты.
* Атрибуты качества системы. Такие атрибуты как готовность, модифицируемость, производительность, безопасность, контролепригодность и практичность
* Коммерческие атрибуты качества (например, Time-to-Market, срок вывода продукта на рынок). Имеет смысл рассматривать только те из атрибутов, реализация которых определяется архитектурой системы
* Атрибуты качества самой архитектуры (например, концептуальная целостность), которые влияют на другие качества – например, модифицируемость

Реализация ряда атрибутов качества по отдельности в сколько-либо сложной системе невозможна. Реализация одного атрибута всегда каким-то образом (положительно или отрицательно) влияет на реализацию других. К примеру, безопасность и надежность, как правило, взаимно конфликтуют

У всех проведенных атрибутов качества есть три основных недостатка:

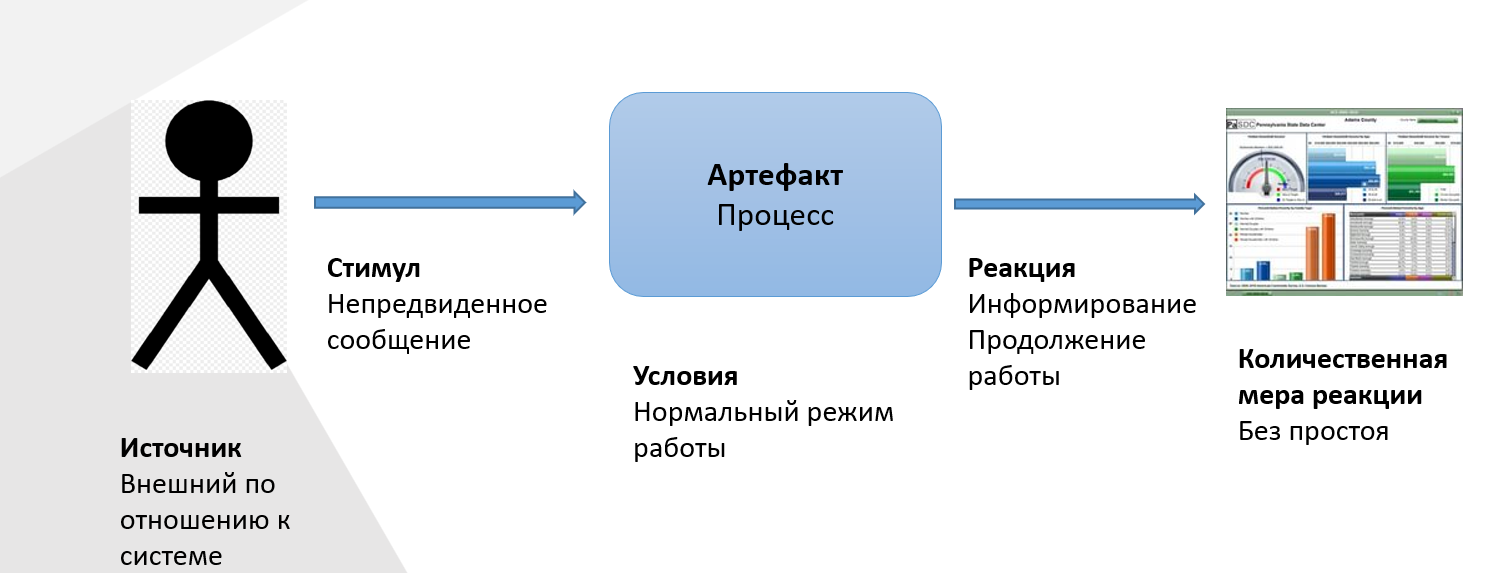
* Все предложенные определения атрибутов непрактичны. Сказать что система будет модифицируемой, - значит не сказать ничего. Любая система модифицируема в отношении одного ряда изменений и немодифицируема в отношении другого такого ряда.
* Частая неоднозначность отнесения аспектов к тому или иному атрибуту качества. Например, аспектом чего является отказ системы: готовности, безопасности или практичности?
* У разных сообществ свои словари. Одно и то же явление специалисты по производительности называют «событием», специалисты по готовности – «сбоем» а специалисты по практичности – «данными, вводимыми пользователем»

Первые две проблемы (непрактичные определения и несогласованность классификаций) решаются путем применения для составления характеристик сценариев атрибутов качества (quality attribute scenarios)

**Сценарием атрибута качества** называется требование, путем выполнения которого этот атрибут реализуется. Сценарий состоит из шести элементов

* Источник стимула. Это некий субъект (человек, вычислительная система) который порождает символ
* Стимул. Стимулом называется наблюдаемое в системе явление, требующее к себе внимания
* Условия. Стимул возникает при конкретных условиях. К примеру, система может находиться в состоянии перегрузки или исполняться в обычном режиме
* Артефакт. Объектом воздействия стимула является некий артефакт. В этом качестве выступает система в целом или ее отдельный элемент
* Реакция. Реакция – это действие, предпринятое в ответ на появление стимула
* Количественная мера реакции. Предпринимаемые в ответ на стимул действия должны быть измеримы – только тогда соответствие требованию можно проверить

**Пример сценария готовности:**

****

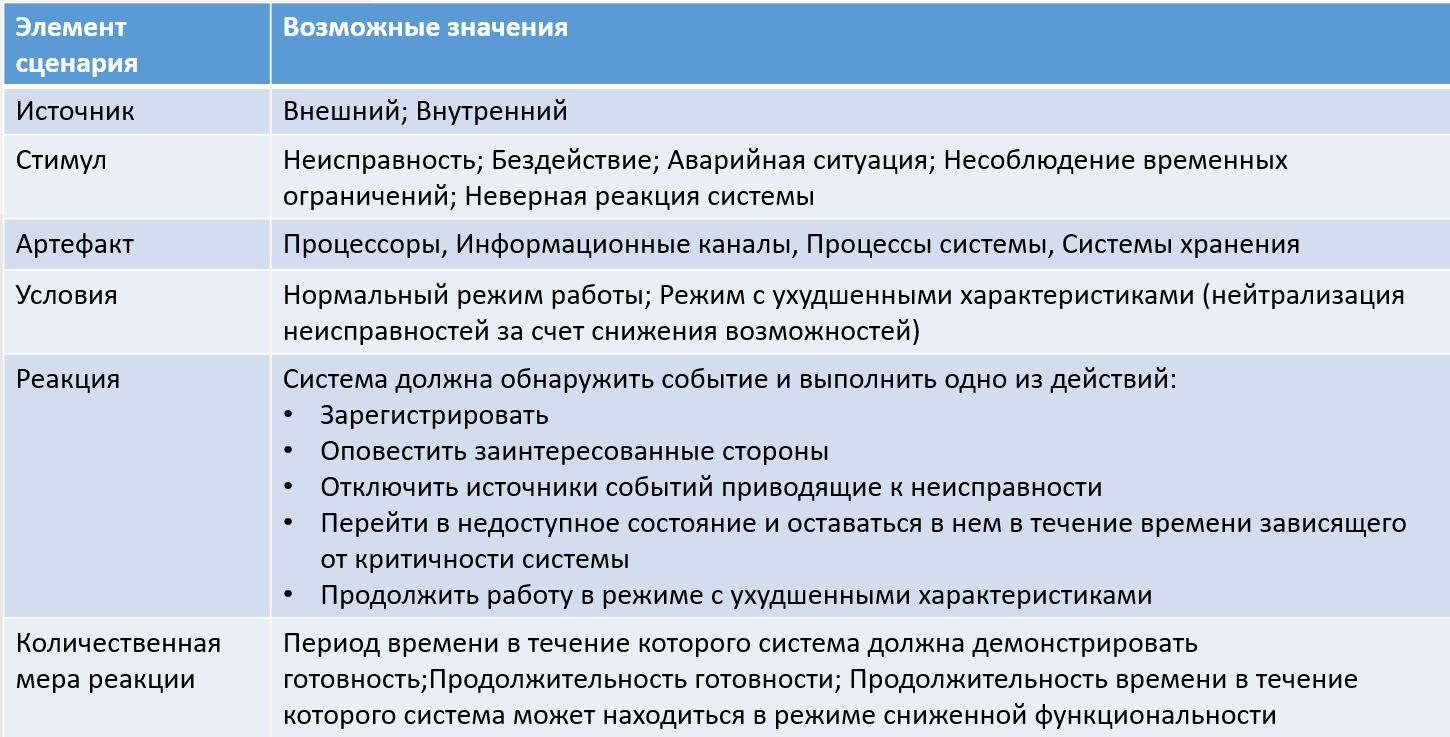
Готовность отражает ситуацию отказа системы и его возможных последствий. Отказом системы называется положение, при котором система теряет способность предоставления услуг, заявленных в ее спецификации. Отказ заметен для пользователей системы, в качестве которых могут выступать люди или другие системы.

Следует четко различать Отказ (failure) и неисправность (fault). Если неисправность не устранена или не замаскирована, она может перейти в состояние отказа. Пользователь системы может наблюдать ситуацию отказа, но не неисправность. Когда неисправность становится обозримой для пользователя, она становится отказом.

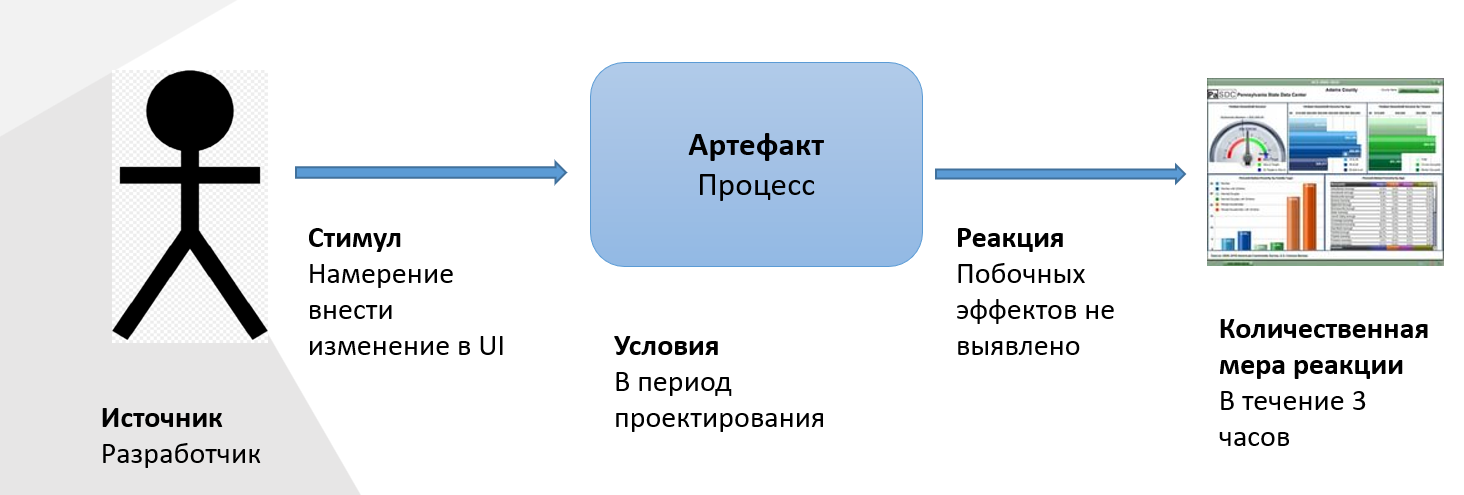
После отказа системы на первый план выходит вопрос о том, сколько времени потребуется на его устранение. Из того, что отказ системы обозрим для пользователей, можно сделать вывод, что период устранения отказа эквивалентен периоду, по истечении которого отказ перестает быть обозримым.

В количественном выражении готовность это вероятность функционирования системы на отрезках времени когда в этом есть необходимость.

****

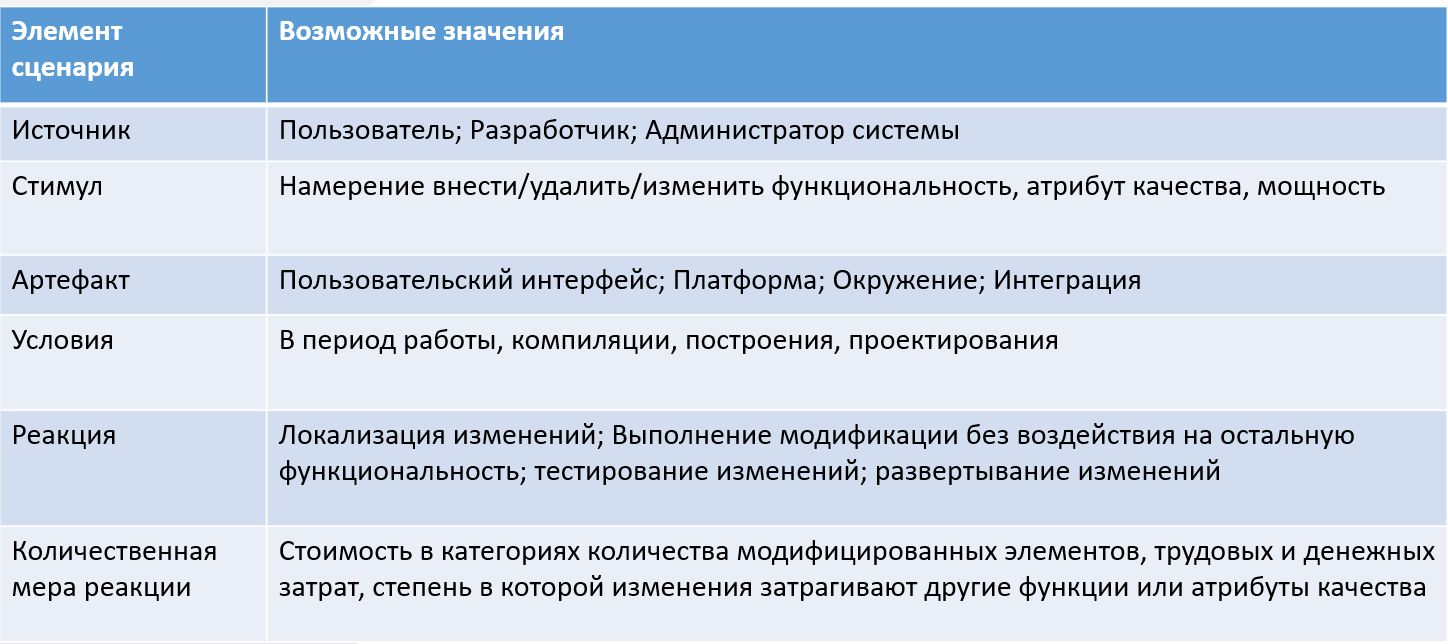
****

**Пример сценария модифицируемости:**

****

Качество модифицируемости выражает стоимость внесения изменений. Оно решает два вопроса:

1. Под воздействием чего артефакт может быть изменен? Изменения можно внести в любой аспект системы — как правило, это ее функции, платформа (аппаратная часть, операционная система, промежуточное программное обеспечение и т. д.)
2. Когда производится модификация и кто ее проводит (условия)? В прошлом изменения чаще всего вносились в исходный код. При этом разработчик проводил модификацию, после чего она тестировалась и размещалась в новой версии. Сегодня проблема времени модификации все больше переплетается с проблемой ее исполнителя. Изменения можно вносить в реализацию (путем редактирования исходного кода), в периоды компиляции (при помощи переключателей периода компиляции), построения (путем выбора тех или иных библиотек), настройки конфигурации (разными методами, в частности — путем установки параметров

****

**Производительность —** это временная характеристика. Когда фиксируются те или иные события (прерывания, сообщения, пользовательские запросы, временные промежутки), система должна на них реагировать. Характеристик поступления событий и реакций на них может быть много, по сути, вопрос заключается в том, за какое время система справляется с реакцией на то или иное событие.

Стимулом для всех сценариев производительности являются поступающие события. Образец поступления может быть периодическим, непереодическим и случайным. Пример стимула: непереодическое инициирование 100 000 транзакций в минуту.

Артефактом для всех сценариев производительности являются предоставляемые системой услуги.

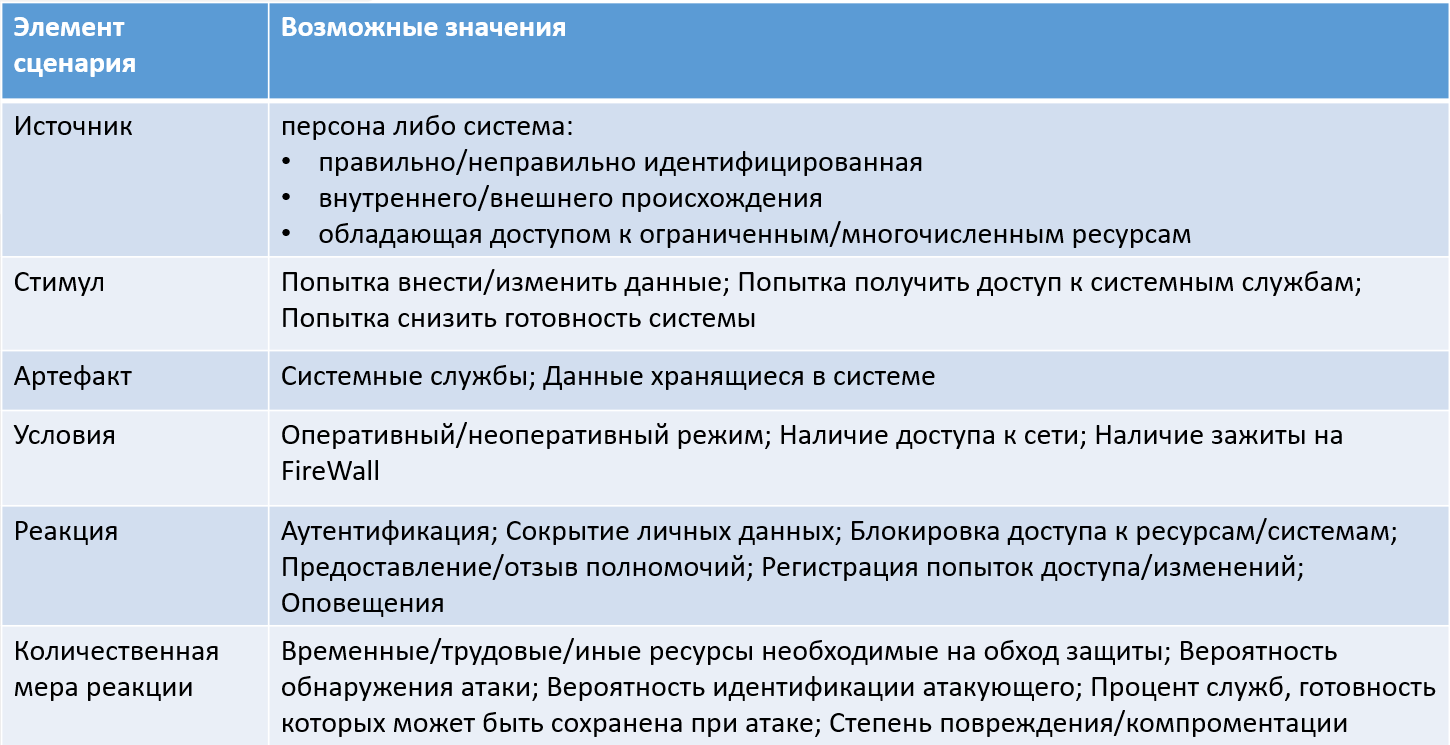


**Безопасность** отражает способность системы противостоять попыткам несанкционированного доступа при одновременном обслуживании легальных пользователей.

Безопасность можно определить как атрибут качества системы, предусматривающий:

* **Строгое выполнение обязательств** — это свойство, согласно которому права на отмену транзакции (операции доступа или модификации данных или услуг) нет ни у одной участвующей стороны
* **Конфиденциальность** — это свойство, предусматривающее защиту данных и услуг от несанкционированного доступа.
* **Целостность** — это свойство данных и служб, в соответствии с которым они предоставляются в оговоренном виде
* **Гарантирование**, участники транзакции не должны выдавать себя за сторонних лиц.
* **Готовность** — это свойство системы, обеспечивающее ее открытость для законных пользователей
* **Аудит** предполагает отслеживание системой всех операций на том уровне, на котором их впоследствии будет возможно реконструировать.

Каждой из перечисленных категорий соответствует ряд общих сценариев



**№ 8**

**1 вопрос (20 баллов): Написать 15 пользовательский историй к веб-сайту**

**портала IT поддержки с указанием критериев приемки и разделением**

**минимум на две роли.**

1. как пользователь веб-сайта, я хочу получать имейл после отправки заявки, чтобы знать, что моя заявка была принята на рассмотрение

критерий приемки: после корректного заполнения и отправки заявки, на почту пользователя должно приходить письмо на указанный имейл в заявке о подтверждении принятия заявки на рассмотрения

1. как пользователь веб-сайта, я хочу при заполнении заявки загружать фото, чтобы в моей заявки была фотография, более подробно описывающая мою проблему/ошибку

критерий приемки: на форме заявки есть поле загрузки фотографии формата jpg, jpeg, png, gif и при загрузку фото отображолось нижи превью этой фотографии

1. как пользователь веб-сайта, я хочу видеть корректный пример заполнения заявки, чтобы понимать, какую именно информацию и в каком формате мне нужно предоставить

критерий приемки: на странице формы заявки есть над самой формой есть пошаговая инструкция, описывающая правильность заполнения заявки

1. как администратор сайта, я хочу видеть диаграмму по заявкам, чтобы отслеживать статистику заявок выполненных, на рассмотрении и в работе

критерий приемки: на главной странице админа есть дэшборд, на котором выведена информация по заявкам на всех стадиях

1. как администратор, я хочу иметь кнопку назначить, чтобы поступившие заявки назначать на сотрудников

критерий приемки: на детальной странице заявки есть поле с выпадающим списком всех сотрудников и кнопка назначить

1. как администратор, я хочу видеть список всех заявок с возможностью перехода на детальной страницу каждой, чтобы изучать заявки для последующего назначения

критерий приемки: на странице все заявки отображается список из всех заявок

1. как администратор, я хочу фильтровать заявки, чтобы иметь более быстрый способ нахождения интересующей меня заявки

критерий приемки: на странице все заявки есть фильтр, содержащий поля выбора статуса заявки, назначенный на выполнение заявки, даты получения заявки/закрытия заявки, сортировка по возрастанию/убыванию по дате или срочности заявки, поиск по названию заявки

1. как пользователь сайта, я хочу иметь возможность выставить уровень срочности своей заявки, чтобы уведомить, насколько критична моя проблема и как быстро ее нужно решить

критерий приемки: на странице формы заявки есть поле срочность с выпадающим списком вариантов срочность: высокая, средняя, низкая

1. как сотрудник ИТ-поддержки, я хочу видеть все заявки , назначенные на меня, чтобы знать какие заявки мне нужно решить

критерий приемки: на странице назначенные мне отображаются все заявки, которые были назначены на него администратором

1. как сотрудник ИТ-поддержки, я хочу сортировать по срочности список своих заявок, чтобы понимать, какие заявки в приоритете

критерий приемки: на странице назначенные мне есть фильтр, в котором можно выбрать отобразить только со срочностью высокая/средняя/низкая или отсортировать по срочности по возрастанию (от низкой к высокой) или по убыванию (от высокой к низкой)

…ну и дальше сами что нить еще нафантазируете от лица пользователя :)

**2 вопрос (20 баллов): Проектное управление. Ограничения и жизненный**

**цикл проекта.**

Проектное управление является одним из двух основных видов управления разработкой (наряду с гибким управлением Agile).

Методологии проектного управления:

* PMBOK
* PRINCE2

Проект — это временное предприятие, направленное на создание уникального продукта, услуги или результата.

Проект — это законченное мероприятие по чему-либо

Не существует идеальной системы управления проектами, подходящей для каждого из видов проектов. Также не существует системы, которая бы подходила каждому руководителю и была удобна для всех членов команды.

Управление проектом - приложение знаний, навыков, инструментов и методов к работам проекта для удовлетворения требований, предъявляемых к проекту.

Управление проектом осуществляется посредством должного применения и интеграции логически сгруппированных 47 процессов управления проектом, объединенных в 5 групп процессов.

• Инициация = идея: процессы, выполняемые для определения нового проекта или новой фазы существующего проекта путем получения авторизации на начало проекта или фазы;

• Планирование: процессы, требуемые для установления содержания работ, уточнения целей и определения направления действий, требуемых для достижения целей проекта;

• Исполнение: процессы, применяемые для выполнения работ, указанных в плане управления проектом, с целью соответствия спецификациям проекта;

• Мониторинг и контроль: процессы, требуемые для отслеживания, анализа, а также регулирования исполнения проекта; выявления областей, требующих внесения изменений в план; и инициирования соответствующих изменений,

• Закрытие: процессы, выполняемые для завершения всех операций в рамках всех групп процессов в целях формального закрытия проекта или фазы

**Ограничения проекта:**

• содержание,

• качество,

• расписание,

• бюджет,

• ресурсы,

• риски.

**Жизненный цикл проекта** — набор фаз, через которые проходит проект с

момента его инициации до момента закрытия.

Все проекты могут иметь следующую структуру жизненного цикла:

• начало проекта;

• организация и подготовка;

• выполнение работ проекта;

• завершение проекта.



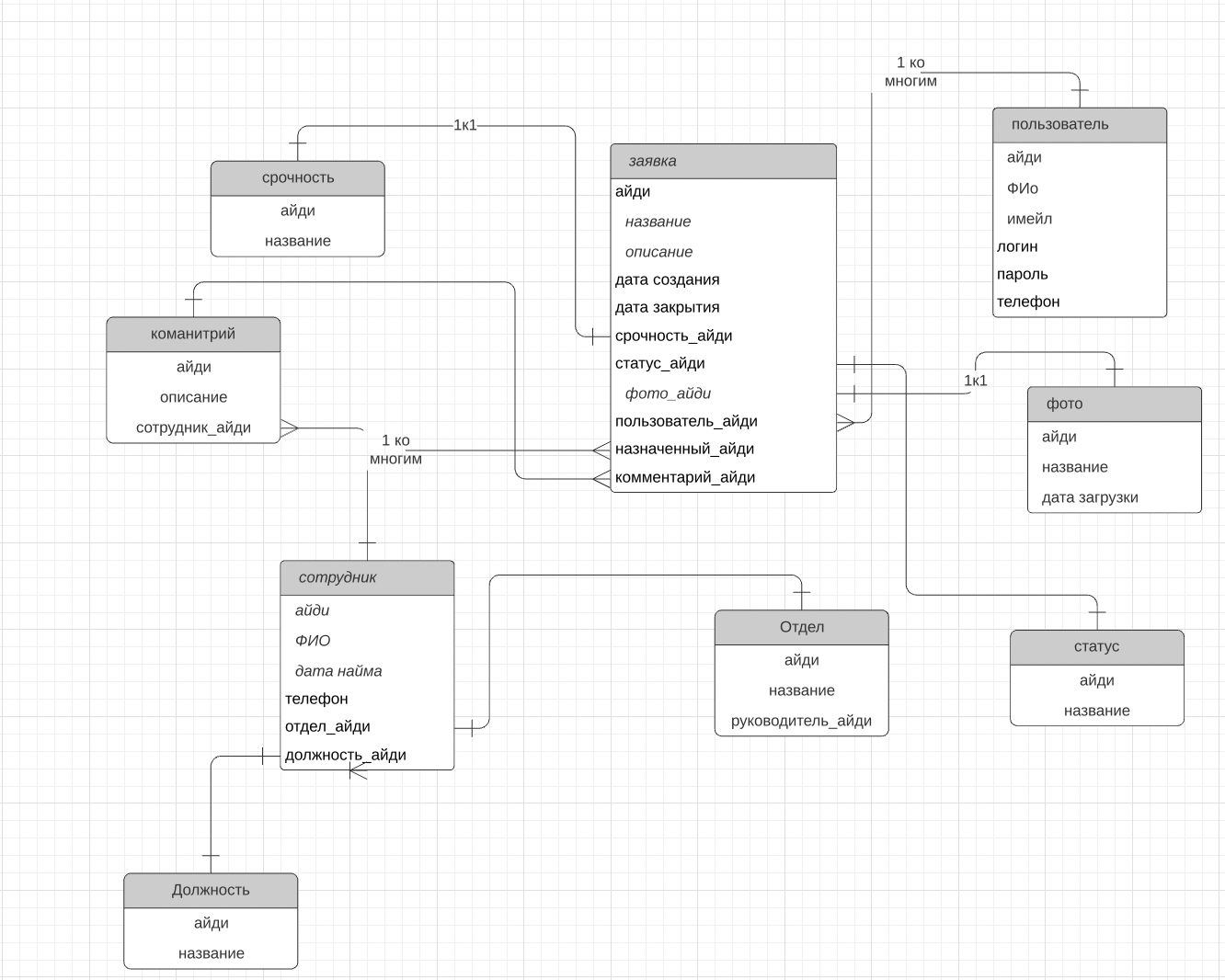
**3 вопрос (20 баллов): Атрибуты качества и сценарии - подробное описание на примере одного из типа сценариев (Модифицируемость, Ремонтопригодность, Производительность, Безопасность...).**

**№ 9**

**Составить ER-диаграмму по порталу**

**ИТ-поддержки. Не менее 6 сущностей, общий состав атрибутов не менее**

**40, связи много ко многим допускаются только с пояснением их**

**необходимости, минимум один пример связи один к одному.**

**2 вопрос (20 баллов): Проектное управление. Методологии. PMBOOK и PRINCE2.**

Проектное управление является одним из двух основных видов управления разработкой (наряду с гибким управлением Agile). **PMBOOK и PRINCE2 выступают в качестве методологий проектного управления.**

Не существует идеальной системы управления проектами, подходящей для каждого из видов проектов. Также не существует системы, которая бы подходила каждому руководителю и была удобна для всех членов команды

**Project Management Body Of Knowledge (PMBOK).**

Управление проектом - приложение знаний, навыков, инструментов и методов к работам проекта для удовлетворения требований, предъявляемых к проекту.

Управление проектом осуществляется посредством должного применения и интеграции логически сгруппированных 47 процессов управления проектом, объединенных в 5 групп процессов.

• Инициация = идея: процессы, выполняемые для определения нового проекта или новой фазы существующего проекта путем получения авторизации на начало проекта или фазы;

• Планирование: процессы, требуемые для установления содержания работ, уточнения целей и определения направления действий, требуемых для достижения целей проекта;

• Исполнение: процессы, применяемые для выполнения работ, указанных в плане управления проектом, с целью соответствия спецификациям проекта;

• Мониторинг и контроль: процессы, требуемые для отслеживания, анализа, а такжЕ регулирования исполнения проекта; выявления областей, требующих внесения изменений в план; и инициирования соответствующих изменений,

• Закрытие: процессы, выполняемые для завершения всех операций в рамках всех групп процессов в целях формального закрытия проекта или фазы.

Область знаний является всеобъемлющей системой понятий, терминов и действий, составляющих профессиональную область, область управления проектами или область деятельности.

● управление интеграцией проекта (процессы и операции, необходимые для определения, уточнения, комбинирования, объединения и координации различных процессов и операций по управлению проектом в рамках групп процессов управления проектом)

● управление содержанием проекта (процессы, требуемые для обеспечения того, чтобы проект содержал все и только те работы, которые требуются для успешного выполнения проекта)

● управление сроками проекта (процессы, необходимые для того, чтобы обеспечить своевременное выполнение проекта)

● управление стоимостью проекта (процессы, необходимые для планирования, оценки, разработки бюджета, привлечения финансирования, финансирования, управления и контроля

стоимости)

● управление качеством проекта (процессы и действия исполняющей организации, которые определяют политики, цели и сферы ответственности в области качества таким образом, чтобы проект удовлетворял тем потребностям, ради которых он был предпринят)

● управление человеческими ресурсами проекта (процессы организации, управления и руководства командой проекта)

● управление коммуникациями проекта

● управление рисками проекта (процессы, связанные с осуществлением планирования управления рисками, идентификацией, анализом, планированием реагирования, а также с

контролем рисков в проекте)

● управление закупками проекта (процессы покупки или приобретения необходимых для осуществления проекта продуктов, услуг или результатов вне команды проекта)

● управление заинтересованными сторонами проекта (процессы, необходимые для выявления людей, групп и организаций, которые могут оказывать или на которых может оказывать воздействие проект)

**Ограничения проекта**: содержание, качество, расписание, бюджет, ресурсы и риски.

Жизненный цикл проекта — набор фаз, через которые проходит проект с момента его инициации до момента закрытия.

**Структура жизненного цикла проекта**: начало проекта, организация и подготовка, выполнение работ проекта, завершение проекта.

**Основные документы:**

• Устав проекта;

* Общая информация (название, спонсор, РП);
* Причины инициации проекта;
* Бизнес-ценность проекта;
* Содержание проекта;
* Ожидаемый результат;
* Ограничения проекта.

• Описание содержание проекта;

* цели, критерии достижения, задачи и результаты, границы проекты, допущения и риски

• План управления проектом;

* план работ по времени и ресурсам
* управление командой проекта
* управление коммуникациями
* риски проекта и план работы с ними
* управление контрактами и закупками
* управление изменениями

• Запрос на изменение;

• Протокол совещания;

• Реестр рисков;

• Отчет о статусе проекта;

• Отчет по завершении проекта.

PMBOK. Планирование. Типы зависимостей операций

Финиш-старт (finish-start, FS). Логическая связь, при которой старт последующей операции зависит от финиша предшествующей операции.

Финиш-финиш (finish-finish, FF). Логическая связь, при которой финиш последующей операции зависит от финиша предшествующей операции.

Старт-старт (start-start, SS). Логическая связь, при которой старт последующей операции зависит от старта предшествующей операции.

Старт-финиш (start-finish, SF). Логическая связь, при которой финиш последующей операции зависит от старта предшествующей операции.

**PRojects IN Controlled Environments version 2 (PRINCE2).**

Методология PRINCE2 в отличие от, например, свода знаний PMBOK не содержит:

• Специализированных аспектов управления проектом, например, отраслевых;

• Конкретных практик и инструментов управления проектами, таких как

диаграмма Ганта, WBS и т.п.

PRINCE2 концентрируется на управленческих сторонах проекта, выраженных в 7 принципах, 7 процессах и 7 темах проекта.

• 7 принципов определяют общие правила управления проектами по PRINCE2,

определяют базу методологии;

• 7 процессов определяют шаги продвижения по проектному циклу;

• 7 тем – аспекты, по которым проводится контроль для достижения успеха

проекта.

**Ограничения проекта:**

• содержание,

• качество,

• деньги,

• время,

• выгоды,

• риски.

**Процессы управления:**

Начало проекта

Инициация проекта

Руководство проектом

Контроль стадии

Управление созданием продукта

Управление границами стадии

Завершение проекта

**Компоненты методологии:**

**Обоснование проекта.** Стоит ли его запускать, получится ли сделать и какими методами. **Организация.** Управляющий комитет контролирует работу руководителя проекта

и отчитывается менеджменту компании.

**Планы.** Одно из свойств Prince2 — планирование на основе **Характеристик продукта**: почему его создают, какие материалы или навыки нужны. Это позволяет сразу сосредоточиться на нужном результате, а не вносить потом поспешные изменения.

**Контроль.**

**Управление рисками.**

**Качество.**

**Изменения.** Они, как и риски, неизбежны: подготовить полностью предсказуемый проект едва ли возможно.

**Принципы:**

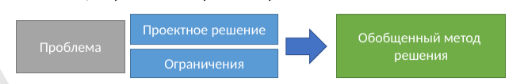
* На каждом этапе проверяйте, остается ли проект целесообразным.
* Учитесь на ошибках.
* Четко указывайте роли и обязанности.
* Управляйте поэтапно.
* Управляйте по необходимости.
* Вмешивайтесь тогда, когда нарушаются одно из шести ограничений.
* Сосредоточьтесь на продукте. Каждый этап нужно завершить промежуточным
* результатом
* Проявляйте гибкость. Если составление документов о проделанной работе занимает
* неоправданно много времени, то можно просто отчитаться устно перед комитетом или
* прислать им электронное письмо.

**3 вопрос (20 баллов): Архитектурные стили. Уровни применения,**

**типичные представители (на примере выбранного - монолит, SOA,**

**Микросервисы...).**

Архитектурный стиль – набор типов элементов и связей между ними, а также ограничений использования, направленный на решение проблемы



Архитектурный шаблон – основа для построения структурных схем программных систем. Описывает набор предопределенных подсистем, их ответственность и правила организации связи между ними

Использование архитектурных стилей:

* уровень решения (solution) - монолит, сервисно-ориентированный, микросервисы, бессерверный
* уровень компонента (component) - “слоистый” (layred), модульный, REST

**Монолит**

Монолитная модель - единая неделимая единципа.

**Концепция:**

различные компоненты приложения объединяются в одну программу на одной платформе. Обычно монолитное приложение состоит из базы данных, клиентского пользовательского интерфейса и серверного приложения. Все части программного обеспечения унифицированы, и все его функции управляются в одном месте.

**Плюсы монолитной архитектуры**

• Упрощенная разработка и развертывание (одна кодовая база, общие компоненты и библиотеки, единая команда)

• Меньше сквозных проблем (приложения зависят от множества межкомпонентных задач, таких как контрольные журналы, ограничение скорости и т. д.)

• Лучшая производительность (при правильной реализации).

**Минусы монолитной архитектуры**

• Кодовая база со временем становится громоздкой

• Сложно внедрять новые технологии (нельзя внедрить что-то в одном месте, только везде и сразу)

• Ограниченная гибкость (каждое небольшое обновление требует полного повторного развертывания)

Монолитная модель не устарела, и в некоторых случаях она по-прежнему прекрасно работает. Некоторые гигантские компании остаются монолитными, несмотря на сегодняшнюю популярность микросервисов и SOA

**Сервис-ориентированная архитектура (SOA)** — это стиль архитектуры программного обеспечения, который предполагает модульное приложение, состоящее из дискретных и слабосвязанных программных агентов, которые выполняют конкретные функции.

Бизнес-функции «модулезированы» и предоставляются приложениям-потребителям в виде отдельных специализированных служб

• Самодокументирующиеся интерфейсы

• Не завязано на конкретную технологию

• Легкость создания бизнес-приложений за счет компоновки сервисов

Манифест SOA

Ориентация на сервисы - это парадигма, которая формирует то, что вы делаете. Сервис-ориентированная архитектура является типом архитектуры, который является результатом ориентации на сервисы. Мы применяем ориентацию на сервисы, чтобы помочь организациям единообразно поставлять самоподдерживающуюся организационную ценность, с увеличенной гибкостью и экономической эффективностью, в согласованности с изменяющимися организационными потребностями. В нашей работе мы даем приоритет:

Организационные ценности выше технической стратегии

Стратегические цели выше выгод отдельного проекта

Внутренне присущая возможность взаимодействия выше заказной интеграции

Совместно используемые сервисы выше реализаций с отдельным назначением

Гибкость выше оптимизации

Эволюционное совершенствование выше стремления к изначальному совершенству

Плюсы SOA

* Повторное использование сервисов. Компоненты можно повторно использовать в нескольких приложениях, без влияния на другие сервисы.
* Легкость в сопровождении. Поскольку каждая служба программного обеспечения является независимой единицей, ее легко обновлять и поддерживать, не затрагивая другие службы.
* Более высокая надежность. Службы легче отлаживать и тестировать, что делает продукты на основе SOA более надежными.
* Параллельная разработка. Поскольку сервис-ориентированная архитектура разбита на прослойки, она поддерживает параллелизм в процессе разработки.

Минусы SOA

* Сложность в управлении. Основным недостатком сервис-ориентированной архитектуры является ее сложность. Каждый сервис должен обеспечивать своевременную доставку сообщений. Количество этих сообщений может превышать миллион за один раз, что затрудняет управление всеми службами.
* Высокие инвестиционные затраты. Разработка SOA требует значительных предварительных инвестиций в человеческие ресурсы, технологии и разработку.
* Дополнительная нагрузка. В SOA все входные данные проверяются до того, как один сервис взаимодействует с другим сервисом.

При использовании нескольких сервисов это увеличивает время отклика и снижает общую производительность.

SOA лучше всего подходит для сложных корпоративных систем, например банковских.

Микросервисы. «Облегченная SOA»

Ключевые атрибуты

• Небольшие (легко разобраться, быстрая поставка)

• Loose Coupling, High Cohesion

• Независимые (легко часто выкладывать новые версии)

• Изоляция проблем (Faults Isolation)

• Поддерживаются SOAP, REST, Event-driven асинхронные вызовы

Проблемы

• N монолитов -> N х M сервисов

• Сложности "нарезки" системы на сервисы

• API Gateway, Service Registry, Load Balancer, Fault Tolerance/Monitoring,

Configuration/Properties Management

• Технологический стек

Характеристики микросервисной архитектуры

* Компоненты -> Сервисы
* Организация вокруг Бизнес-ценностей
* Продукты а не Проекты
* Децентрализованное управление
* Децентрализованное управление данными
* Автоматизация инфраструктуры
* Проектирование с учетом отказов
* Эволюционная разработка

**Плюсы**

• Легко разрабатывать, тестировать и развертывать. Самое большое преимущество микросервисов перед другими архитектурами заключается в том, что небольшие отдельные сервисы могут создаваться,

тестироваться и развертываться независимо

• Повышенная гибкость. С помощью микросервисов несколько команд могут работать над своими сервисами независимо и быстро.

• Возможность горизонтального масштабирования. Вертикальное масштабирование может быть ограничено пропускной способностью каждого сервиса. Но горизонтальное масштабирование (создание большего количества сервисов в одном пуле) не ограничено и может работать с микросервисами динамически.

**Минусы**

• Сложность

• Безопасность

Микросервисы хороши, но не для всех типов приложений. Этот шаблон отлично подходит для развивающихся приложений и сложных систем. Вы можете думать о том, чтобы выбрать архитектуру микросервисов, когда у вас есть несколько опытных команд и когда приложение достаточно сложное, чтобы разбить его на сервисы.

**Беcсерверная (Serverless) архитектура**

Serverless архитектура — это подход применения облачных вычислений для создания и запуска приложений и сервисов без необходимости управления инфраструктурой. В таких приложениях выполнение кода управляется платформой, что позволяет разработчикам развертывать код, не беспокоясь об обслуживании и обеспечении сервера.

Serverless архитектура включает в себя две концепции:

FaaS (Function as a Service — функция как услуга) — модель облачных вычислений, которая позволяет разработчикам загружать части функционала в облако и позволяет этим частям выполняться независимо

BaaS (Backend as a Service — бэкэнд как услуга) — модель облачных вычислений, которая позволяет разработчикам передавать на аутсорсинг аспекты бэкэнда (управление базой данных, облачное хранилище, хостинг, аутентификация пользователей и т. д.), а также писать и поддерживать только часть

внешнего интерфейса.

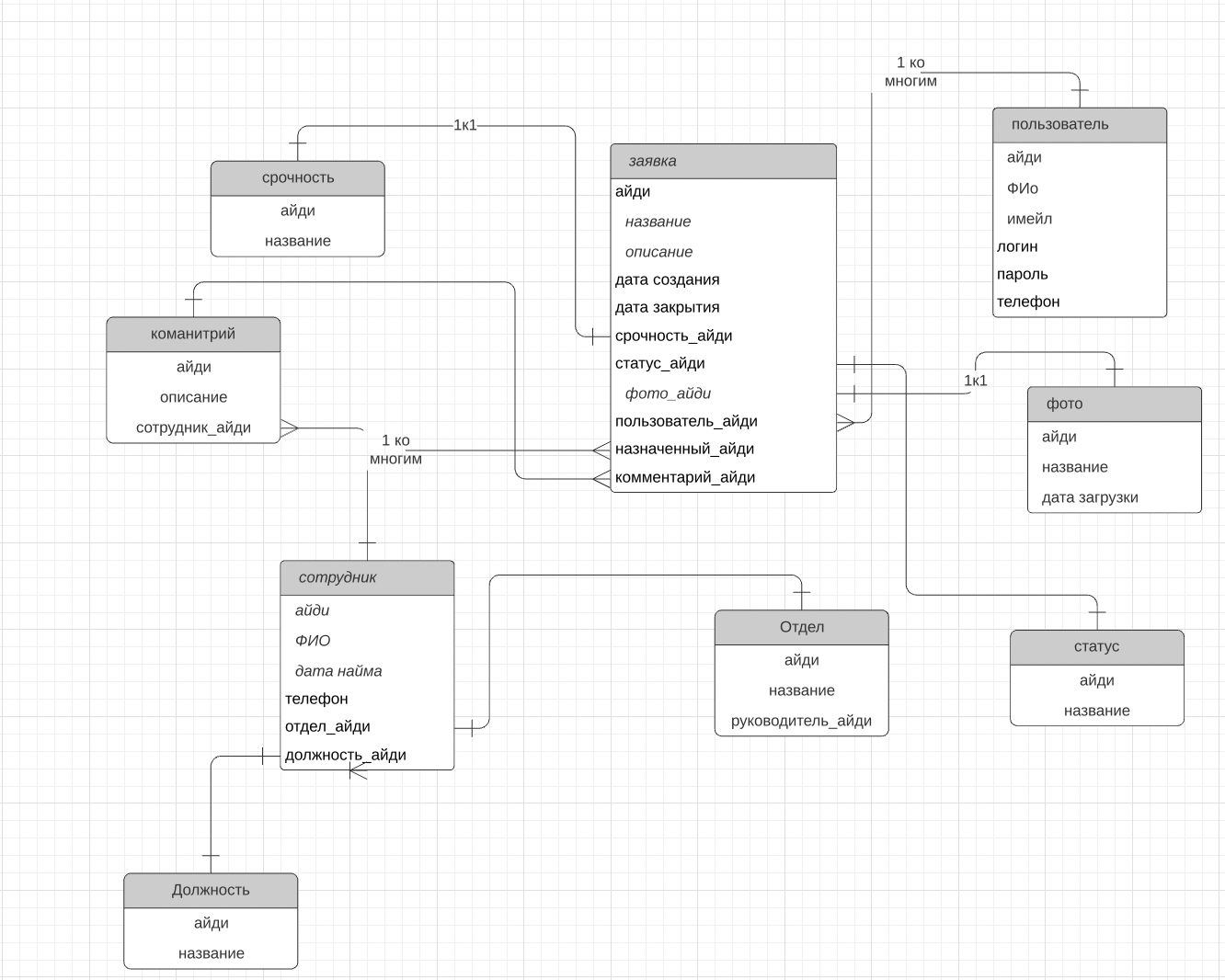
**№ 10**

**1 вопрос (20 баллов): Составить ER-диаграмму по порталу**

**ИТ-поддержки. Не менее 6 сущностей, общий состав атрибутов не менее**

**40, связи много ко многим допускаются только с пояснением их**

**необходимости, минимум один пример связи один к одному.**

****

**2 вопрос (20 баллов): Масштабирование гибкой разработки. Проблемы и**

**решения. SAFe как пример масштабируемого фреймворка.а**

Scaled Agile Framework® (SAFe®) — это набор организационных шаблонов и шаблонов рабочих процессов для [реализации agile-методик в масштабе всей компании](https://www.atlassian.com/ru/agile/agile-at-scale). Эта платформа представляет собой совокупность знаний, куда входят структурированное руководство по ролям и обязанностям, способы планирования работы и управления ею, а также соответствующие ценности.

Базовый SAFe — содержит минимальный набор ролей, событий и артефактов, необходимых для постоянной доставки бизнес-решений через Agile-поезд (ART, Agile Release Train), как команду Agile-команд.

Платформа SAFe применяется во множестве agile-команд, обеспечивая согласованность, помогая выполнять совместную работу и поставку. В ее основу легли три основных блока знаний: гибкая (agile) разработка программного обеспечения, «бережливая» (lean) разработка продукции и системное мышление.

### Принципы SAF Принцип № 1. Смотрите с точки зрения экономики Принцип № 2. Применяйте системное мышление Принцип № 3. Допускайте вариативность; сохраняйте варианты Принцип № 4. Выполняйте инкрементальные сборки с быстрыми циклами обучения, интегрированными в процесс Принцип № 5. Контрольные точки должны основываться на объективной оценке работающих систем Принцип № 6. Визуализируйте и ограничивайте незавершенные работы (WIP), уменьшайте объем работ и управляйте длиной очередей Принцип № 7. Применяйте каденции, выполняйте синхронизацию с помощью кросс-доменного планирования Принцип № 8. Раскройте внутреннюю мотивацию работников умственного труда Принцип № 9. Децентрализуйте принятие решений

**3 вопрос (20 баллов): Документирование программных систем.**

**Неформальные нотации. Плюсы и минусы, области применения.**

Содержание документации зависит от предполагаемых вариантов ее применения.

Документация ни при каких обстоятельствах не может быть универсальной. С одной стороны, она должна быть абстрактной и, следовательно, доступной для понимания новыми сотрудниками, но с другой — весьма детальной — настолько, чтобы ее можно быть использовать как план проведения анализа.

Заинтересованные в составлении документации лица преследуют разные цели.

Стремиться следует к тому, чтобы любое заинтересованное лицо смогло быстро найти необходимую информацию, при этом не тратя время на незначительные (с его точки зрения) сведения.

Для того чтобы понять, как структурировать документацию и повысить удобство ее использования, следует составить представление о заинтересованных лицах и разобраться в том, для чего она им может понадобиться.

Наиболее важным понятием документирования программной архитектуры является «представление» (view) - способ фиксации структуры, с ним связан основной принцип документирования программной архитектуры:

Документирование архитектуры подразумевает документирование всех значимых представлений с последующей фиксацией сведений, относящихся одновременно к нескольким представлениям

Этот принцип полезен тем, что он помогает разбить проблему документирования архитектуры на ряд менее обширных элементов:

♦ выбор значимых представлений;

♦ документирование представления;

♦ документирование сведений, относящихся к нескольким представлениям.

Общие подходы к документированию программных систем

Типы нотаций

* **формальные нотации**

•Семантика полностью стандартизована, смысл диаграмм определяется однозначно

• Возможен формальный анализ как синтаксиса, так и семантики модели

• Примеры: AADL, ER

Особенности:

• Избыточно сложны

• Требуются глубокие знания языка описания

• ВСЕГДА требуется дополнительное ПО

• Больше подходят для технического описания

аппаратных комплексов

* **Частично формальные нотации**

Нотация стандартизована, есть наборы графических примитивов и правила их использования

• Полная семантика и возможные трактовки элементов диаграмм не зафиксированы

• Анализ семантики упрощен по сравнению с неформальными нотациями

• Примеры: UML, BPMN, ERD

Особенности:

Требуется изучение нотации всеми

участниками процесса

• Как правило, требуется дополнительное ПО

• Фокусировка на определенных областях, атрибутах

* **Неформальные нотации**

• Диаграммы в свободной форме

• Семантика моделей на естественном языке (цвета, форма, надписи)

• Информация диаграмм не может быть формально проанализирована

• Легко понятные, но при этом высок риск неправильного понимания

• Пример: классические boxes-and-lines например в PowerPoint

Особенности:

• Для понимания требуется «ключ»

• Содержимое раскрывается с помощью фигур, цветов и стрелок

• Легко создавать

• Самые популярные

• Хороши для презентаций

Выбор нотации всегда должен учитывать:

Какие заинтересованные лица (stakeholders) будут вовлечены в работу с документацией

Стандарты организации/проектной группы

Более формальные нотации требуют больших затрат, но дают новые возможности (анализ и т.д.)

Менее формальные нотации легче создавать и применять, но они дают меньше гарантий

Нотации: типичные представители

• UML

• BPMN

• C4

• FlowChart/swimlane

**№ 11**

**1 вопрос (20 баллов): Написать не менее 10 различных дефектов на**

**страницу оформления обращения в ИТ-поддержку и на страницу**

**просмотра списка своих обращений. Описание должно быть достаточным**

**для его воспроизведения и ожидаемым результатом. Должны быть ссылки**

**на предполагаемые сценарии тестирования.**

| Дата | 22/05/2021 | |
| --- | --- | --- |
| Название | Неправильное отображение расписания в виде таблицы | |
| Компонент веб-приложения | Функция отображения макета расписания | |
| Номер версии | v 1.0 | |
| Критичность бага | Major | |
| Приоритет бага | Medium | |
| Статус | Open | |
| Автор баг-репорта | Молчанова В А | |
| Исполнитель | Габов А С | |
| Окружение | Google Chrome Version 89.0.4389.128 (Official Build) (64-bit) | |
| Номер тестового сценария | 3 | |

| Шаги воспроизведения | Фактический результат | Ожидаемый результат |
| --- | --- | --- |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу формы обращения | при переходе выдает 404 ошибку | На странице отображается форма с полями для обращения |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу просмотра списка своих обращений | при переходе выдает 404 ошибку | на странице отображается список всех обращений пользователя |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу просмотра списка своих обращений  3. кликнуть на название любого из обращений в списке | при переходе выдает 404 ошибку | на странице должна отображаться детальная информация по заявке |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу просмотра списка своих обращений  3. в фильтре поставить галочку отображать только “в работе” | на странице отображается весь отсортированный список обращений | на странице должны отображаться только те обращение, у которых стоит статус “в работе” |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу формы обращения  3. заполнить поле “название” = тест  4. заполнить поле “описание” = тест  5. выбрать из выпадающего списка “срочность” = высокая  6. в поле “фотография” загрузить любое фото формата png  7. нажать на кнопку “отправить” | выдает 500 ошибку | должен переходить на страницу всех обращений пользователя и отображать всплывающее сообщение “обращение было успешно отправлено!” |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу формы обращения  3. заполнить поле “название” = тест  4. заполнить поле “описание” = тест  5. не заполнять поле “срочность”  6. в поле “фотография” загрузить любое фото формата png  7. нажать на кнопку “отправить” | переходит на страницу всех обращений пользователя и отображает всплывающее сообщение “обращение было успешно отправлено!” | возвращение на страницу формы обращения, выделение поля срочность красным фоном и вывод сообщения над формой “поле срочность обязательно для ввода” с сохранением всех введенных данных пользователем |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу формы обращения  3. заполнить поле “название” = тест  4. заполнить поле “описание” = тест  5. выбрать из выпадающего списка “срочность” = высокая  6. в поле “фотография” загрузить любое фото формата png  7. нажать на кнопку “отправить” | возвращение на страницу формы обращения, выделение поля срочность красным фоном и вывод сообщения над формой “поле срочность обязательно для ввода” с пустыми полями ввода | возвращение на страницу формы обращения, выделение поля срочность красным фоном и вывод сообщения над формой “поле срочность обязательно для ввода” с сохранением всех введенных данных пользователем |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу формы обращения  3. заполнить поле “название” = тест  4. заполнить поле “описание” = тест  5. выбрать из выпадающего списка “срочность” = высокая  6. в поле “фотография” загрузить любое фото формата png | не загружает прикрепленную фотографию | загрузить фотографию и отобразить превью вместо поля “загрузка фотографии” |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу формы обращения  3. заполнить поле “название” = тест  4. заполнить поле “описание” = тест  5. выбрать из выпадающего списка “срочность” = высокая  6. в поле “фотография” загрузить любое фото формата png  7. нажать на кнопку “отправить” | ничего не происходит | должен переходить на страницу всех обращений пользователя и отображать всплывающее сообщение “обращение было успешно отправлено!” |
| 1. на странице авторизации Ввести в поле «Логин» логин (test), и в поле «Пароль» - пароль (test) для авторизации  2. перейти на страницу просмотра списка своих обращений  3. кликнуть на название любого из обращений в списке | переходит на детальную страницу обращения, но информации по обращению нет | на странице должна отображаться детальная информация по заявке |

| Приложение | \*скриншот\* | |
| --- | --- | --- |
| Комментарий | - | |

**2 вопрос (20 баллов): Гибкие технологии разработки. Kanban. Сравнение**

**со Scrum.**

**Agile** — это структурированный итеративный подход к управлению проектами и разработке продуктов.

**Скрам** — это фреймворк, в котором можно использовать разнообразные процессы и методы.

Скрам является:

• компактным,

• простым для понимания,

• трудным для совершенного овладения.

**Канбан** — это метод для определения, управления и совершенствования сервисов, поставляющих результаты умственного труда, такие как экспертные и креативные услуги, а также разработку физических или программных продуктов.

Канбан-метод базируется на визуализации нематериальной деятельности, умственного труда, в целях правильной работы сервиса, достижения баланса между работой запрошенной потребителем и возможностями сервиса. С этой целью используются канбан системы — поточные системы, поставляющие результат и имеющие ограничение незавершенной работы (Work In Progress, WiP) с помощью использования визуальных сигналов.

Основные практики Канбана определяют базовые действия лиц, занимающихся управлением канбан-системами.

Всего существует шесть таких практик:

1 Визуализируй

2 Ограничивай количество незавершенной работы

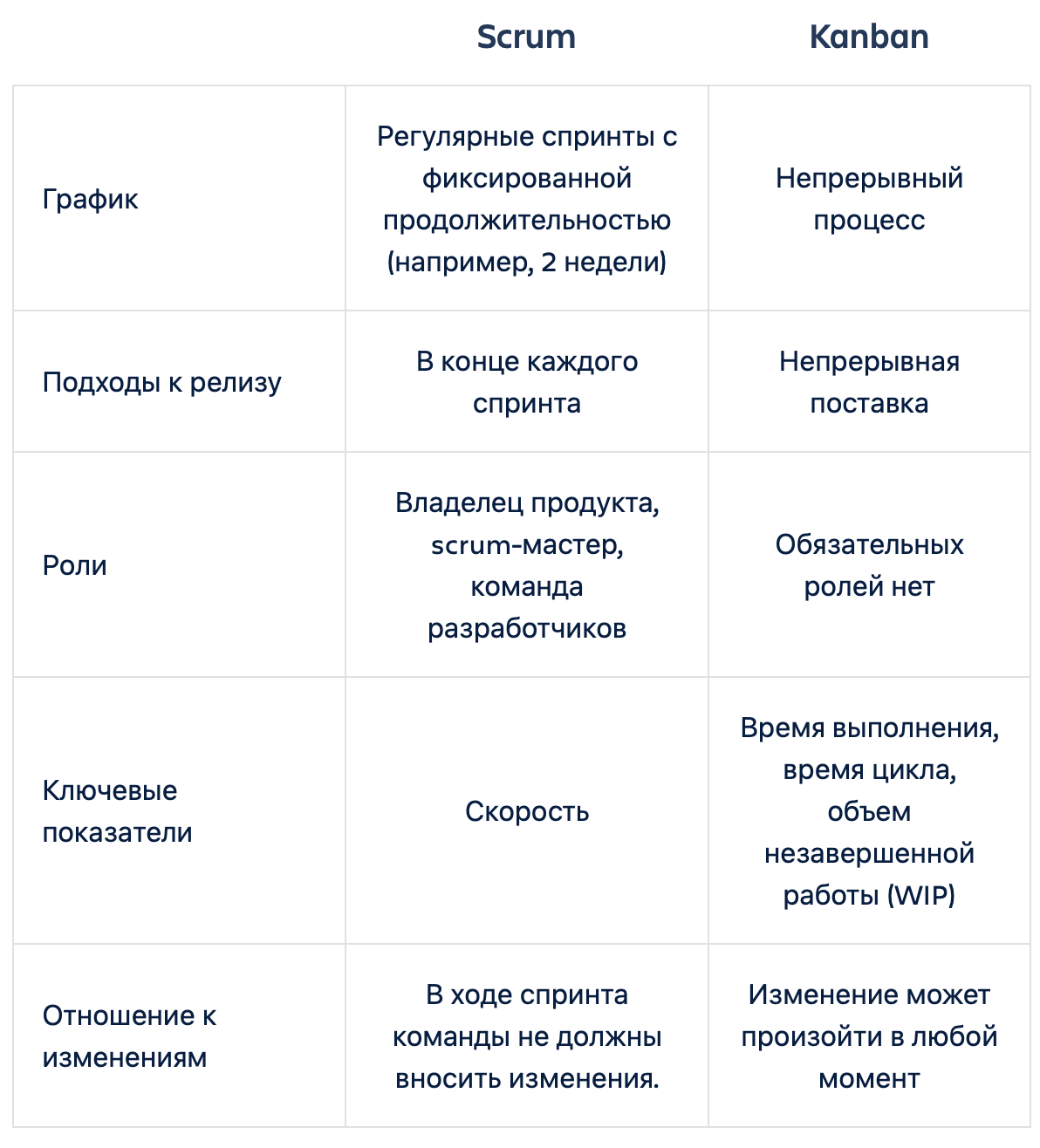
3 Управляй потоком

4 Делай правила работы явными

5 Внедряй циклы обратной связи

6 Улучшайтесь совместно, эволюционируйте на основе экспериментов

Методики Kanban предполагают непрерывность и большую подвижность, а Scrum основывается на коротких структурированных спринтах работы. Обе методологии помогают улучшить качество продуктов (и сервисов) и избавляют от множества проблем.



**3 вопрос (20 баллов): Рефакторинг. Техники рефакторинга (на примере одной из техник).**

**Рефакторинг** — процесс изменения внутренней структуры программного продукта, не затрагивающий её внешнего поведения и имеющий целью облегчение понимания программного кода и, пусть и не всегда, оптимизацию производительности.

Обычно рефакторят код под контролем прохождения автоматизированных тестов, без этого достаточно сложно убедиться, что внесённые изменения не являются деструктивными и поведение программы осталось прежним.

Наиболее частые причины для рефакторинга:

● дублирование кода

● длинные методы

● объёмные классы

● длинные списки параметров

● избыточные временные переменные

● классы данных

● несгруппированные данные

● несоблюдение стандартов кодирования

Рефакторинг сам по себе нередко повышает производительность, так как выявляются и удаляются лишние конструкции, от которых результат работы не зависит, но на время выполнения программы такие конструкции могут существенно влиять; иногда меняется и последовательность выполнения программного кода, что тоже может позитивно повлиять на производительность.

Одним из важных выборов при рефакторинге является время его проведения. Нужно ли выделять для

проведения рефакторинга (например) две недели после пары месяцев работы?

Лучшей стратегией является постоянный рефакторинг, когда изменения небольшие но вносятся сразу по

мере необходимости. Конечно, для такого стиля работы необходима автоматизация тестирования.

Существует огромное число техник рефакторинга. Вот несколько из них:

1. Извлечение переменной
2. Встраивание класса
3. Замена значения ссылкой
4. Объединение условных операторов

**Техника “Извлечения метода”**

У нас есть код, в котором какой-то фрагмент можно сгруппировать.

Суть данной техники - убрать огромные нечитабельные куски кода

Пример:

Часто это касается методов вывода.

У нас есть несколько print, которые мы вызываем где-то в конце коде. Мы просто берем и запихиваем эти print в отдельный метод вывода. Вуаля! Вы обладаете техникой “извлекания метода”

**№ 12**

**1 вопрос (20 баллов): Рассмотрим ситуацию когда вы на стороне**

**подрядчика должны подготовить комплект документов для представления**

**проекта спонсорам проекта на стороне заказчика. Опишите состав**

**документов, примерное содержимое, используемые нотации.**

## Перечень проектных документов на стадии инициации

С целью защиты проекта на проектном комитете рабочая группа формирует пакет документов, который может включать следующие:

1. Концепция проекта (или аванпроект);

Концепция проекта – это основной документ, описывающий структуру проекта, цели, задачи, ограничения и допущения. Также она отражает будущий результат, сроки и способы его достижения, сведения о рисках и технико-экономических показателях. [Написание концепции проекта](https://pmskills.ru/fundamentals-of-pm/project-initiation/napisanie-kontseptsii-proekta) – это тоже мини-проект со своими этапами и сроками.

1. Технико-экономическое обоснование (или бизнес-план); (будем использовать [ГОСТ 24.202-80](https://files.stroyinf.ru/Data2/1/4294752/4294752966.pdf) так как содержит все необходимые требования к ТЭО создания АСУ)

Технико-экономическое обоснование (ТЭО) — это документ, обосновывающий целесообразность создания продукта или услуги. ТЭО содержит анализ затрат и результатов проекта. Оно позволяет инвесторам определить, стоит ли вкладывать деньги в предлагаемый проект.

1. Жизненный цикл проекта; (будем строить по V-диаграмме)

Описание жизненного цикла проекта – это документ, предназначенный для описания основных этапов проекта со сроками и критериями прохождения контрольных рубежей.

1. Дорожная карта; (для визуализации roadmap будем использовать инструмент miro, так как там уже есть готовые шаблоны)

Визуализация [жизненного цикла проекта](https://pmskills.ru/fundamentals-of-pm/fazy-zhiznennogo-tsikla-proekta) с основными этапами его реализации называется Дорожной картой.

1. Устав проекта; (нотацию pmbok будем использовать)

Устав проекта – это документ, кратко описывающий:

* цели и результаты проекта, ради которых он создан;
* ограничения, которым следует удовлетворить;
* организационную структуру управления проектом, её персональный состав,
* взаимодействие и ответственность всех субъектов управления данным проектом.

1. Сводный план проекта (или стратегический план); (бу)

Сводный план проекта – календарно-сетевой график проекта, детализированный до уровня работ и содержащий сведения об ответственных исполнителях работ.

1. Бюджет проекта (финансовый план);

Бюджет проекта – документ, описывающий подробное распределение входящих и исходящих денежных потоков проекта по периодам и статьям затрат.

1. План управления рисками; (Методология управления рисками проектов была реализована и закреплена в виде универсальной «Методики построения карты рисков для проектного менеджмента»)

План управления рисками проекта описывает основные факторы возникновения рисков, их качественные и количественные характеристики, планы реагирования на риски, а также необходимые для этого ресурсы.

1. План управления коммуникациями проекта; (процессы управления коммуникациями в рамках стандарта PMBOK)

План управления коммуникациями проекта – документ, описывающий способы передачи информация о ходе, результатах, рисках и проблемах проекта.

1. Презентационные материалы.

Подготовить презентацию по бизнес части и технической части проекта

Вообще не существует четко определенного перечня документов, требуемых для инициации проекта. Каждая организация в рамках проектного управления формирует свой перечень необходимых документов и их шаблоны.

**2 вопрос (20 баллов): Назовите основные практики DevOps и приведите примеры**

**внутри этих практик.**

«DevOps» – сочетание «development» и «operations» (разработка и операции). Игра слов отражает основную идею, заложенную в DevOps – сотрудничество между различными дисциплинами разработки.

Технология разработки ПО с помощью специальных инструментов, которые помогают автоматизировать весь процесс разработки. Так намного быстрее происходит выпуск и обновление программного продукта или приложения.

**Непрерывная интеграция (CI, Continuous Integration)** – это практика разработки программного обеспечения, которая заключается в выполнении частых автоматизированных сборок для скорейшего выявления ошибок. Это позволяет постоянно быть уверенным в том, что код находится в рабочем состоянии. Переход к непрерывной интеграции позволяет снизить трудоёмкость интеграции изменений кода с существующей кодобазой и сделать её более предсказуемой за счет наиболее раннего обнаружения и устранения ошибок и противоречий, сделать процесс разработки прозрачным для всех участников команды.

**Непрерывная поставка (CDL, Continuous delivery)** – практика разработки программного обеспечения, гарантирующая то, что программное обеспечение постоянно готово к развертыванию в промышленную эксплуатацию. Практика включает в себя автоматизацию развертывания на тестовые среды, автоматизацию тестирования успешности развертывания на уровне администраторов (технические и функциональные тесты развертывания), а также интеграцию с процессами автоматизированного функционального, нагрузочного тестирования и динамического тестирования на безопасность.

**Непрерывное развертывание (CDP , Continuous deployment)** – практика разработки программного обеспечения, направленная на полную автоматизацию поставки от среды разработки в промышленную среду. Является дальнейшим развитием процесса Непрерывной поставки. Характеризуется тесным взаимодействием разработки и эксплуатации.

**Непрерывное измерение (CM, Continuous Measurement)** – это DevOps практика, которая заключается в постоянном отслеживании показателей всех систем и процессов, составляющих весь DevOps цикл. Она позволяет всем участникам процессов DevOps оперативно реагировать на любые изменения метрик, чем зачастую помогает выявлять и решать проблемы ещё до того, как ситуация станет критичной.

**Непрерывное тестирование (Continuous Testing)** – это практика выполнения автоматизированных тестов в рамках конвейера Continuous Delivery для получения немедленной обратной связи о результатах проверки правильности функционирования системы.Практика включает в себя все виды тестирований на не промышленных средах.

**3 вопрос (20 баллов): Методологии управления качеством.**

**Классификация.**

Управление качеством – непрерывный процесс целенаправленного воздействия на объекты управления в области качества, осуществляемый на всех этапах и стадиях жизненного цикла продукции (услуги), имеющий целью формирование, обеспечение и поддержание заданного (требуемого) уровня качества, удовлетворяющего требованиям потребителей и общества в

целом.

**Классификация методологий управления качеством ПС**

Методологии, целью которых является успешное выполнение отдельного проекта. К этой группе относится большинство проектных методологий и практически все адаптивные методологии. Логика успешного функционирования организации выглядит как рост компетенций, создание

технических активов за счет последовательного выполнения успешных проектов.

Методологии, обеспечивающие устойчивое функционирование компании-разработчика ПО, нацелены на последовательное развитие компетенций. К этой группе относятся модели зрелости (CMM, CMMI). Логика успешного функционирования предполагает создание, контроль и непрерывное улучшение способностей организации к выполнению проектов, и, как

следствие, успешное выполнение проектов

**Классификация методологий по характеру знаний:**

**Инженерные методологии** основываются на технологических принципах и направлены на совершенствование конечных продуктов, таких как программный код, тестовые примеры, прототипы, документация. Инженерные инструментальные необходимы

квалифицированному разработчику.

**Управленческие методологии** основываются на принципах теории управления (менеджмента), в их основе лежат такие концепции, как тотальное управление качеством, управление проектами, управление знаниями.

**Интегрированные методологии** средства объединяют инженерные концепции

и концепции управления.

**Прогнозируемые и адаптивные методологии:**

**Прогнозируемые методологии** основываются на предпосылке о возможности и целесообразности детального планирования будущего. Для ИТ-проекта формулируются требования к разрабатываемой системе, формируется план проекта и определяется потребность в ресурсах. Изменения в плане проекта и требованиях считаются нежелательными. Проектные методологии этого класса используют каскадную модель жизненного цикла.

**Адаптивные методологии** нацелены на преодоление ожидаемой неполноты требований и их постоянного изменения. В основе адаптивных методологий лежит итерационная модель жизненного цикла. Примером адаптивных методологий являются Scrum, Канбан, Crystal, Extreme Programming. Адаптивные методологии учитывают психологические особенности процесса разработки ПО. Одним из значимых факторов успеха использования адаптивных методологий является высокая квалификация специалистов, в первую очередь – разработчиков.

**№ 13**

**1 вопрос (20 баллов): Написать 15 пользовательский историй к веб-сайту**

**заказа авиабилетов с указанием критериев приемки и разделением**

**минимум на две роли.**

1. пользователь - менять местами город отправления и город прибытия, чтобы иметь возможность купить билет туда и обратно по отдельности

критерии приемки –

· кнопка стрелок между 2 полями ввода при нажатии меняет их значения местами

2. пользователь - просмотреть популярные направления, чтобы иметь возможность спланировать путешествие

критерии приемки –

· кнопка в шапке сайта должна переносить на страницу «популярное»

· страница популярное должна обновляться динамически

· страница популярное должна получать актуальную информацию о популярности направлений

3. пользователь - выбрать веганское питание, чтобы иметь возможность пообедать в течение длинного перелета

критерии приемки –

· при выборе места должен быть чекбокс «веганское питание»

· информация о питании должна заноситься в информацию о брони в системе

4. пользователь - купить билет туда и обратно, чтобы иметь возможность купить билеты быстро

· при выборе направления должен быть чекбокс «туда и обратно»

· результаты поиска обязаны быть валидными, т.е. не должно выдаваться невозможных сочетаний

5. пользователь - выбрать отель на странице заказа авиабилетов, чтобы купить все в одном месте

· при покупке билета должна быть кнопка «заказать проживание»

· при нажатии на кнопку должен происходить открытие новой вкладки с заказом проживания

6. пользователь - выбрать класс обслуживания, чтобы иметь возможность сравнить цены и выбрать подходящий тариф

· на странице покупки билета должна быть форма выбора класса обслуживания

· при выборе класса обслуживания должны отображаться места этого класса

7. пользователь - зарегистрироваться онлайн на рейс, чтобы иметь возможность не стоять в очереди на регистрацию в аэропорту

· на странице онлайн регистрации должны быть поля ввода фамилии и идентификатора билета

· формы должны проверять вводимые значения

8. пользователь - докупить багажное место, чтобы иметь возможность привести тяжелые предметы

• при покупке билета должна быть кнопка «докупить багажное место»

• при нажатии на кнопку должен происходить открытие новой вкладки с этим функционалом

9. пользователь - выбрать количество пассажиров, чтобы иметь возможность купить билеты для всей семьи

• при покупке билета должна быть кнопка «купить семейный билет»

• при нажатии на кнопку должен происходить открытие новой вкладки с этим функционалом

10. пользователь - составить сложный маршрут, чтобы иметь возможность сразу купить билет в несколько городов и стран

• при покупке билета должна быть кнопка «купить семейный билет»

• при нажатии на кнопку должен происходить открытие новой вкладки с этим функционалом

11. пользователь - войти в личный кабинет, чтобы иметь возможность смотреть историю своих заказов

· на главной странице сайта должна быть кнопка входа

· при нажатии на кнопку должен появляться оверлей входа

· информация о данных пользователей должна передаваться и храниться в системе

12. администратор - отобразить статистику покупок билетов по направлениям

· сервер должен логировать все покупки на сайте

· должна быть реализована система группировки информации по необходимым признакам

· должен быть дэшборд с возможностью выбора параметров агрегации статистики

13. администратор - отобразить статистику по возрасту пассажиров

• сервер должен логировать все покупки на сайте

• должна быть реализована система группировки информации по необходимым признакам

• должен быть дэшборд с возможностью выбора параметров агрегации статистики

14. администратор - отобразить нагрузку на сайт и её историю

· должен быть реализован мониторинг и логирование нагрузки

· дэшборд должен иметь вкладку со статистикой по нагрузке на сайт

15. администратор - отобразить статистику о количестве текущих пользователей сайта

• должен быть реализован мониторинг текущих пользователей сайта

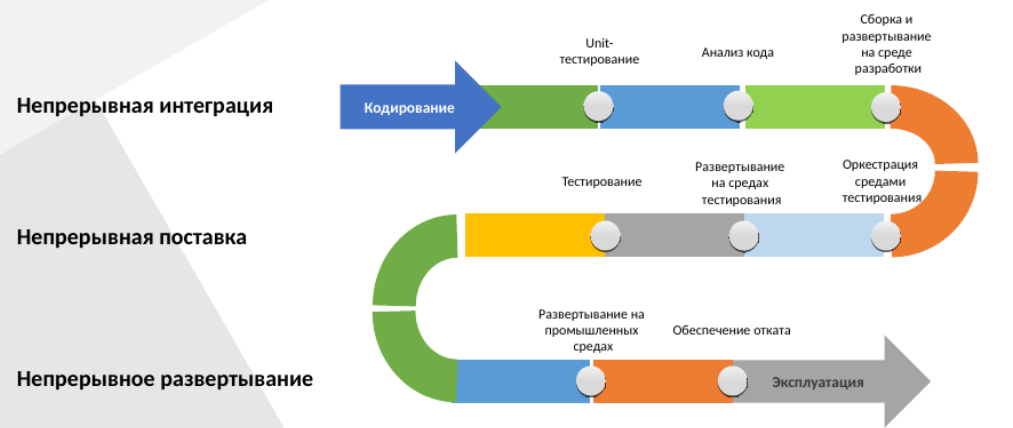
• дэшборд должен иметь вкладку со статистикой по текущим пользователям.

**2 вопрос (20 баллов): DevOps. Компоненты и процессы. Понятие безопасной**

**разработки.**

DevOps — это технологическая структура, которая обеспечивает взаимодействие между командами разработчиков и операционными командами для более быстрого развертывания кода в производственных средах с возможностью повторения действий и автоматизации.

Процессы DevOps



DevSecOps – это тот же DevOps, но с учетом включения в себя практик безопасной разработки.



**3 вопрос (20 баллов): Атрибуты качества систем и их связь с**

**функциональными требованиями. Сценарии атрибутов качества.**

Архитектурно-экономический цикл, как правило, основан на требованиях к качеству, определяемыми прямыми коммерческими мотивами, то есть, функциональностью разрабатываемой программы или программного комплекса. Это предполагает включения атрибутов качества как элементов требований к функциональности – при описании возможностей и поведения системы и оказываемых ей услуг. Функциональность становится ведущим показателем качества и занимает в схеме разработки центральное, или даже единоличное положение

**Классификатор атрибутов качества:**

1. Атрибуты качества системы. Будем рассматривать такие атрибуты как готовность, модифицируемость производительность, безопасность, контролепригодность и практичность
2. Коммерческие атрибуты качества (например, Time-to-Market, срок вывода продукта на рынок). Имеет смысл рассматривать только те из атрибутов, реализация которых определяется архитектурой системы
3. Атрибуты качества самой архитектуры (например, концептуальная целостность), которые влияют на другие качества – например, модифицируемость

Сценарием атрибута качества называется требование, путем выполнения которого этот атрибут реализуется.

Сценарий состоит из шести элементов:

1. Источник стимула. Это некий субъект (человек, вычислительная система) который порождает символ
2. Стимул. Стимулом называется наблюдаемое в системе явление, требующее к себе внимания
3. Условия. Стимул возникает при конкретных условиях. К примеру, система может находиться в состоянии перегрузки или исполняться в обычном режиме
4. Артефакт. Объектом воздействия стимула является некий артефакт. В этом качестве выступает система в целом или ее отдельный элемент
5. Реакция. Реакция – это действие, предпринятое в ответ на появление стимула
6. Количественная мера реакции. Предпринимаемые в ответ на стимул действия должны быть измеримы – только тогда соответствие требованию можно проверить

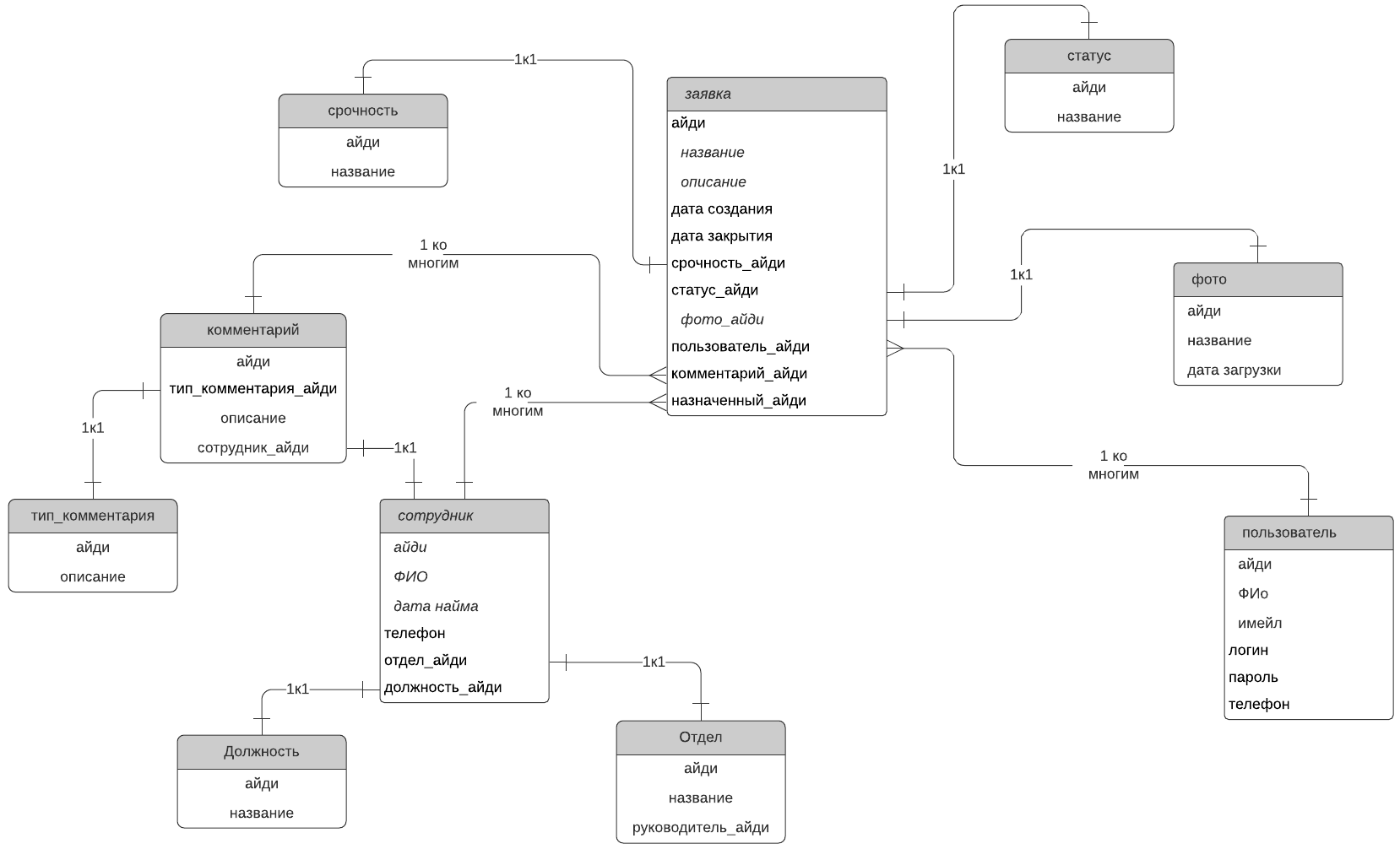
**№ 14**

**1 вопрос (20 баллов): Составить ER-диаграмму модели данных для сайта**

**продажи авиабилетов. Не менее 6 сущностей, общий состав атрибутов не**

**менее 40, связи много ко многим допускаются только с пояснением их**

**необходимости, минимум один пример связи один к одному.**

****

**2 вопрос (20 баллов): Понятие безопасной обработки. DevSecOps - как**

**встраивается в каждую практику.**

Цель DevSecOps состоит в том, чтобы сделать безопасность частью рабочего процесса разработки программного обеспечения с помощью передовых методов безопасного кодирования и автоматизации тестирования, а не закреплять ее на более поздних этапах цикла, как это было в случае с каскадными моделями разработки. Это позволяет предотвращать проблемы безопасности и нарушений комплайнса на наиболее ранних этапах жизненного цикла разработки. И разработчик уже при написании кода может получать рекомендации по обеспечению его безопасности.

DevSecOps решает наиболее распространенную проблему, с которой сталкивались многие компании на пути к DevOps, когда более быстрые выпуски кода приводили к большему количеству уязвимостей. Благодаря DevSecOps, критические проблемы безопасности решаются по мере их поступления, а не после возникновения угрозы или компрометации.

Осознание современных угроз и скрытых рисков, понимание того, как баг может повлиять на конечный продукт, знание средств и систем защиты – все это поможет выстроить некий план того, как необходимо внедрять DevSecOps-практики в каждодневную работу команды разработчиков и DevOps-инженеров.

Следующий фактор – это наличие бюджета на закупку и поддержку различных средств безопасности. Нужно понимать приоритетность внедрения решений, а также обладать специфической экспертизой в некоторых тонкостях реализации защиты.

Существует несколько ключевых точек при внедрении DevSecOps:

* Организация безопасного цикла разработки ПО, включающего моделирование угроз, оценку рисков, статический и динамический анализ кода.
* Усиление текущей облачной инфраструктуры с помощью нативных опций безопасности, а также различных коммерческих средств и систем защиты.
* Важность создания в компании внутренней базы знаний о проблемах безопасности.
* Обучение команды разработчиков правилам написания безопасного кода

**3 вопрос (20 баллов): Архитектурные принципы проектирование. Основные**

**принципы и подробно на примере 1 – 2 выбранных.**

Архитектурные принципы представляют собой фундаментальные «аксиомы» (или правила), которые используются в качестве «отправных точек» для принятия архитектурных решений. В свою очередь, архитектурные принципы являются подмножеством более общего понятия ИТ-принципов, которые определяют основные аспекты всей деятельности, связанной с применением информационных технологий.

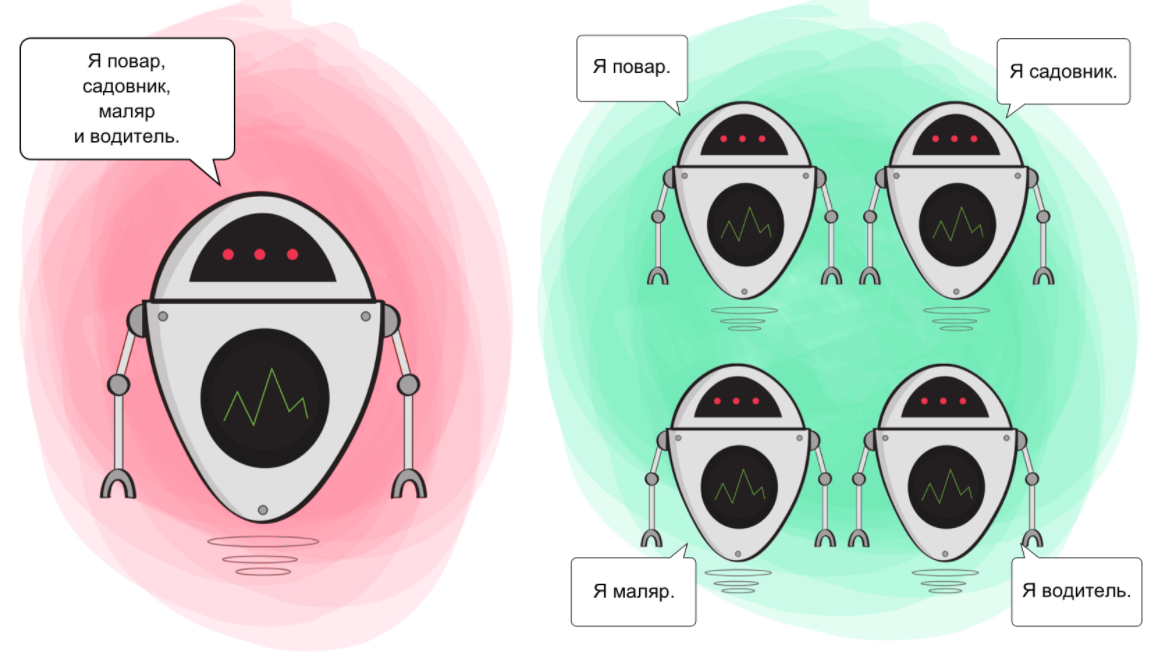
* Принцип единой ответственности (Single Responsibility Principle)
* Принцип наименьшего знания (Principle of Least Knowledge)
* Принцип низкой связанности (Low Coupling)
* Принцип сильного зацепления (High Cohesion)
* Принцип поддержания состояния (Stateful/Stateless)

**Принцип единой ответственности (Single Responsibility Principle):**

Принцип единой ответственности (SRP) говорит о том, что у каждого объекта есть своя ответственность и причина существования и эта ответственность у него только одна.

Это принцип, объясняющий КАК декомпозировать, то есть где провести линию разделения.

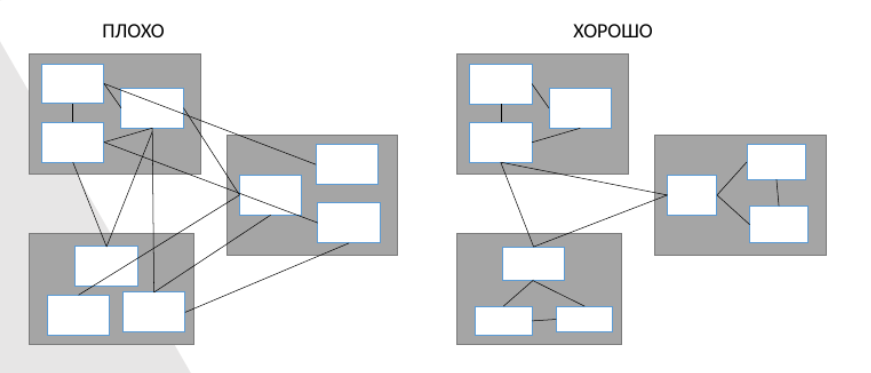
Он говорит, что декомпозировать надо по принципу разделения "ответственности", то есть по задачам тех или иных объектов.

****

**Низкая связаность (Low Coupling)**

Необходимо распределить ответственности между компонентами (классами, модулями) так, чтобы обеспечить минимальную связанность.

На UML — диаграмме классов для такой системы можно будет увидеть как зависимость класса A на класс B, так и зависимость класса B на класс A. Почему это плохо? Дело в том, что мы не можем отдать класс A без класса B, также как и класс B без класса A: их нельзя переиспользовать по — отдельности, только вместе. Чем меньше связей между классами — тем лучше.

****

**№ 15**

**1 вопрос (20 баллов): Написать 6 сценариев тестирования к порталу**

**ИТ-поддержки с общим количеством шагов не менее 40. Обязательно**

**должны быть использованы методы анализа граничных значений и**

**таблицы альтернатив.**

**ГРАНИЧНЫЕ ЗНАЧЕНИЯ**

Условие — в поля ввода можно внести только целые числа от 0 до 10 000.

Определяемся с существующими границами — так как в условии все значения от 0 до 10 000 приведут к одному и тому же результату, то границы две: нижняя и верхняя.

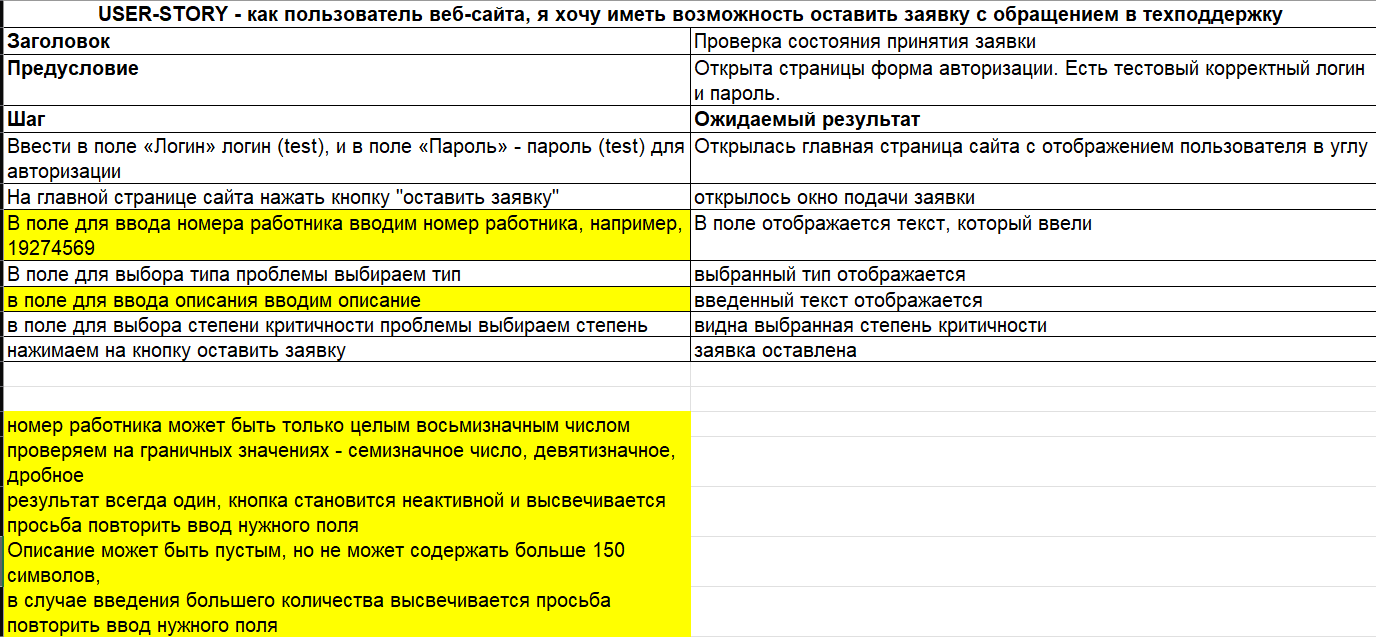
Первое граничное значение — 0

Второе граничное значение — 10 000

Добавляем к ним, стоящие рядом значения ( опять же, если бы, числа были дробными, то пришлось бы, для начала, определится с количеством знаков после запятой. Например, если десятые, то значения были бы: -0,1 0 0,1 и 9999,9 10000 10000,1):

-1, 0, 1

9 999, 10 000, 10 001

****

**ТАБЛИЦА РЕШЕНИЙ/АЛЬТЕРНАТИВ**

****

**2 вопрос (20 баллов): Управление конфигурациями в DevOps и ITIL.**

Управление конфигурациями в ITIL и DevOps называется по-разному:

В ITIL оно называется CMDB, в DevOps — система контроля версий, СВК.

ITIL (Information Technology Infrastructure Library — библиотека инфраструктуры информационных технологий) — это общепризнанный набор рекомендаций, призванный помочь организациям максимально эффективно использовать ИТ путем согласования ИТ-услуг с бизнес-стратегией. Появился в 80-х.

DevOps — это технологическая структура, которая обеспечивает взаимодействие между командами разработчиков и операционными командами для более быстрого развертывания кода в производственных средах с возможностью повторения действий и автоматизации.

В систему версионного контроля принято добавлять всё необходимое для (вос)создания любой из ваших сред: разработки, тестирования, продуктива. Включая компиляторы и дополнительные инструменты, используемые разработчиками, полный набор из операционных систем и всех необходимых компонет, стеков, файлов настройки зон DNS, конфигураций брендмауэров и прочее.

Построение управления конфигурациями согласно ITIL - это непростая затея, огромное число ручного труда предполагает наполнение и актуализация CMDB — вплоть до предания забвению результатов многомиллионного проекта с закупленным многомиллионным программным обеспечением.

"Правильный" DevOps позволяет за весьма короткое время (минуты) создать любую часть любой среды, вместо того, чтобы искать и устранять причины неработоспособности этой части этой среды (часы и дни).

**3 вопрос (20 баллов): Тактики обеспечения атрибутов качества (на примере одного**

**из типов Модифицируемость, Ремонтопригодность, Производительность,**

**Безопасность...).**

Тактика — это проектное решение, которое влияет на управление реакцией по атрибуту качества. Совокупность тактик называется архитектурной стратегией (architectural strategy).

Цели стратегий безопасности это строгое выполнение обязательств, конфиденциальность, целостность и гарантирование.

Достичь этих целей помогают следующие тактики:

• Аутентификация пользователей. Это проверка, является ли пользователь или удаленный

компьютер тем, за кого себя выдает

• Авторизация. Проверка, обладает ли аутентифицированный пользователь или процесс

достаточными правами на запрашиваемый ресурс.

• Обеспечение конфиденциальности данных. Например, шифрация каналов и данных

• Обеспечение целостности. Данные должны передаваться в неизменном виде. Содержащаяся в них служебная информация – например, контрольные суммы или хэши – может шифроваться

вместе с данными или отдельно.

• Минимизация подверженности внешним воздействиям. Распределение данных и служб по

независимым хостам

• Ограничения доступа. Проверка доступа на firewall с ограничением портов отправки и

назначения.

**Тактики восстановления** после атак могут быть ориентированы на восстановление предшествующего состояния или на идентификацию исполнителя атаки (последние можно причислить как к превентивным, так и к карательным мерам).

****

**№ 16**

**1 вопрос (20 баллов): Написать 6 сценариев тестирования к сайту по**

**продаже авиабилетов с общим количеством шагов не менее 40.**

**Обязательно должны быть использованы методы анализа граничных**

**значений и таблицы альтернатив.**

**ГРАНИЧНЫЕ ЗНАЧЕНИЯ**

Условие — в поля ввода можно внести только целые числа от 0 до 10 000.

Определяемся с существующими границами — так как в условии все значения от 0 до 10 000 приведут к одному и тому же результату, то границы две: нижняя и верхняя.

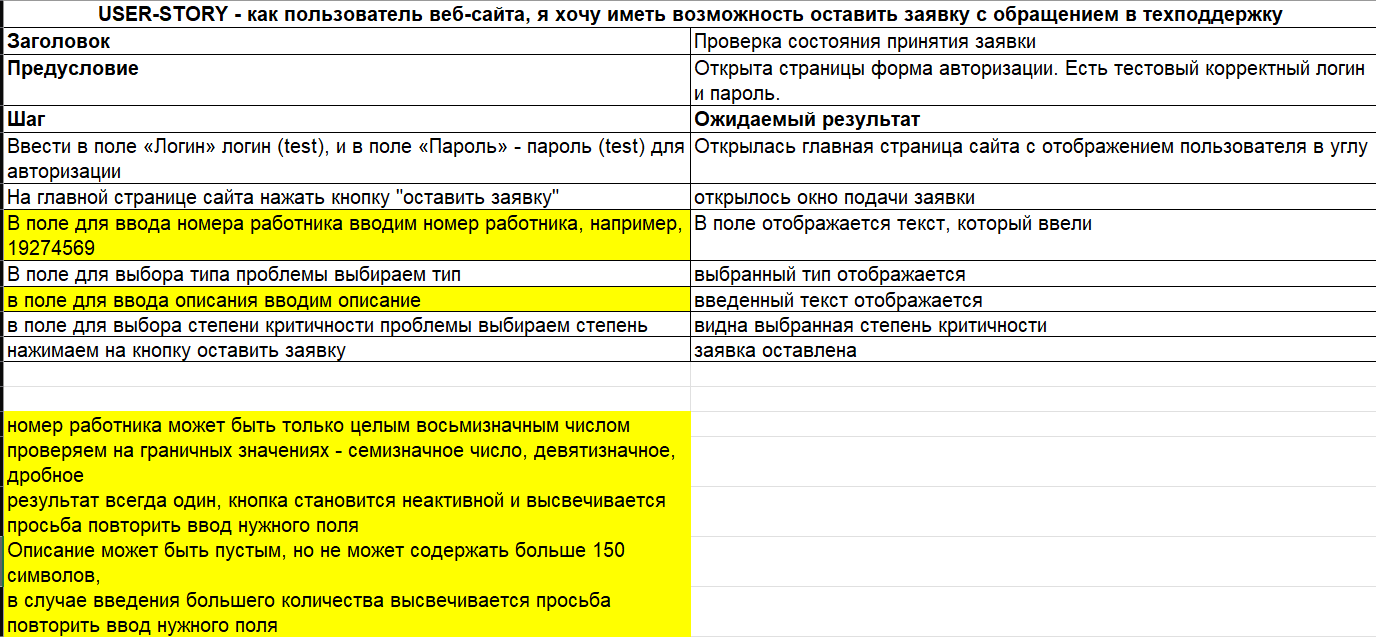
Первое граничное значение — 0

Второе граничное значение — 10 000

Добавляем к ним, стоящие рядом значения ( опять же, если бы, числа были дробными, то пришлось бы, для начала, определится с количеством знаков после запятой. Например, если десятые, то значения были бы: -0,1 0 0,1 и 9999,9 10000 10000,1):

-1, 0, 1

9 999, 10 000, 10 001

****

**ТАБЛИЦА РЕШЕНИЙ/АЛЬТЕРНАТИВ**

****

**2 вопрос (20 баллов): Тестирование. Методы черного и белого**

**ящика. Статическое и динамическое тестирование.**

**Black Box**

**Summary: Мы не знаем, как устроена тестируемая система.**

Тестирование методом «черного ящика», также известное как тестирование, основанное на спецификации или тестирование поведения – техника тестирования, основанная на работе исключительно с внешними интерфейсами тестируемой системы.

**тестирование черного ящика – это:**

– тестирование, как функциональное, так и нефункциональное, не предполагающее знания внутреннего устройства компонента или системы.

– тест-дизайн, основанный на технике черного ящика – процедура написания или выбора тест-кейсов на основе анализа функциональной или нефункциональной спецификации компонента или системы без знания ее внутреннего устройства.

Почему именно «черный ящик»? Тестируемая программа для тестировщика – как черный непрозрачный ящик, содержания которого он не видит. Целью этой техники является поиск ошибок в таких категориях:

– неправильно реализованные или недостающие функции;

– ошибки интерфейса;

– ошибки в структурах данных или организации доступа к внешним базам данных;

– ошибки поведения или недостаточная производительность системы;

Таким образом, мы не имеем представления о структуре и внутреннем устройстве системы. Нужно концентрироваться на том, *что* программа делает, а не на том, *как* она это делает.

**Пример**:

Тестировщик проводит тестирование веб-сайта, не зная особенностей его реализации, используя только предусмотренные разработчиком поля ввода и кнопки. Источник ожидаемого результата – спецификация.

Поскольку это *тип* тестирования, по определению он может включать другие его виды. Тестирование черного ящика может быть как функциональным, так и нефункциональным. Функциональное тестирование предполагает проверку работы функций системы, а нефункциональное – соответственно, общие характеристики нашей программы.

Техника черного ящика применима на всех *уровнях* тестирования (от модульного до приемочного), для которых существует спецификация. Например, при осуществлении системного или интеграционного тестирования, требования или функциональная спецификация будут основой для написания тест-кейсов.

Техники тест-дизайна, основанные на использования черного ящика, включают:

– классы эквивалентности;

– анализ граничных значений;

– таблицы решений;

– диаграммы изменения состояния;

– тестирование всех пар.

**Преимущества:**

– тестирование производится с позиции конечного пользователя и может помочь обнаружить неточности и противоречия в спецификации;

– тестировщику нет необходимости знать языки программирования и углубляться в особенности ПО

– тестирование может производиться специалистами, независимыми от отдела разработки, что помогает избежать предвзятого отношения;

– можно начинать писать тест-кейсы, как только готова спецификация.

**Недостатки:**

– тестируется только очень ограниченное количество путей выполнения программы;

– без четкой спецификации (а это скорее реальность на многих проектах) достаточно трудно составить эффективные тест-кейсы;

– некоторые тесты могут оказаться избыточными, если они уже были проведены разработчиком на уровне модульного тестирования;

**White Box**

**Summary: Нам известны все детали реализации тестируемой программы.**

Тестирование методом белого ящика (также: прозрачного, открытого, стеклянного ящика; основанное на коде или структурное тестирование) – метод тестирования программного обеспечения, который предполагает, что внутренняя структура/устройство/реализация системы известны тестировщику. Мы выбираем входные значения, основываясь на знании кода, который будет их обрабатывать. Точно так же мы знаем, каким должен быть результат этой обработки. Знание всех особенностей тестируемой программы и ее реализации – обязательны для этой техники. Тестирование белого ящика – углубление во внутренне устройство системы, за пределы ее внешних интерфейсов.

**тестирование белого ящика – это:**

– тестирование, основанное на анализе внутренней структуры компонента или системы.

– тест-дизайн, основанный на технике белого ящика – процедура написания или выбора тест-кейсов на основе анализа внутреннего устройства системы или компонента.

**Пример:**

Тестировщик, который, как правило, является программистом, изучает реализацию кода поля ввода на веб-странице, определяет все предусмотренные (как правильные, так и неправильные) и не предусмотренные пользовательские вводы, и сравнивает фактический результат выполнения программы с ожидаемым. При этом ожидаемый результат определяется именно тем, как должен работать код программы.

Тестирование методом белого ящика похоже на работу механика, который изучает двигатель машины, чтобы понять, почему она не заводится.

Техника белого ящика применима на разных уровнях тестирования – от модульного до системного, но главным образом применяется именно для реализации модульного тестирования компонента его автором.

**Преимущества:**

– тестирование может производиться на ранних этапах: нет необходимости ждать создания пользовательского интерфейса;

– можно провести более тщательное тестирование, с покрытием большого количества путей выполнения программы.

**Недостатки:**

– для выполнения тестирования белого ящика необходимо большое количество специальных знаний

– при использовании автоматизации тестирования на этом уровне, поддержка тестовых скриптов может оказаться достаточно накладной, если программа часто изменяется.

**Сравнение Black Box и White Box**

**Статическое тестирование** – тип тестирования, который предполагает, что программный код во время тестирования не будет выполняться. При этом само тестирование может быть как ручным, так и автоматизированным.

Статическое тестирование начинается на ранних этапах жизненного цикла ПО и является, соответственно, частью процесса верификации.

Большинство статических техник могут быть использованы для «тестирования» любых форм документации, включая вычитку кода, инспекцию проектной документации, функциональной спецификации и требований.

Виды статического тестирования:

– вычитка исходного кода программы;

– проверка требований.

**Динамическое тестирование** – тип тестирования, который предполагает запуск программного кода. Таким образом, анализируется поведение программы во время ее работы.

Для выполнения динамического тестирования необходимо чтобы тестируемый программный код был написан, скомпилирован и запущен. При этом может выполняться проверка внешних параметров работы программы: загрузка процессора, использование памяти, время отклика и т.д. – то есть, ее производительность.

Динамическое тестирование является частью процесса валидации программного обеспечения.

Кроме того, динамическое тестирование может включать разные подвиды, каждый из которых зависит от:

• Доступа к коду (тестирование черным, белым и серым ящиками).

• Уровня тестирования (модульное, интеграционное, системное и приемочное тестирование).

• Сферы использования приложения (функциональное, нагрузочное, тестирование безопасности и пр.).

**3 вопрос (20 баллов): Архитектурные стили на уровне компонентов. Слоистая и**

**модульная архитектура.**

**«Слоистая» архитектура**

• Может быть логической и физической

• Структура системы разделяется на несколько слоев

• Каждый слой работает поверх следующего

• Есть четко описанные интерфейсы между слоями

• Уменьшается сложность, увеличивается модульность, упрощается сопровождение

• Различные критерии деления на слои

**Модульная архитектура**

**Почему:**

• Модульность -> более обслуживаемая и легкая в подержке система

• Приложения должны иметь возможности расширения функционала без простоя

**Модульность:**

• Модуль как независимая единица поставки

• У модуля есть имя, версия и список зависимостей

• Взаимодействие с другими модулями через четкий интерфейс, детали реализации скрыты

• Модульность уровня DesignTime и RunTime

**Проблемы:**

• Объекты, пакеты и архивы не являются независимыми

**Решение:**

• Контейнеризация

**№ 17**

**1 вопрос (20 баллов): Рассматривая работу над проектом создания сайта**

**заказа билетов, разработать матрицу заинтересованных лиц и**

**представлений для формирования пакета документации по проекту. Для**

**этой цели выделить группы заинтересованных лиц (не менее 6) и типы**

**документов (примеры общая архитектурная схема, диаграмма**

**развертывания) не менее 10. В матрице в ячейках пересечения документа и группы заинтересованных лиц проставить значения:**

**2 - документ основной для данной группы**

**1 - документ доводится до сведения участникам данной группы**

**пусто - данный документ не предназначен для данной группы**

**Для нескольких типов документов описать предлагаемые нотации (с пояснением принципа выбора).**

****

**Инженеры будут разрабатывать ПО, поэтому все прикладные диаграммы напрямую относятся к ним, оформлены они будут в UML.**

**Системные администраторы уже будут руководствоваться более высокоуровневыми диаграммами, также оформленными в UML в силу удобства (IDEF0), но так как DFD – зона их прямой ответственности, DFD будет оформлена в ERD**

**Руководство интересует простое и четкое изложение сути проекта, для них всё будет оформлено в boxes and lines**

**Архитекторы будут пользоваться AADL для четкой структурированности системы.**

**Для менеджера проектов основным инструментом может стать например MS Project**

**2 вопрос (20 баллов): Тестирование. Уровни и типы тестирования.**

**Тестирование программного обеспечения – это:**

– процесс исследования ПО с целью получения информации о качестве продукта;

– **процесс проверки соответствия заявленных к продукту требований и реально реализованной функциональности, осуществляемый путем наблюдения за его работой в искусственно созданных ситуациях и на ограниченном наборе тестов, выбранных определенным образом;**

– оценка системы с тем, чтобы найти различия между тем, какой система должна быть и какой она есть.

В широком смысле тестирование – это одна из техник контроля качества (Quality Control), которая включает планирование, составление тестов, непосредственно выполнение тестирования и анализ полученных результатов.

Важно понимать, что тестирование ПО включает не только собственно проведение тестов, но и многие другие действия, связанные с процессом обеспечения качества:

· анализ и планирование;

· разработку тестовых сценариев;

· оценку критериев окончания тестирования;

· написание отчетов;

· рецензирование документации (в том числе и исходного кода);

· проведение статического анализа.

Уровни тестирования

***Модульное тестирование* (Автономное или Unit-тестирование)**

Входные требования — Архитектура компонентов или модель “нижнего уровня” системы (Component Design или Low Level Design)

Объект тестирования — Разработанные компоненты

Определение: На данном уровне тестируются по отдельности небольшие элементы системы, максимально отделенные от других элементов и, в то же время, пригодные для тестирования. Такое тестирование обычно проводится сразу же вслед за разработкой каждого из элементов и направлено на оценку соответствия функциональности каждого из компонентов спроектированной “модели компонентов”.

***Комплексное тестирование* (Сборочное тестирование, integration testing или interface testing)**

Входные требования — Архитектура системы или модель “верхнего уровня” системы (System Design или High Level Design)

Объект тестирования — Собранная из компонентов система или подсистема

Определение: на данном уровне тестируются объединенные элементы (компоненты или подсистемы) общей системы, чаще всего некоторая взаимодействующая между собой группа элементов.

Комплексное тестирование направлено не на проверку функционирования каждого из компонентов, а на проверку взаимодействия компонентов в соответствии с «Архитектурой системы».

Тесты данного уровня обычно проверяют все интерфейсы взаимодействия между компонентами, определенные в системной архитектуре, до тех пор, пока все компоненты не будут разработаны, отлажены и проинтегрированы друг с другом в единую систему.

***Системное тестирование* (system testing)**

Входные требования — Системные спецификации (System Specification)

Объект тестирования — Разработанная система

Определение: после того, как система собрана из составляющих компонентов, она должна быть протестирована на соответствие “Системным спецификациям” – реализованы ли все функциональные и нефункциональные требования к разрабатываемой системе.

На данном уровне тестируется приложение или система (одно или более приложений) целиком.

***Приемочное тестирование* (Приемо-сдаточное тестирование или acceptance testing)**

Входные требования — Требования (Requirements)

Объект тестирования — Разработанная система

Определение: на данном уровне завершенное приложение (система) тестируется Заказчиком, конечными пользователями или соответствующими уполномоченными с целью определения соответствия системы “Требованиям Заказчика” и готовности системы к внедрению. Приемосдаточные испытания оформляют процесс передачи продукта от Разработчика Заказчику. В зависимости от особенностей продукта и от требований Заказчика они могут проводиться в различной форме. Например, в виде альфа- или бета-тестирования.

Приемочное тестирование схоже с системным тестированием, но со следующим различием:

· Системное тестирование проверяет, что разработанная система соответствует специфицированным требованиям;

· Приемочное тестирование проверяет, что разработанная система удовлетворяет запрошенным Заказчиком требованиям с упором на нужды конечных пользователей в данной предметной области.

***Операционное тестирование* (Release Testing)**

Входные требования — Бизнес-модель (Business Case или Business Model)

Объект тестирования — Разработанная система

Определение: даже если система удовлетворяет всем требованиям, важно убедиться в том, что она удовлетворяет нуждам пользователя и выполняет свою роль в среде своей эксплуатации, как это было определено в бизнес-модели системы. Следует учесть, что и Бизнес-модель может содержать ошибки. Поэтому так важно провести операционное тестирование как финальный шаг Валидации.

Кроме этого, тестирование в среде эксплуатации позволяет выявить и нефункциональные проблемы, такие как: конфликт с другими системами, смежными в области бизнеса или в программных и электронных окружениях; недостаточная производительность системы в среде эксплуатации и др.

Очевидно, что нахождение подобных вещей на стадии внедрения – критичная и дорогостоящая проблема. Поэтому так важно проведение не только верификации, но и валидации, с самых ранних этапов разработки ПО.

Виды тестирования

***Инсталляционное тестирование* (Installation testing)**

Определение: В процессе инсталляционного тестирования проверяется корректность установки и деинсталляции программного продукта в среде максимально приближенной к эксплуатационной. Проверка правильности установки программного продукта должна быть обязательным элементом проекта по тестированию любого продукта.

Цель: Основная цель состоит в том, чтобы убедиться, что продукт может быть установлен/деинсталлирован при различных условиях – таких как: новая инсталляция, усовершенствование системы (upgrade), установка по умолчанию, полная установка, установка по выбору.

***Дымное тестирование* (проверка на дым, Smoke testing)**

Определение: Первый прогон программы (после написания или после внесения существенных изменений). Как правило, используется для определения, готова ли программа для проведения более обширного тестирования.

Цель: Выявление проблем «лежащих на поверхности» – тестируется чаще всего основная бизнес-логика программы

***Функциональное тестирование* (Functional testing)**

Определение: Проверка соответствия продукта функциональным требованиям и спецификациям

Цель: Проверка соответствия продукта функциональным требованиям и спецификациям

***Регрессионное тестирование* (Regression testing)**

Определение: Повторное тестирование после внесения изменений в программное обеспечение или в его окружение (в новой версии приложения), чтобы убедиться в том, что функции, которые работали в предыдущей версии системы, по-прежнему работают так, как ожидалось, а найденные дефекты успешно исправлены (все протестированное ранее тестируется повторно)

Цель: Выявление потенциальных проблем, которые могли возникнуть в результате изменений. Проверка исправления найденных ранее дефектов.

***Интеграционное тестирование* (Integration testing)**

Определение: Проверка скомбинированных компонентов прикладной программы с целью определения корректности их совместного функционирования

Цель: Выявление потенциальных проблем в совместном функционировании компонент

***Тестирование графического интерфейса пользователя* (User Interface testing)**

Определение: Тестирование интерфейса – экранов, кнопок и т.д. Большая часть функциональности ПО реализуется, как правило, через пользовательский интерфейс.

Цель: Обнаружение ошибок в интерфейсе и поиск ошибок в функциональности посредством интерфейса

***Тестирование производительности* (Performance testing)**

Определение: Проверка скорости работы системы (время отклика, частота транзакций и другие зависящие от времени) в имитационной и реальной средах

Цель: Установить реальную производительность программного продукта

***Нагрузочное тестирование* (Load testing)**

Определение: Это те же тесты производительности, при которых система подвергается различным нагрузкам; при этом цель этого тестирования – оценить способность системы правильно функционировать при некотором превышении планируемых нагрузок при реальной эксплуатации (система имеет некоторый «запас прочности»)

Цель: Убедиться в том, что система работает соответственно ожидаемым рабочим нагрузочным параметрам (какой предел работоспособности)

***Стресс тестирование* (Stress testing)**

Определение: Является одним из разновидностей тестирования на производительность. Проверяется поведение системы при недостатке ресурсов (дискового пространства, обрывов сети и т.д.).

Цель: Проверка того, что система адекватно реагирует на те или иные стрессовые ситуации

***Конфигурационное тестирование* (Configuration testing)**

Определение: Конфигурационное тестирование – тестирование работы на различных платформах. Различные варианты аппаратной конфигурации, версии операционной системы и окружения.

Цель: Проверить работоспособность системы при различных конфигурациях

***Тестирование интернационализации* (Internationalization testing)**

Определение: Этот вид тестирует насколько продукт готов к тому, чтобы быть адаптированном для работы в других локалях с другим языком пользовательского интерфейса, отличном от языка по умолчанию (как правило, это английский)

Цель: Проверить способность продукта быть быстро локализованным под необходимую локаль потенциальных пользователей системы

***Локализационное тестирование* (Localization testing)**

Определение: Локализационное тестирование, в свою очередь, проверяет, правильно ли локализован продукт. То есть, переведен на другой язык и корректно работает с учетом национальных особенностией страны или региона, в котором будет продаваться и использоваться продукт.

Цель: Проверить, правильно ли локализован продукт

**3 вопрос (20 баллов): Рефакторинг. Назначение и принципы.**

**Рефакторинг** представляет собой процесс такого изменения программной системы, при котором не меняется внешнее поведение кода, но улучшается его внутренняя структура. Это способ систематического приведения кода в порядок, при котором шансы появления новых ошибок минимальны.

**Назначение рефакторинга**

§ Рефакторинг улучшает композицию программного обеспечения

§ Рефакторинг облегчает понимание программного обеспечения

§ Рефакторинг помогает найти ошибки

§ Рефакторинг позволяет быстрее писать код

**Принципы рефакторинга**

**Рефакторинг (Refactoring):** изменение во внутренней структуре программного обеспечения, имеющее целью облегчить понимание его работы и упростить модификацию, не затрагивая наблюдаемого поведения.

**Производить рефакторинг (Refactor):** изменять структуру программного обеспечения, применяя ряд рефакторингов, не затрагивая его поведения.

Рефакторинг не меняет видимого поведения программного обеспечения. Оно продолжает выполнять прежние функции. Никто — ни конечный пользователь, ни программист — не сможет сказать по внешнему виду, что что-то изменилось.

**№ 18**

**1 вопрос (20 баллов): Рассматривая работу над проектом создания портала IT поддержки, разработать матрицу заинтересованных лиц и представлений для формирования пакета документации по проекту. Для этой цели выделить группы заинтересованных лиц (не менее 6) и типы документов (примеры общая архитектурная схема, диаграмма развертывания, диаграмма класса) не менее 10. В матрице в ячейках пересечения документа и группы заинтересованных лиц проставить значения:**

**2 - документ основной для данной группы**

**1 - документ доводится до сведения участникам данной группы**

**пусто - данный документ не предназначен для данной группы**

**Для нескольких типов документов описать предлагаемые нотации (с пояснением принципа выбора).**

|  | Диаграмма классов | Диаграмма компонентов | Диаграмма состояний | Диаграмма вариантов использования | Диаграмма композитной структуры | DFD | IDEF0 | план управления проектом | Техническое задание | план управления рисками |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ИТ-инженеры | 2 | 2 | 2 | 2 | 2 | 2 | 2 |  |  |  |
| Системные администраторы | 1 | 1 | 1 |  |  |  |  |  |  |  |
| Высшее руководство |  |  |  |  |  |  |  | 2 |  |  |
| Менеджер проектов |  |  |  |  |  |  |  | 1 | 1 | 2 |
| Пользователи |  |  |  |  |  |  |  |  |  |  |
| Архитектор | 1 | 1 | 1 | 1 | 1 | 1 | 1 |  | 1 |  |
| Заказчик |  |  |  |  |  |  |  |  | 2 | 1 |

**2 вопрос (20 баллов): Тестирование. Методы и инструменты.**

**Классификация дефектов.**

Широко используемыми методами тестирования являются модульное тестирование, интеграционное тестирование, приемочное тестирование, и тестирование системы. Программное обеспечение подвергается этим испытаниям в определенном порядке.

1. Модульное тестирование

2. Интеграционное тестирование

3. Системное тестирование

4. Приемочные испытания

**Модульное тестирование**

В первую очередь проводится модульный тест. Как подсказывает название, это метод испытания на объектном уровне. Отдельные программные компоненты тестируются на наличие ошибок. Для этого теста требуется точное знание программы и каждого установленного модуля. Таким образом, эта проверка осуществляется программистами, а не тестерами. Для этого создаются тест-коды, которые проверяют, ведет ли программное обеспечение себя так, как задумывалось.

**Интеграционное тестирование**

Отдельные модули, которые уже были подвергнуты модульному тестированию, интегрируются друг с другом, и проверяются на наличие неисправностей. Такой тип тестирования в первую очередь выявляет ошибки интерфейса. Интеграционное тестирование можно осуществлять с помощью подхода "сверху вниз", следуя архитектурному сооружению системы. Другим подходом является подход «снизу вверх», который осуществляется из нижней части потока управления.

**Системное** **тестирование**

В этом методе вся система проверяется на наличие ошибок и багов. Этот тест осуществляется путем сопряжения аппаратных и программных компонентов всей системы, и затем выполняется ее проверка. Это тестирование числится под методом тестирования "черного ящика", где проверяются ожидаемые для пользователя условия работы программного обеспечения.

**Приемочные испытания**

Это последний тест, который проводится перед передачей программного обеспечения клиенту. Он проводится, чтобы гарантировать, что программное обеспечение, которое было разработано отвечает всем требованиям заказчика. Существует два типа приемо-сдаточных испытаний - то, которое осуществляется членами команды разработчиков, известно, как внутреннее приемочное тестирования (Альфа-тестирование), а другое, которое проводится заказчиком, известно, как внешнее приемочное тестирования. Если тестирование проводится с помощью предполагаемых клиентов, оно называется приемочными испытаниями клиента. В случае если тестирование проводится конечным пользователем программного обеспечения, оно известно, как приемочное тестирование (бета-тестирование).

**Инструменты тестировщика**

**1.Управление тестированием**

Для ручного тестирования ПО используется множество разнообразных инструментов. Например, управление тестированием может вестись в таких специализированных системах, как Redmine, Jira, HP ALM, IBM Rational Quality Manager, MS Team Foundation Server, TestRail, TestLink и других.

Если тестирование выполняется на условиях аутсорсинга (проектная команда работает на территории заказчика), то выбор в пользу того или иного инструмента, связанного с написанием кейсов, обычно зависит от того, что использует (или хочет использовать) сам заказчик. Тем не менее, есть базовый инструментарий, который можно использовать практически в любом проекте, его мы и рассмотрим ниже.

**2.Текстовые, XML-редакторы и файловые менеджеры**

Для поиска, конвертации и сравнения файлов очень удобно использовать Notepad++ или PSPad. Первый — это почти тот же Блокнот для Windows, только с открытым исходным кодом и поддерживающий синтаксис большого числа языков программирования (написан на C++). Второй — редактор для текста и кода, позволяющий одновременно работать над несколькими документами и использовать несколько языков программирования, а также создавать шаблоны для автоматизации повторяющихся действий.

Хорошие XML-редакторы: Altova XML Spy (работа с XML и XSD), а также XMLPad (с отладчиком XSL). Среди файловых менеджеров можно выделить Total Commander (для Windows и Android, может соединяться с FTP-серверами), Free Commander (бесплатный, с возможностью группового переименования и сравнения файлов), trolCommander (кроссплатформенный, написан на Java) и Far Manager (консольный).

**3.Генераторы данных**

Практика показывает, что многие тестировщики используют в своей работе текстовые файлы с одними и теми же тестовыми данными. К сожалению, это приводит к тому, что баги «привыкают» к тестам и со временем перестают находиться («эффект пестицида»). Чтобы не ломать голову перед каждым проектом, можно использовать специальные инструменты, которые умеют генерировать данные. Например, Mockaroo. Это приложение умеет не только подбирать данные (допустим, username), но и генерировать SQL-запросы.

Еще один отличный инструмент, незаменимый для исследовательского тестирования, — плагин Bug Magnet. Он подходит для браузеров Chrome и Firefox, и содержит предустановленные наборы тестовых данных, которые, к тому же, предусмотрительно разбиты на группы (язык, формат, длина). Помимо сохраненных данных, можно добавлять и использовать свои.

**4.Инструменты для работы со снимками экранов и записи видео с содержимым**

Для ручного тестирования ПО скриншотеры просто необходимы. Пожалуй, один из лучших — GreenShot — бесплатный, с открытым исходным кодом. Кроме того, есть Snagit, ScreenHunter, Snipping Tool, Monosnap и многие другие.

Что касается инструментов для записи видео с содержимым экрана, то одни из лучших — Free Screen Video Recorder и Ashampoo Snap. Среди любопытных решений — CamStudio, Jing, которые позволяют делать скринкасты (видеозахват экрана с речевым сопровождением).

Иногда бывает нужна анимация (GIF-файлы). В этих случаях помогут LICEcap и Recordit, которые легко и быстро сохраняют записанное видео в GIF.

Тем, кому приходится иногда сравнивать графические файлы, стоит обратить внимание на ImageDupeless, ImageDiscerner и FastStone Image Viewer.

**5.Планировщики задач и интеллект-карты**

Для того, чтобы все проектные задачи были выполнены в срок, важно правильно их распределить. Ставить задачи, получать напоминания и вычеркивать из списка выполненное можно даже с помощью корпоративной почты в MS Outlook. Хотя гораздо эффективнее использовать для этого специальные инструменты, вроде Evernote. Для планирования командной работы отлично подходит Trello.

Поскольку наш мозг не всегда хорошо воспринимает информацию в виде текстов, списков и таблиц, ее можно визуализировать. Диаграмма связей или интеллект-карта — отличный метод структурирования задач (а также мыслей и планов). Если добавить в Mind Map данные о тестах, которые нужно провести, то сразу станет понятно, сколько их, какие между ними связи, есть ли среди них что-то лишнее или недостающее. С помощью интеллект-карт так же можно изобразить порядок своих действий, этапы работы и другое.

Есть много онлайн-инструментов для создания таких карт, но могу выделить MindJet (добавляет информацию с разных сайтов, приложений, платформ и контактов), MindMup (бесплатный, легко сохранять карты и делиться), Coggle (простой в освоении), Bubbl.us (онлайн редактор и приложение) и XMind (серьезный софт для построения ментальных карт и различных диаграмм).

**6.Чек-листы**

Чек-листы — это простой, но эффективный способ не только для упорядочивания повторяющихся действий, но и для самопроверки. Можно использовать уже готовые списки, составленные и любезно выгруженные другими тестировщиками в сеть, а можно подготовить свои собственные и использовать их, например, перед приемочным тестированием, для юзабилити-тестирования и т.д.

Один из самых удобных инструментов для создания чек-листов — Sitechco. Это бесплатный онлайн-сервис, в котором легко завести свои чек-листы, хранить результаты, делиться ими с командой, просматривать отчетность и статистику.

**Классификация дефектов**

*Классификация дефектов, с точки зрения* **степени влияния (Severity)** *на работоспособность ПО:*

**Blocker.** Ошибка, которая приводит программу в нерабочее состояние. Дальнейшая работа с программной системой или ее функциями – невозможна.

**Critical.** Критический дефект, приводящий некоторый ключевой функционал в нерабочее состояние. Так же это может быть существенное отклонение от бизнес логики, неправильная реализация требуемых функций, потеря пользовательских данных и т.д.

**Major.** Весьма серьезная ошибка, свидетельствующая об отклонении от бизнес логики или нарушающая работу программы. Не имеет критического воздействия на приложение.

**Minor.** Незначительный дефект, не нарушающий функционал тестируемого приложения, но который является несоответствием ожидаемому результату. Например, ошибка дизайна.

**Trivial.** Баг, не имеющий влияние на функционал или работу программы, но который может быть обнаружен визуально. Например, ошибка в тексте.

*Градация дефектов, с точки зрения* **приоритетности исправления (Priority)***:*

**High.** Баг должен быть исправлен как можно быстрее, т.к. он критически влияет на работоспособность программы.

**Medium.** Дефект должен быть обязательно исправлен, но он не оказывает критическое воздействие на работу приложения.

**Low.** Ошибка должна быть исправлена, но она не имеет критического влияния на программу и устранение может быть отложено, в зависимости от наличия других более приоритетных дефектов.

С помощью такой классификации организована работа многих систем отслеживания ошибок, в том числе Jira.

**3 вопрос (20 баллов): Метрики программного кода. Основные принципы и**

**подходы, инструменты, практика применения.**

Метрика — это количественная мера позволяющая оценить, в какой степени система, компонент системы или процесс обладают заданным качеством.

Основные **цели** применения метрик программного кода:

· Измерение качества продукта либо процесса

· Прогнозирование качества продукта либо процесса

· Повышение качества продукта либо процесса.

В общем случае применение метрик позволяет руководителям проектов и предприятий изучить сложность разработанного или даже разрабатываемого проекта, оценить объем работ, стилистику разрабатываемой программы и усилия, потраченные каждым разработчиком для реализации того или иного решения. В частности, применение метрик позволяет решать такие задачи как:

· Оценка производительности применения новых средств и методов

· Определение тенденций производительности с течением времени

· Оценка стоимости и графика будущих проектов

· Прогноз будущих потребностей в персонале

· Контроль стабильности качества ПО

· Прогноз стоимости владения (поддержки и развития) ПО

Метрики сложности программ принято разделять на 4 основные группы:

· **Количественные метрики**

· **Метрики сложности потока управления программы**

· **Метрики сложности потока управления данными**

· **Метрики объектно-ориентированных систем**

В пакет VisualStudio входит специальный

инструмент Code Metrics, собирающий информацию о четырех видах метрик для приложений из результирующего IL-кода:

– цикломатическая сложность (Cyclomatic Complexity) – число линейно независимых маршрутов через программный код;

– глубина наследования (Depth of Inheritance)

– число наследуемых классов;

– сцепление классов (Class Coupling) – зависимость классов друг от друга;

– количество строк кода (Lines of Code) –

поскольку подсчет производится именно в ILкоде, то не учитываются строки с фигурными

скобками, комментарии и т. п., зато сжатый синтаксис (например, LINQ-выражения) может развернутся на несколько строк. Также в нем рассчитывается комплексный показатель качества кода MI (Maintainability Index), принимающий значения от 0 до 100.

Расчет происходит по формуле:

MI = MAX (0, (171 − 5, 2 \* ln(HV ) + −0, 23 \*CC −16, 2 \* ln(LoC)) \*100 /171))

где

HV (Halstead Volume) – вычислит. сложность, пропорциональная числу операторов в коде;

CC – цикломатическая сложность;

LoC – количество строк кода.

Чем ниже показатель MI, тем сложнее производить поддержку кода (его анализ). Значения

от 20 до 100 считаются удовлетворительными для понимания,

от 10 до 19 – нежелательными,

от 0 до 9 – тяжелыми для понимания.

**Количественные метрики**

LOC-оценка (Lines Of Code) — это самая простая и старая методика, основанная на подсчете строк кода (физических – SLOC или логических DSI). Эти метрики чувствительны к используемому языку программирования, что накладывает ограничения на их применение.

Показатели, по смыслу близкие к LOC-оценке и объединенные в одно «семейство»:

· количество пустых строк,

· количество комментариев,

· процент комментариев (отношение числа строк, содержащих комментарии к общему количеству строк, выраженное в процентах),

· среднее число строк для функций (классов, файлов),

· среднее число строк, содержащих исходный код для функций (классов, файлов),

· среднее число строк для модулей.

Дополнительные показатели, которые можно рассматривать как аналог LOC:

· общие трудозатраты (в человеко-месяцах, человеко-часах);

· объем документации;

· ошибки, обнаруженные в течение года эксплуатации;

· количество людей, работавших над проектом;

· срок разработки.

Метрики Хольстеда (Haulsted)

• **n=n1+n2 — словарь программы**

• **N=N1+N2 — длина программы**

• **n'=n1'+n2' — теоретический словарь программы**

• **N'= n1\*log2(n1) + n2\*log2(n2) — теоретическая длина программы** (для стилистически корректных программ отклонение N от N' не превышает 20%)

• **V=N\*log2n — объем программы**

• **V'=N'\*log2n' — теоретический объем программы**, где n\* — теоретический словарь программы

• **L=V'/V — уровень качества программирования,** для идеальной программы L=1

• **L'= (2 \* n2)/ (n1\*N2) — уровень качества программирования**, основанный лишь на параметрах реальной программы без учета теоретических параметров

• **EC=V/(L') \* 2 — сложность понимания программы,**

• **D=1/ L' — трудоемкость кодирования программы,**

• **I=V/D — информационное содержание программы**, данная характеристика позволяет определить умственные затраты на создание программы

• **E=N' \* log2(n/L) — оценка необходимых интеллектуальных усилий при разработке программы,** характеризует число требуемых элементарных решений при написании программы

Данные метрики основаны на следующих показателях:

· n1 — число уникальных операторов программы (словарь операторов)

· n2 — число уникальных операндов программы (словарь операндов),

· N1 — общее число операторов в программе,

· N2 — общее число операндов в программе,

· n1' — теоретическое\* число уникальных операторов,

· n2' — теоретическое\* число уникальных операндов.

\* Теоретическое – относится к программе, в которой все частные алгоритмы реализованы в самом языке.

**Метрики сложности потока управления программ**.

Это - класс метрик, основанный на анализе **управляющего графа программы**.

Пусть представлена некоторая программа. Для данной программы строится ориентированный граф, содержащий лишь один вход и один выход, при этом вершины графа соотносят с теми участками кода программы, в которых имеются лишь последовательные вычисления, и отсутствуют операторы ветвления и цикла, а дуги соотносят с переходами от блока к блоку и ветвями выполнения программы. Условие при построении данного графа: каждая вершина достижима из начальной, и конечная вершина достижима из любой другой вершины.

Такой граф носит название **Управляющего графа**

Цикломатическая сложность

Самой распространенной оценкой, основанной на анализе графа управления, является цикломатическая сложность программы (цикломатическое число Мак-Кейба).

Цикломатическая сложность определяется как

**V(G) = e — n + 2p**

где e — количество дуг

n — количество вершин

p — число компонент связности.

Число компонентов связности графа можно рассматривать как количество дуг, которые необходимо добавить для преобразования графа в сильно связный.

Для графов корректных программ, т. е. графов, не имеющих недостижимых от точки входа участков и «висячих» точек входа и выхода, сильно связный граф, как правило, получается путем замыкания дугой вершины, обозначающей конец программы, на вершину, обозначающую точку входа в эту программу, и формула для расчета цикломатической сложности приобретает вид: **V(G)=e — n + 2**

Дополнительные методики на основе графа управления

К сожалению, оценка цикломатической сложности не способна различать циклические и условные конструкции. Еще одним существенным недостатком подобного подхода является то, что программы, представленные одними и теми же графами, могут иметь совершенно разные по сложности предикаты. Для устранения этих недостатков в разное время разрабатывались похожие методики:

**метод Хансена**. Мера сложности программы в данном случае представляется в виде пары (цикломатическая сложность, число операторов). Преимуществом данной меры является ее чувствительность к структурированности ПО.

**Метрика Пивоварского.** Модификация меры цикломатической сложности. Позволяет отслеживать различия не только между последовательными и вложенными управляющими конструкциями, но и между структурированными и неструктурированными программами.

**Мера Вудворда** — количество пересечений дуг управляющего графа. Так как в хорошо структурированной программе таких ситуаций возникать не должно, то данная метрика применяется в основном в слабо структурированных языках (ассемблер)

**Метрики сложности потока управления данными**

**Метрика Чепина**

Метрика Чепина: суть метода состоит в оценке информационной прочности отдельно взятого программного модуля с помощью анализа характера использования переменных из списка ввода-вывода. Все множество переменных, составляющих список ввода-вывода, разбивается на 4 функциональные группы :

1. «P» — вводимые переменные для расчетов и для обеспечения вывода,

2. «M» — модифицируемые, или создаваемые внутри программы переменные,

3. «C» — переменные, участвующие в управлении работой программного модуля (управляющие),

4. «T» — не используемые в программе («паразитные») переменные.

Поскольку каждая переменная может выполнять одновременно несколько функций, необходимо учитывать ее в каждой соответствующей функциональной группе. Q = a1\*P + a2\*M + a3\*C + a4\*T a1, a2, a3, a4 - коэффициенты. Автор методики предлагает использовать в виде Q = P + 2M + 3C + 0.5T

**Метрики объектно-ориентированных программных систем**

При конструировании объектно-ориентированных программных систем значительная часть затрат приходится на создание модели системы. Важно иметь возможность оценки качества этих моделей, используя числовую оценку. Решение данной задачи требует введения специального метрического аппарата. Этот аппарат развивает методики оценивания сложных программных систем, основанные на метриках сложности, связности и сцепления, но учитывает специфические особенности объектно-ориентированных решений.

Объектно-ориентированные метрики вводятся с целью:

• улучшить понимание качества продукта;

• оценить эффективность процесса конструирования;

• улучшить качество работы на этапе проектирования.

**Набор метрик Чидамбера и Кемерера**

**Метрика 1: Взвешенные методы на класс WMC (Weighted Methods Per Class)** учитывает комплексный показатель на основе количества методов в классе и их сложности

**Метрика 2: Высота дерева наследования DIT (Depth of Inheritance Tree)** максимальная длинна цепочки наследования классов. Рекомендация не более 7

**Метрика 3: Количество детей NOC (Number of children)** количество непосредственных потомков одного класса. Рекомендация не более 7 (9)

**Метрика 4: Сцепление между классами объектов СВО (Coupling between object classes):** Количество классов, о которых «знает» данный класс

**Метрика 5: Отклик для класса RFC (Response For a Class):** количество методов класса плюс количество методов других классов, вызываемых из данного класса. Метрика RFC является мерой потенциального взаимодействия данного класса с другими классами, позволяет судить о динамике поведения

**Метрика 6: Недостаток связности в методах LСOM (Lack of Cohesion in Methods)** показывает, насколько методы не связаны друг с другом через свойства (переменные). Можно определить как количество пар методов, не связанных по свойствам класса, минус количество пар методов, имеющих такую связь.

**Метрики Лоренца и Кидда**

**Метрики, ориентированные на классы:**

**Метрика 1: Размер класса CS (Class Size) –** число операций (методов) класса. CS <=20

**Метрика 2: Количество операций, переопределяемых подклассом, NOO (Number of Operations Overridden by a Subclass) –** число унаследованных операций, переопределяемых классом. NOO <=3

**Метрика 3: Количество операций, добавленных подклассом, NOA (Number of Operations Added by a Subclass) –** число новых методов, вводимых подклассом. NOA <= 4 для листового класса

**Метрика 4: Индекс специализации SI (Specialization Index) –** насколько методы класса НЕ используют совместно атрибуты

**Операционно-ориентированные метрики:**

**Метрика 5: Средний размер операции OSAVG (Average Operation Size)** – количество сообщений, посылаемых операцией. <= 25

**Метрика 6: Сложность операции ОС (Operation Complexity) --** число унаследованных операций, переопределяемых классом. NOO <=3

**Метрика 7: Среднее количество параметров на операцию NPAVG (Average Number of Parameters per operation)**

**Проектные метрики:**

**Метрика 8: Количество описаний сценариев NSS (Number of Scenario Scripts)**

**Метрика 9: Количество ключевых классов NKC (Number of Key Classes)**

**Метрика 10: Количество подсистем NSUB (NumberofSUBsystem)**

## 

пасть порву моргалы выколю за такое количество букв в одном вопросе

**№ 19**

**1 вопрос (20 баллов): Представить планирование на 3 спринта с 15 пользовательскими историями. На историях должны быть названия, оценка в стори-пойнтах согласно ряду Фибоначчи. У спринтов должны быть емкость и скорость. Расчет емкости должен быть обоснован. Тематика разработки - портал ИТ-поддержки.**

**2 вопрос (20 баллов): Опишите основные дополнения, которые привносит**

**Safe по сравнению с классическим Agile.**

1. Принципы и стратегия:

Scaled Agile Framework, также известный как SAFe — гибкий фреймворк для разработки программного обеспечения, позволяющий использовать аgile-методологии в больших командах размером более 50 человек. Agile — это итеративный метод, используемый для разработки продукта, ориентированный на непрерывное выполнение поставленных задач. Это зависит от межфункциональных команд, спринтов (наборов историй) и конкретных ролей, которые помогают в выполнении задания. Agile в основном разрабатывается для небольшой команды из десяти или менее человек. SAFe, с другой стороны, представляет собой гибкую структуру для предприятия, которая не ограничивается небольшими командами и помогает предприятиям масштабировать бережливые и гибкие методы.

2. Полученные выгоды:

Agile предоставляет проектным группам, спонсорам, лидерам и клиентам множество преимуществ для конкретных проектов, в том числе:

· Сокращение отходов за счет минимизации ресурсов

· Более быстрое время обработки

· Оптимизированные процессы разработки

· Повышенное внимание к конкретным потребностям клиентов

· Увеличение частоты сотрудничества и обратной связи

Принимая во внимание, что многочисленные бизнес-преимущества, предоставляемые SAFe, заключаются в следующем:

· Более быстрая кодификация гибких практик

· Предоставляет практическую основу для практиков, поскольку они могут напрямую писать процессы, не дожидаясь учебных занятий по разработке процессов.

· Приводит к сокращению дефектов примерно на 50 % и повышению производительности на 35 %.

· Время выхода на рынок значительно увеличивается (примерно на 50%).

3. Основные ценности:

Портфолио SAFe основано на четырех основных принципах :

· согласование

· встроенное качество

· прозрачность

· выполнение программы.

Точно так же манифест Agile также основан на следующих четырех ценностях:

· Люди и взаимодействия важнее процессов и инструментов

· Работающее программное обеспечение над исчерпывающей документацией

· Сотрудничество с клиентами в ходе переговоров по контракту

· Реагирование на изменение вместо следования плану

Вывод

SAFe — это методология, использующая гибкую структуру в соответствии со спецификацией. В случае крупных организаций, которые имеют дело с крупномасштабными и интенсивными проектами, SAFe будет более выгодным, поскольку он использует комбинацию существующих принципов бережливого производства и гибкости и объединяет их в шаблонной структуре. Кроме того, SAFe позволяет людям использовать гибкие фреймворки с большими группами, добавляя дополнительный уровень коммуникации и контроля.

Однако существуют различные другие факторы, такие как участие руководства в проекте, структура компании, количество работающих сотрудников, требования заинтересованных сторон и т. д., которые будут влиять на решение организации о выборе наиболее подходящей гибкой структуры.

**3 вопрос (20 баллов): Документирование программных систем. Частично**

**формальные и формальные нотации. Плюсы и минусы, области применения.**

**Частично формальные нотации**

• Нотация стандартизована, есть наборы графических примитивов и правила их использования

• Полная семантика и возможные трактовки элементов диаграмм не зафиксированы

• Анализ семантики упрощен по сравнению с неформальными нотациями

• Примеры: UML, BPMN, ERD

• Требуется изучение нотации всеми участниками процесса

• Как правило, требуется дополнительное ПО

• Фокусировка на определенных областях, атрибутах

**Формальные нотации**

• Семантика полностью стандартизована, смысл диаграмм определяется однозначно

• Возможен формальный анализ как синтаксиса так и семантики модели

• Специализированное по с возможностями анализа полноты описания и генерации программного кода на целевых платформах

• Примеры: AADL, ER

• Избыточно сложны

• Требуются глубокие знания языка описания

• ВСЕГДА требуется дополнительное ПО

• Больше подходят для технического описания аппаратных комплексов

Выбор нотации всегда должен учитывать:

· Какие заинтересованные лица (stakeholders) будут вовлечены в работу с документацией

· Стандарты организации/проектной группы

· Более формальные нотации требуют больших затрат, но дают новые возможности (анализ и т.д.)

· Менее формальные нотации легче создавать и применять, но они дают меньше гарантий

**№ 20**

**1 вопрос (20 баллов): Представить планирование на 3 спринта с 15 пользовательскими историями. На историях должны быть названия, оценка в стори-пойнтах согласно ряду Фибоначчи. У спринтов должны быть емкость и скорость. Расчет емкости должен быть обоснован. Тематика разработки - портал ИТ-поддержки.**

**2 вопрос (20 баллов): Требования к ПО. Опишите основные типы сбора**

**требований и приведите примеры конкретных видов.**

Задача формирования требований - сбор целей и требований из различных источников.

В соответствии с моделью Кано, в зависимости от степени влияния на удовлетворение, требования можно разделить на **три категории:**

• **базовые (ожидаемые);**

• **основные (желаемые);**

• **восхищающие (привлекательные).**

Различные **техники** сбора требования:

• методы исследования (например, интервью, анкетирование);

• методы креативности (например, мозговой штурм, парадоксальный мозговой штурм, изменение перспективы, метод аналогии);

• документоориентированные методы (например, программная археология, «автозапись», повторно используемые требования, изучение существующей документации);

• методы наблюдения (например, полевые наблюдения, обучение);

• методы поддержки (например, ментальные карты, мастер-классы, карта "Класс-Ответственность-Кооперация", аудио- и видеозаписи, диаграммы вариантов использования, прототипы).

### **Анкетирование**

Данный способ подразумевает под собой составление листа-опросника (анкеты, брифа), который может содержать открытые (требуют от опрашиваемого сформулировать его ответ) и закрытые (требуют от опрашиваемого выбрать ответ из предложенных вариантов) вопросы.

Анкетирование используется для того, чтобы подтвердить или детализировать ранее известные требования, выбрать параметры для решений.

Самым известным примером анкетирования может быть “Бриф на разработку сайта” — анкета содержащая список основных требований и информацию о будущем сайте.

**Преимущества**:

1. Высокая скорость получения результатов.
2. Сравнительно небольшие материальные затраты.

**Недостатки**:

1. Данный метод не подходит для выявления неявных требований.
2. При составлении опросника физически невозможно учесть все необходимые вопросы.

### **Интервью**

Этот метод известен многим, своего рода беседа “по душам” с заинтересованным лицом, тет-а-тет.

Необходимо задавать открытые вопросы для получения информации и закрытые для того, чтобы подтвердить или опровергнуть конкретные варианты требований.

Данный способ применяется, в основном, для получения информации по какой-либо конкретной теме и/или для уточнения требований.

Многим может показаться этот способ достаточно легким, но это не так. Провести хорошее интервью достаточно сложно. Вы должны гибко реагировать на реакцию интервьюируемого и в случае необходимости изменять порядок заготовленных вопросов или их формулировку. Не забудьте включить диктофон во время интервью или вести заметки.

**Из плюсов:**

1. Возможность задавать вопросы в произвольной последовательности.
2. Возможность использовать вспомогательный материал.
3. Анализ невербальной реакции опрашиваемого человека, позволит сделать дополнительный вывод о достоверности его ответов.

**Из минусов:**

1. Интервью отнимает достаточно много времени и сил.
2. Дополнительной сложностью является получение одинаковых ответов от интервьюируемых.

**Автозапись**

Данный метод подразумевает под собой работу с записями, письмами (электронными письмами), а также с любым другим документом, автором которого является Заказчик или конечный пользователь (т.е. стейкхолдер).

В действительности это может быть и документ и наговоренная на диктофон последовательность действий или самая обычная салфетка, на которой заказчик набросал свои идеи / проблемы /хотелки, которые необходимо превратить в полноценные требования, согласовать и передать в разработку.

Примером такого метода может быть работа с концепцией или видением проекта (сам Заказчик любит называть это — «ТЗ»), которую он прислал вам на момент начала работ по аналитике.

**Преимущество:**

* Помогает лучше понять сложные процедуры или процессы.

**Недостаток:**

* Данный метод сильно зависит от опыта Заказчика, а также от его умения формулировать и выражать свои мысли.

### **Изучение существующей документации**

Данная методика может быть использована при наличии в организации документации, которая может помочь в определении потребностей Заказчика. Примеры документации включают в себя: регламенты,

описания процессов, структура организации, спецификации продукта, различные процедуры, стандарты и инструкции, шаблоны документов и т.д.

Выявленные требования являются основой для дальнейшего анализа и должны быть детализированы.

Данная методика применима, например, при автоматизации устоявшихся в организации регламентированных бизнес процессов.

**Плюс:**

* Быстрое получение информации.

**Минус:**

* Данный способ не применим при наличии в компании только базовых документов (или при их полном отсутствии) или, если в компании Заказчика не поддерживается актуальность документации.

### **Повторное использование спецификации**

Повторно использовать спецификации можно в том случае, если есть уже завершенные один или несколько подобных проектов.

Техническое задание, подготовленное на предыдущем проекте может быть использовано для другого проекта с целью сократить продолжительность сбора, анализа и разработки требований, что позволит быстрее начать разработку.

Например, ТЗ для интернет магазинов похожи друг на друга и содержат одинаковые требования.

В большинстве случаев только часть документации актуальна для нового проекта, поэтому потребуется тщательная проверка требований на соответствие текущим целям и задачам Заказчика.

**Преимущество:**

* Сокращение времени на разработку документации.

**Недостатки:**

* Высокая стоимость первого проекта.
* Излишняя детализация требований, может привести к их дорогостоящим изменениям в будущем.

### **Представитель заказчика в компании разработчика**

Один из наиболее эффективных методов сбора требований, поскольку позволяет получать от представителя Заказчика своевременную оценку прогресса, корректности реализации, в короткие сроки получать обратную связь (фидбек) и дополнительную информацию для корректировки и разработки требований.

Метод часто применяется для сбора и управления требованиями при итерационной разработке, позволяет оперативно собирать, согласовывать и дорабатывать требования.

В дополнение можно сказать, что наличие представителя заказчика в компании разработчика является одним из главных правил Agile.

**Преимущество:**

* Быстрое получение обратной связи и информации от Заказчика.

**Недостатки:**

* Достаточно высокая цена для Заказчика.
* Затраты по времени на адаптацию сотрудника.

**Работа «в поле»**

Работа «в поле» состоит из наблюдения за деятельностью и процессами будущих пользователей системы, и в определении требований, основанных на этом наблюдении. Если говорить проще, то это наблюдение за тем, как работают пользователи, и документирование процесса, задач и результатов их деятельности.

Метод позволяет избежать проблем, связанных с трудностями стейкхолдеров в описании и выражении своих потребностей. В некоторых случаях процесс наблюдения может сопровождаться «интервьюированием» пользователей для уточнения особенностей и деталей их работы и задач. В процессе наблюдения можно так же выявить пути оптимизации бизнес процессов Заказчика.

**Преимущества:**

1. Позволяет наглядно увидеть проблему и разработать наиболее оптимальный вариант ее решения.
2. Помогает наиболее точно собрать требования, наблюдая за работой сотрудников.

**Недостатки:**

1. В процессе наблюдения могут быть упущены некоторые альтернативные сценарии бизнес процесса.
2. Трудно применим на секретных предприятиях или опасных (вредных) производствах.

### **Обучение**

Обучение — это процесс, в котором Заказчик или любой другой человек из организации Заказчика, знающий процесс, обучает аналитика по принципу учитель — ученик.

Метод полезен, когда процессы и деятельность сотрудников Заказчика трудно описать с помощью других методов или Заказчик не может предоставить адекватное описание требований.

Обучение позволяет лучше понять сложные бизнес-процессы, а также преодолеть трудности, связанные с нехваткой абстрактного мышления и самовыражения у будущих пользователей системы.

**Преимущество:**

* Позволяет понять сложный бизнес процесс, что позволяет предложить наилучшее решение.

**Недостатки:**

* Высокая стоимость и длительность.
* Метод неприменим на опасных (вредных) производствах.

**Мозговой штурм**

Мозговой штурм — наиболее часто используемый метод получения требований, которые связанны с новыми или плохо изученными направлениями деятельности организации Заказчика или функциями системы.

Он позволяет собрать множество идей от различных заинтересованных лиц (стейкхолдеров) в кратчайшие сроки и практически бесплатно.

Во время мозгового штурма участники «накидывают» любые идеи, касающиеся решения данной проблемы.

С помощью этой методики можно проработать несколько различных вариантов решения заданной проблемы, а также разрешить конфликты требований.

**Плюсы:**

* Позволяет генерировать множество (в том числе и нестандартных) вариантов решений, а также позволяет участникам развивать идеи друг друга.

**Минусы:**

* Участники мозгового штурма должны быть мотивированы на идеи.
* Трудно применим в распределенных командах.

### **Совещание**

Совещание — встреча, ориентированная на обсуждение конкретных вопросов, которые были определены и озвучены участникам заранее.

На такие встречи привлекаются люди, которые придерживаются различных точек зрения по текущей проблеме и могут помочь описать требования, основываясь на взглядах с разных сторон. В процессе совещания уточняется общий список требований, выявляются скрытые требования и решаются конфликты требований.

Совещания являются одной из ключевых практик в Agile, т.к. в них участвуют все стороны, заинтересованные в развитии проекта и решении проблемы.

**Плюсы:**

* Позволяет развить и детализировать требования, определить приоритеты.

**Недостатки:**

* Сложности в организации встречи, если команда географически разделена, могут возникнуть трудности с присутствием всех необходимых людей на совещании.
* Консенсус необязательно будет достигнут.

### **Use case**

Use cases или варианты использования позволяют собрать и сформировать функциональные требования от лица участников [Диаграммы вариантов использования](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B0_%D0%BF%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82%D0%BE%D0%B2) определяют границы решения и показывают связи с внешними системами и участниками.

Метод позволяет детализировать требования с точки зрения пользователей, а также помогает уточнить и систематизировать функционал, который требуется реализовать.

**Плюсы:**

* Позволяет проработать все варианты развития сценария (основной и альтернативные сценарии)

**Минусы:**

* Метод не применим для сбора нефункциональных требований.

**3 вопрос (20 баллов): Паттерны проектирования. Структурные паттерны (на**

**выбранном примере).**

**Паттерн проектирования** — это часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы.

Паттерны часто путают с алгоритмами, ведь оба понятия описывают типовые решения каких-то известных проблем. Но если алгоритм — это чёткий набор действий, то паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм — это кулинарный рецепт с чёткими шагами, а паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

**Адаптер** — это структурный паттерн проектирования, который позволяет объектам с несовместимыми интерфейсами работать вместе.

**Проблема**

Представьте, что вы делаете приложение для торговли на бирже. Ваше приложение скачивает биржевые котировки из нескольких источников в XML, а затем рисует красивые графики.

В какой-то момент вы решаете улучшить приложение, применив стороннюю библиотеку аналитики. Но вот беда — библиотека поддерживает только формат данных JSON, несовместимый с вашим приложением.

Вы смогли бы переписать библиотеку, чтобы та поддерживала формат XML. Но, во-первых, это может нарушить работу существующего кода, который уже зависит от библиотеки. А во-вторых, у вас может просто не быть доступа к её исходному коду.

**Решение**

Вы можете создать *адаптер*. Это объект-переводчик, который трансформирует интерфейс или данные одного объекта в такой вид, чтобы он стал понятен другому объекту.

При этом адаптер оборачивает один из объектов, так что другой объект даже не знает о наличии первого. Например, вы можете обернуть объект, работающий в метрах, адаптером, который бы конвертировал данные в футы.

Адаптеры могут не только переводить данные из одного формата в другой, но и помогать объектам с разными интерфейсами работать сообща. Это работает так:

1. Адаптер имеет интерфейс, который совместим с одним из объектов.
2. Поэтому этот объект может свободно вызывать методы адаптера.
3. Адаптер получает эти вызовы и перенаправляет их второму объекту, но уже в том формате и последовательности, которые понятны второму объекту.

Иногда возможно создать даже *двухсторонний адаптер*, который работал бы в обе стороны.

Таким образом, в приложении биржевых котировок вы могли бы создать класс XML\_To\_JSON\_Adapter, который бы оборачивал объект того или иного класса библиотеки аналитики. Ваш код посылал бы адаптеру запросы в формате XML, а адаптер сначала транслировал входящие данные в формат JSON, а затем передавал бы их методам обёрнутого объекта аналитики.

**Аналогия из жизни**

Когда вы в первый раз летите за границу, вас может ждать сюрприз при попытке зарядить ноутбук. Стандарты розеток в разных странах отличаются. Ваша европейская зарядка будет бесполезна в США без специального адаптера, позволяющего подключиться к розетке другого типа.

**№ 21**

**1 вопрос (20 баллов): Представить планирование на 3 спринта с 15 пользовательскими историями. На историях должны быть названия, оценка в стори-пойнтах согласно ряду Фибоначчи. У спринтов должны быть емкость и скорость. Расчет емкости должен быть обоснован. Тематика разработки - веб-сайт заказа авиабилетов.**

Заранее условимся, что 1 storypoint - 1 час работы 1-го разработчика

Каждый спринт длится 2 недели

**1 спринт**

Я как пользователь хочу менять местами город отправления и город прибытия, чтобы иметь возможность посмотреть билет туда и обратно по отдельности (3)

Я как пользователь хочу открыть Q&A на веб-сайте, чтобы иметь возможность быстро найти ответ на свой вопрос (21)

Я как пользователь хочу просмотреть популярные направления, чтобы иметь возможность спланировать путешествие (5)

Я как пользователь хочу выбрать веганское питание, чтобы иметь возможность пообедать в течение длинного перелета (8)

Я как пользователь хочу купить билет туда и обратно, чтобы иметь возможность купить билеты быстро (13)

Всего: 3+5+8+13+21 = 50 часов

**2 спринт**

Я как пользователь хочу выбрать отель на странице заказа авиабилетов, чтобы купить все в одном месте (21)

Я как пользователь хочу почитать статьи о путешествиях, чтобы иметь возможность принять во внимание советы по перевозке зимнего инвентаря (13)

Я как пользователь хочу выбрать класс обслуживания, чтобы иметь возможность сравнить цены и выбрать подходящий тариф (3)

Я как пользователь хочу зарегистрироваться онлайн на рейс, чтобы иметь возможность не стоять в очереди на регистрацию в аэропорту (8)

Я как пользователь хочу докупить багаж, чтобы иметь возможность привести тяжелые предметы (5)

Всего: 3+5+8+13+21 = 50 часов

**3 спринт**

Я как пользователь хочу выбрать количество пассажиров, чтобы иметь возможность купить билеты для всей семьи (3)

Я как пользователь хочу указать на календаре дату вылета, чтобы иметь возможность узнать день недели, на который куплен билет (8)

Я как пользователь хочу составить сложный маршрут, чтобы иметь возможность сразу купить билет в несколько городов и стран (13)

Я как пользователь хочу войти в личный кабинет, чтобы иметь возможность смотреть историю своих заказов (21)

Я как пользователь хочу изменить страницу веб-сайта на темную тему, чтобы иметь возможность сохранить заряд батареи надолго (5)

Всего: 3+5+8+13+21 = 50 часов

**Нам необходимо 50+50+50 = 150 рабочих часов => что необходимо привлечь двух разработчиков, которые работают по 40 часов в неделю, а так как спринт длится 2 недели, то этого времени должно хватить, так как у нас будет 40\*2\*2 = 160 часов**

**2 вопрос (20 баллов): Назовите основные мероприятия Scrum и опишите**

**цель проведения каждого.**

1. **Организация бэклога**. За это мероприятие, также известное как ведение бэклога, несет ответственность владелец продукта. В число его основных обязанностей входят приведение продукта в соответствие с его концепцией и постоянное отслеживание настроений на рынке и потребностей клиента. Для этого владелец продукта и ведет список, изменяя в нем приоритеты и поддерживая его в актуальном виде на основании информации от пользователей и команды разработчиков, чтобы в любое время можно было приступить к работе над внесенными в него задачами.
2. **Планирование спринта**. На этом собрании команда разработчиков под руководством Scrum-мастера планирует работу (объем спринта), которую необходимо выполнить в течение текущего [спринта](https://confluence.atlassian.com/agile/glossary/sprint). На нем выбирается цель спринта. Затем в спринт добавляются конкретные пользовательские истории из бэклога продукта. Эти истории всегда соотносятся с целью. При этом команда Scrum согласовывает такие истории, которые можно будет реализовать на практике в ходе спринта. В конце собрания по планированию каждый член команды Scrum должен четко представлять, какие задачи можно выполнить за спринт и как поставить инкремент.
3. **Спринт**. Спринт — это фактический промежуток времени, в течение которого команда Scrum совместно работает над созданием готового инкремента. Как правило, спринт длится две недели, хотя некоторым командам проще спланировать объем спринта на одну неделю или поставить инкремент, обладающий достаточной ценностью, за месяц. Дейв Уэст из Scrum.org рекомендует планировать спринт тем короче, чем сложнее работа и чем больше в ней неизвестных. Но последнее слово всегда за командой. Не стесняйтесь менять продолжительность спринта, если покажется, что она вам не подходит. В течение этого периода владелец продукта и команда разработчиков могут пересмотреть объем спринта, если это необходимо. Это и есть ключ к пониманию эмпирической сути Scrum.  
    Все мероприятия, от планирования до ретроспективы, проводятся в течение спринта. После того как временной промежуток для спринта определен, он должен оставаться неизменным, пока ведется разработка. Так команда будет извлекать ценные уроки из прошлого опыта и применять выводы к будущим спринтам.

4. **Ежедневное Scrum-совещание, или стендап**. Это очень короткое ежедневное собрание, которое для удобства проводится в одно и то же время (обычно утром) и в одном и том же месте. Многие команды стараются уложиться в 15 минут, однако это лишь рекомендация. Такое собрание также называется «ежедневным стендапом», что подчеркивает его краткость. Ежедневное Scrum-совещание проводится, чтобы каждый участник команды был в курсе происходящего, не отклонялся от цели и получал план работы на ближайшие 24 часа.

**Стендап — подходящее время сообщить обо всем, что мешает вам достичь цели спринта, в том числе о блокерах.**

Чаще всего в рамках стендапа каждому участнику команды предлагается ответить на следующие три вопроса, связанные с достижением цели спринта:

• «Что мне удалось сделать вчера?»

• «Что я планирую сделать сегодня?»

• «Может ли мне что-то помешать?»

**5. Обзор итогов спринта**. В конце спринта команда собирается для просмотра демонстрации инкремента (или для его изучения) в неформальной обстановке. Команда разработчиков представляет заинтересованным сторонам и коллегам завершенные рабочие задачи из бэклога, чтобы собрать отзывы. Владелец продукта решает, стоит ли выпускать инкремент, хотя в большинстве случаев команда получает зеленый свет.

На собрании по обзору итогов владелец продукта также изменяет бэклог продукта на основании результатов последнего завершенного спринта. Этот процесс может перейти в планирование следующего спринта. Если спринт длится один месяц, отводите под собрание для обзора итогов не более четырех часов.

**6. Ретроспектива спринта**. [Ретроспектива](https://www.atlassian.com/ru/agile/scrum/retrospectives) проводится, чтобы команда зафиксировала и обсудила все успехи и неудачи спринта, проекта, участников и их взаимоотношений, инструментов или даже определенных собраний. Цель ретроспективы — создать условия, чтобы команда могла уделить внимание всему, что удалось и что нужно улучшить в следующий раз, и не зацикливалась на неудачах.

**3 вопрос (20 баллов): Документирование программных средств. Нотации.**

**Рассмотреть пример одной из нотаций на выбор. Основные типы диаграмм,**

**области применения.**

**Неформальные нотации**

• Диаграммы в свободной форме

• Семантика моделей на естественном языке (цвета, форма, надписи)

• Информация диаграмм не может быть формально проанализирована

• Легко понятные, но при этом высок риск неправильного понимания

• Пример: классические boxes-and-lines, например в PowerPoint

• Для понимания требуется «ключ»

• Содержимое раскрывается с помощью фигур, цветов и стрелок

• Легко создавать

• Самые популярные

• Хороши для презентаций

**Частично формальные нотации**

• Нотация стандартизована, есть наборы графических примитивов и правила их использования

• Полная семантика и возможные трактовки элементов диаграмм не зафиксированы

• Анализ семантики упрощен по сравнению с неформальными нотациями

• Примеры: UML, BPMN, ERD

• Требуется изучение нотации всеми участниками процесса

• Как правило, требуется дополнительное ПО

• Фокусировка на определенных областях, атрибутах

**Формальные нотации**

• Семантика полностью стандартизована, смысл диаграмм определяется однозначно

• Возможен формальный анализ как синтаксиса так и семантики модели

• Специализированное по с возможностями анализа полноты описания и генерации программного кода на целевых платформах

• Примеры: AADL, ER

• Избыточно сложны

• Требуются глубокие знания языка описания

• ВСЕГДА требуется дополнительное ПО

• Больше подходят для технического описания аппаратных комплексов

Выбор нотации всегда должен учитывать:

· Какие заинтересованные лица (stakeholders) будут вовлечены в работу с документацией

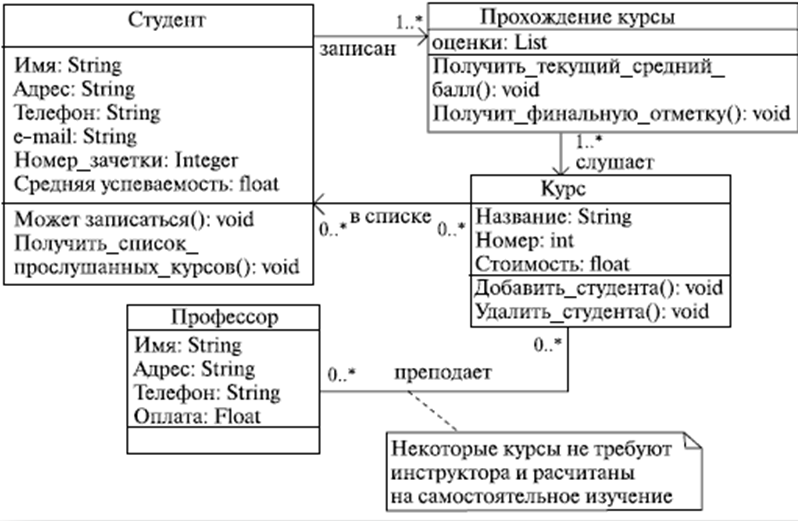
· Стандарты организации/проектной группы

· Более формальные нотации требуют больших затрат, но дают новые возможности (анализ и т.д.)

· Менее формальные нотации легче создавать и применять, но они дают меньше гарантий

Рассмотрим простой **пример нотации UML** (диаграмма классов)

Класс (class) — категория вещей, которые имеют общие атрибуты и операции. Сама диаграмма классов являет собой набор статических, декларативных элементов модели. Она дает нам наиболее полное и развернутое представление о связях в программном коде, функциональности и информации об отдельных классах. Приложения генерируются зачастую именно с диаграммы классов.



Для класса "студент" есть таблица, содержащая атрибуты: имя, адрес, телефон, e-mail, номер зачетки, средняя успеваемость. И также показаны связи данной сущности с другими: прохождением курса, какой курс слушает, кто профессор. В этом примере добавляются функции, которые могут быть применены к сущности "студент".

**№ 22**

**1 вопрос (20 баллов): Написать не менее 10 различных дефектов на страницу авторизации и страницу оплаты к веб-сайту заказа билетов в кино. Описание должно быть достаточным для его воспроизведения и ожидаемым результатом. Должны быть ссылки на предполагаемые сценарии тестирования.**

**2 вопрос (20 баллов): Процесс управления релизами. Выделите основные**

**понятия и опишите процесс**

**Что такое управление релизами**

Релизами называют запланированные изменения, которые вносят в программное или аппаратное обеспечение. вероника лохушка

Управление релизами по ITIL − это комплексный процесс, объединяющий планирование, реализацию, тестирование и внедрение изменений в ИТ-продукт, а также контроль их качества при дальнейшем использовании.

**Зачем автоматизировать процесс управления релизами**

Внедрение процесса управления релизами позволяет:

1. Вносить изменения в ИТ-среду без ухудшения качества обслуживания.
2. Уменьшить число инцидентов, вызванных несовместимостью новых систем с установленным аппаратным и программным обеспечением.
3. За счет тщательного тестирования новых ИТ-решений выявлять и предотвращать потенциальные вопросы и проблемы у пользователей.
4. Снизить количество неконтролируемых версий программного обеспечения в ИТ-среде и тем самым снизить риски, связанные с использованием нелицензионного ПО.
5. Предотвратить утрату исходных файлов программного обеспечения.

**Из каких этапов состоит процесс управления релизами**

**1. Планирование**

На этом этапе собираются требования к продукту от клиентов и внутренних заказчиков, анализируется актуальность предложенных изменений с учетом ресурсов, компетенций, времени на реализацию и совместимости с уже существующими функциями ПО.

Приоритетные запросы группируются и разделяются на блоки, задачи и подзадачи. Составляется подробный план релиза со сроками и контрольными точками. Указываются специалисты, которых необходимо привлечь к участию в проекте.

**2. Сборка релиза**

Организуется процесс разработки и стартует реализация намеченных функциональных изменений. Объемный по содержанию релиз разбивается на спринты и итерации.

**3. Тестирование**

Проводится тестирование доработок в собственной ИТ-среде. При необходимости формируются группы пользователей, которым предоставляется доступ к бета-версии для выявления критичных багов и первичной «обкатки» новых возможностей. По итогам тестирований производится финальная подготовка релиза с исправлением всех ошибок.

**4. Развертывание**

Осуществляется выпуск изменений, передача в эксплуатацию, документирование изменений. Выходит новая версия или сам конечный продукт. Все пользователи получают доступ к функциям, которые разрабатывались в релизе.

**5. Контроль качества**

После развертывания участвующая в релиз менеджменте команда подводит итоги, анализирует выполненное. Если при внедрении новых функций возникает потребность в сопутствующих доработках, специалисты фиксируют их и планируют дальнейшее усовершенствование продукта в рамках нового релиза.

**3 вопрос (20 баллов): Документирование программных систем. Частично формальные и формальные нотации. Плюсы и минусы, области применения.**

**Частично формальные нотации**

• Нотация стандартизована, есть наборы графических примитивов и правила их использования

• Полная семантика и возможные трактовки элементов диаграмм не зафиксированы

• Анализ семантики упрощен по сравнению с неформальными нотациями

• Примеры: UML, BPMN, ERD

• Требуется изучение нотации всеми участниками процесса

• Как правило, требуется дополнительное ПО

• Фокусировка на определенных областях, атрибутах

**Формальные нотации**

• Семантика полностью стандартизована, смысл диаграмм определяется однозначно

• Возможен формальный анализ как синтаксиса так и семантики модели

• Специализированное по с возможностями анализа полноты описания и генерации программного кода на целевых платформах

• Примеры: AADL, ER

• Избыточно сложны

• Требуются глубокие знания языка описания

• ВСЕГДА требуется дополнительное ПО

• Больше подходят для технического описания аппаратных комплексов

Выбор нотации всегда должен учитывать:

· Какие заинтересованные лица (stakeholders) будут вовлечены в работу с документацией

· Стандарты организации/проектной группы

· Более формальные нотации требуют больших затрат, но дают новые возможности (анализ и т.д.)

Менее формальные нотации легче создавать и применять, но они дают меньше гарантий

**№ 23**

**такой же как 22\***

**№ 24**

**1 вопрос (20 баллов): Предложить вариант высокоуровневой архитектурной схемы для портала IT поддержки. Основные требования к системе – централизованное управление, возможность работы с заявками через web-интерфейс и с мобильных устройств, система авторизации и аутентификации, система мониторинга и система обработки статистики.**

**Описать вариант реализации компонент и их развертывания, высокоуровневые потоки данных.**

Примитивная архитектура такая:

Нам нужен веб-сервер, который будет общаться с веб-интерфейсом

Через веб-интерфейс эти заявки будут приниматься, отправляться фронтом серверу

Сервер уже будет сохранять в бд

Например

Бд бывают двух видов OLTP и OLAP, запросы будут лететь в OLTP, а затем реплицироваться (дублироваться) в OLAP. OLAP базы нужны для удобно построения статистики

Эти инструменты показывают различную статистику по системе типа CPU load, Men usage и подобные метрики с машин

Внутри сервера надо выделить компоненты, которые отвечают за аутентификацию и авторизацию

Или сделать отдельный микросервис отвечающий за это (это удобно когда несколько сервисов привязаны к одному пользователю, типа Я музыка и Я такси связаны через Я паспорт)

Так называемый SSO (Single Sign On - единая точка входа)

Для хранения сессии пользователя лучше всего использовать jwt токен, который будет храниться и в мобильном приложении, и в браузере в local storage

Если авторизация внутри основного сервера будет, то можно добавить middleware (штука которая перехватывает запросы, проверяет jwt, берет из jwt инфу о пользователе, и добавляет в запрос)

Если авторизация отдельным микросервисом, то сервис может принимать запрос, проверять jwt, добавлять в заголовок инфу о пользователе и перенаправлять запрос к основному серверу

Этакий прокси сервер, который jwt превращает в инфу о пользователе

Можно добавить SMTP сервер, который будет принимать письма и по шине данных (типа NATS)/брокер сообщений (типа rabbit MQ) отправлять письма основному серверу, который будет их парсить и складывать в базу

Система мониторинга:

Для этого есть различные решения, например graphite + graphana (можно только графану загуглить, графит найдётся). Также графане небоходим источник данных

Основной сервер можно развернуть не в одном экземпляре, а в нескольких, тогда возможно ещё нужен будет load balancer для распределения нагрузки между экземплярами сервера, чтоб не было такого, что нагружают только один экземпляр и он умирает. Пример load balancer’a - traefik

К вопросу о деплое. Отличный инструмент для разворачивания нескольких экземпляров - kubernetes, там приписывается какие компоненты надо задеплоить с какими конфигами, сколько экземпляров и т.д. и т.п.

если брать kubernetes, то docker в качестве средства контейнеризации

**2 вопрос (20 баллов): Процессы тестирования ПО. Опишите основные**

**способы деления видов тестирования.**

Процессы тестирования:

Всего принято выделять 7 этапов тестирования:

1. **Работа с требованиями**. Знакомство с требованиями заказчика, что должен из себя представлять итоговый продукт, обсуждение.
2. **Разработка стратегии тестирования**. Оценка сроков тестирования, выявление среды тестирования, объединение всей информации, полученной при работе с требованиями.
3. **Создание тестовой документации.** Написание сценариев, которые позволят проверить функционал.
4. **Тестирование прототипа.** Тестирование основного функционала продукта, корректировка целей, добавление фичей.
5. **Основное тестирование.** Выполнение общей проверки продукта.
6. **Стабилизация.** На данном этапе происходит работа над устранением багов.
7. **Эксплуатация.** Проводится регресс-тестирование, устранение ошибок, которые нашел конечный пользователь.

Деление видов тестироваия:

1. по цели (функциональные, нефункциональные)
2. по степени автоматизации (ручное, автоматизированное)
3. по позитивности сценария (позитивная проверка, негативная)
4. по доступу к коду (белый ящик, серый ящик, черный ящик)
5. по уровню (модульное, интеграционное, системное, приемочное)
6. по исполнителю (альфа/бета - тестирование)
7. по формальности (тестирование по тестам, исследовательское, свободное)
8. по важности (smoke, тестирование критического пути, расширенное)

**3 вопрос (20 баллов): Мониторинг систем. Основные решаемые задачи, принципы организации, типы комплексных систем мониторинга**

Мониторинг ИТ систем является составной частью управления информационной инфраструктурой предприятия, заключающейся в постоянном наблюдении и периодическом анализе ИТ объектов с отслеживанием динамики происходящих с ними изменений. Ключевой задачей систем мониторинга ИТ является получение, сохранение и анализ информации о состоянии подконтрольных элементов ИТ

структуры компании

Как правило, выделяется два уровня мониторинга:

• Мониторинг инфраструктуры

• Мониторинг сервисов

**Внедрение комплексной системы мониторинга позволяет:**

• своевременно фиксировать возникновение проблем в работе компонентов ИТ структуры;

• обнаруживать место и характер неполадки;

• определять влияние возникшей проблемы на возможность предоставления ИТ сервисов (это необходимо для верной расстановки приоритетов в работах по ликвидации сбоев;

• проактивно следить за изменениями в функционировании инфраструктуры;

• предотвращать вероятные сбои.

**Подходы к созданию комплексной системы мониторинга**

В практике, выделяют следующие основные подходы:

♦ подход от инфраструктуры (т.е. «снизу вверх»)

♦ подход от ИТ сервисов (т.е. «сверху вниз»)

**Организация системы мониторинга от ИТ сервисов**

Применение данного подхода заключается в формировании каталога услуг и отвечает методологии сервисного подхода к управлению ИТ (ITSM). Предполагается, что для каждого сервиса (услуги) должна быть разработана своя сервисно-ресурсная модель, отражающая взаимодействие между сервисом и другими компонентами инфраструктуры, нужными для его работы. С использованием сервисно-ресурсной модели проводится процедура настройки программы мониторинга с целью контроля функционирования ИТ сервиса и всех связанных с ним компонентов инфраструктуры. Этот подход способствует тому, что системные консоли становятся полезными не только отвечающим за поддержку определенных сервисов ИТ-специалистов, но и диспетчерской службе, а также руководству ИТ-отдела

**Что входит в типовую систему мониторинга?**

1 Мониторинг сетей

2 Мониторинг серверов и рабочих станций

3 Мониторинг приложений и сервисов

* активный мониторинг программ и сетевых сервисов
* мониторинг сбоев в программах
* мониторинг производительности приложений
* мониторинг сервисов

4 Предоставление данных:

* портал системы управления ресурсами;
* формирование отчетности

**Основные принципы организации системы мониторинга**

• Лучше использовать стандартные решения (Nagios, Zabbix и т.п.). Собственная

система - это долго и дорого. Лучше потратить время на изучение какой-то стандартной системы и научиться писать к ней плагины, чем разрабатывать

свою. Самописные системы тяжело развивать, особенно новым сотрудникам,

не принимавшим участие в её создании.

• Система мониторинга не имеет смысла без организации дежурных смен

(и/или мгновенных уведомлений администратору) для устранения проблем.

• Постоянная подстройка порогов реакции. Тысячи уведомлений будут

бесполезны, правило должно быть таким: пришло уведомление - значит

серьёзная проблема или авария.

• Автоматизация типовых реакций. Все что можно автоматизировать и нужно

автоматизировать, в первую очередь некритичные рутинные операции.

• Нужна распределённая система мониторинига, которая может мониторить и

саму себя. Либо две системы мониторинга: одна мониторит одну часть проекта, вторая - другую, и каждая из них мониторит другую систему мониторинга.

**Zabbix** — система мониторинга и отслеживания статусов разнообразных сервисов компьютерной сети, серверов и сетевого оборудования

Поддерживает несколько видов мониторинга:

• Simple checks — может проверять доступность и реакцию стандартных сервисов, таких как SMTP или HTTP, без установки какого-либо программного обеспечения на наблюдаемом хосте.

• Zabbix agent — может быть установлен на UNIX-подобных или Windows-хостах для получения данных о нагрузке процессора, использования сети, дисковом пространстве и так далее.

• External check — выполнение внешних программ, также поддерживается мониторинг через SNMP

Основные возможности:

• Распределённый мониторинг — до нескольких тысяч узлов.

• Сценарии на основе мониторинга

• Автоматическое обнаружение

• Централизованный мониторинг журналов

• Веб-интерфейс для администрирования и настройки

• Отчётность и тенденции

• SLA-мониторинг

• Поддержка высокопроизводительных агентов (zabbix-agent) практически для всех платформ

• Комплексная реакция на события

**№ 25**

**1 вопрос (20 баллов): Написать базовые положения соглашения об уровне обслуживания веб-сайта заказа билетов со стороны внешнего подрядчика. Выделить три уровня приоритета, критерии определения приоритета и временные характеристики.**

Соглашение об уровне обслуживания веб-сайта заказа билетов со стороны внешнего подрядчика:

Название системы

ИТ-служба

Границы оказания услуг

Услуги оказываются на территории по следующему адресу: г. Москва, ул. Бориса Галушкина д.11, а также дистанционно для всех пользователей нашей системы

Услуги оказываются с 08:00 до 18:00 МСК в рабочие дни

Список предоставляемых услуг:

Обработка сообщений от пользователей

Разбор кейсов

Мониторинг и поддержка работоспособности системы (после внедрения нашей системы)

Приоритеты

| Приоритет | Определение | Промежуток времени для решения |
| --- | --- | --- |
| Высокий | проблема, которая повлекла за собой потерю работоспособности системы. | в течение 1-3 часа |
| Средний | проблема, которая повлекла незначительную работоспособность системы, но возможно использовать альтернативные пути решения | в течение 3-9 часов |
| Низкий | проблема, которая не связана с потерей работоспособности системы. Например, ошибка в документации | в течение 9-12 часов |

**2 вопрос (20 баллов): Требования к ПО. Виды требований. Техники сбора и**

**описания.**

Требование - это:

1. Условие или возможность, требуемая пользователем для решения задач или достижения целей.
2. Условие или возможность, которыми система/компонент системы должна обладать для обеспечения условий контракта, стандартов, спецификаций или др. регулирующими документами.

Требования можно разделить на две большие группы:

1. Функциональные требования
2. Нефункциональные требования

Функциональные требования определяют, что разработчики должны создать, чтобы пользователи смогли выполнить свои задачи (пользовательские требования) в рамках бизнес-требований. Такие требования описываются в форме традиционных утверждений со словами «должен» или «должна»

Нефункциональные требования определяют стандарты производительности и атрибуты качества программного обеспечения, например удобство использования системы, эффективность, безопасность, масштабируемость и т.д.

**Техники сбора требований.**

Можно долго перечислять различные техники сбора требований. Но вот те, которые помогут это сделать эффективно:

1. **Анкетирование**

Данный способ подразумевает под собой составление листа-опросника (анкеты, брифа), который может содержать открытые (требуют от опрашиваемого сформулировать его ответ) и закрытые (требуют от опрашиваемого выбрать ответ из предложенных вариантов) вопросы.

1. **Представитель заказчика в компании разработчика**

Один из наиболее эффективных методов сбора требований, поскольку позволяет получать от представителя Заказчика своевременную оценку прогресса, корректности реализации, в короткие сроки получать обратную связь (фидбек) и дополнительную информацию для корректировки и разработки требований.

1. **Наблюдение**

Наблюдение за тем, как работают пользователи, и документирование процесса, задач и результатов их деятельности.

1. **Use cases**

Use cases или варианты использования позволяют собрать и сформировать функциональные требования от лица участников, определяют границы решения и показывают связи с внешними системами и участниками.

**3 вопрос (20 баллов): Паттерны проектирования. Порождающие паттерны**

**(на выбранном примере)**

**№ 26**

**1 вопрос (20 баллов): Написать базовые положения соглашения об уровне**

**обслуживания веб-сайта заказа билетов со стороны внешнего подрядчика.**

**Выделить три уровня приоритета, критерии определения приоритета и**

**временные характеристики.**

**2 вопрос (20 баллов): Назовите основные мероприятия Scrum и опишите**

**цель проведения каждого.**

**3 вопрос (20 баллов): Документирование программных систем.**

**Неформальные нотации. Плюсы и минусы, области применения**