Список вопросов по курсу “Управление ИТ-проектами”

Оглавление

1. Характерные черты ИТ-проекта. Отличия проектной деятельности от других видов деятельности.

Проект – это временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов не существует двух одинаковых проектов, все проекты уникальны!

- не существует идеальной системы управления проектами, подходящей для каждого из видов проектов
- не существует системы, которая бы подходила каждому руководителю и была удобна всем членам команды
- за время развития проектного управления было создано немало эффективных подходов, методик и стандартов, которые можно взять на вооружение.

Существующие подходы отличаются друг от друга - по областям применения, степени детализации и уровню формализации направлены на достижение определённых целей

- включают последовательное выполнение логически взаимосвязанных операций
- имеют определённую ограниченную протяжённость во времени, с фиксированным началом и концом
- неповторимы и уникальны

2. Управление проектом. Ограничения проекта СРОКИ СОДЕРЖАНИЕ СТОИМОСТЬ. Роль руководителя ИТ-проекта.

В управление проектом входит

- определение основных требований заказчика;
- установка определённых и достижимых целей;
- согласование противоречащих требований по времени, качеству, содержанию и стоимости;
- корректировка, в соответствии с ожиданиями участников проекта, планов и подходов к реализации.
- Управление это приложение навыков, знаний, методов и инструментов к работам проекта для удовлетворения требований заказчика, предъявляемых к проекту
- Управление проектом выполняется с помощью интеграции и применения сорока двух логически сгруппированных процессов управления проектами, объединённых в пять групп процессов. Данные пять групп начинаются от инициации и планирования далее переходят в исполнение и мониторинг, поддерживаются постоянным управлением и в итоге завершаться

Менеджер проекта (руководитель проекта) – член проектной команды, ответственный за выполнение ключевых показателей эффективности проекта - KPI (Key Performance Indicators)

Ограничения проекта:

- сроки
- содержание
- стоимость (бюджет)

- ресурсы

- качество
- риски

3. Успешность ИТ-проектов. Основные причины их провалов.

- анализ результатов проектной деятельности в области информационных технологий показал существующую тенденцию к снижению качества проектов. Причин этому много, но мы, вслед за Р.Глассом, будем считать одной из основных — неверную оценку проекта к неверной оценке приводит порочное руководство проектом, при котором руководитель не знает существующих методик оценки, не умеет ими пользоваться, применяет их не по назначению или полностью их игнорирует. В любом случае проект из-за дефицита времени попадает в тяжёлое положение и либо теряет качество, либо закрывается.

По данным за 2013 г. ,опрос PM Expert 100 отечественных компаний позволил выявить основные проблемы российских организаций: отсутствие актуальной картины проекта, задержки в ходе выполнения, отсутствие четко распределенной ответственности.

После внедрения системы управления улучшение ситуации отметили 30% респондентов, 65% указали, что ситуация значительно улучшилась. Использование стандартных методологий управления проектами показывает, что оценки российских заказчиков вполне совпадают с мировой статистикой. Данные, представленные в аналитическом исследовании компании IBM, говорят о том, что прирост стоимости по результатам управляемого проекта достигает 20% по сравнению с тем, как если бы он выполнялся без применения соответствующей методологии.

Причины провалов:

- отсутствие требований заказчика
- неполнота требований и их частое изменение
- отсутствие требуемого опыта и ресурсов
- “забытые работы” и неполнота планирования
- отсутствие взаимодействия с заказчиком
- грубые ошибки в оценке трудоёмкости, сроков и длительности работ

Успешность программного проекта

- Баланс факторов: персонал, методология, границы проекта, время разработки, качество
- Чтобы ни случилось – это к худшему
- Численность разработчиков увеличивать нельзя
- Новая методологию предполагает ее освоение
- Сокращение границ в конце не спасает
- Жертвуют либо временем, либо качеством, а чаще тем и другим

4. Основные методы оценки программных проектов. Аналитические модели.

Основные критерии метода оценки

- формирует корректные оценки;
- экономически эффективный;
- учитывает вероятностный характер оценок;
- не зависит от масштаба и методологии управления;

- учитывает экспертные оценки;
- обеспечивает достаточные время и точность оценки;
- простой в практическом применении;
- прозрачный для сотрудников, принимающих решения;
- допускает эффективную программную поддержку модели;
- учитывает лучший мировой опыт.

На заре промышленного программирования, для задач математического характера, возникла идея строгой математической оценки трудоемкости и сроков работ.

На статистике по выполненным проектам выявлялись закономерности выполнения и соотношения параметров, строилась математическая модель, для прогнозирования аналогичных параметров

Оценка стоимости затрат в таких моделях имеет вид

$$f(x1, x2, \ldots, xn),$$

где x1, x2, …, xn — стоимостные факторы, а f — функция, отличная от линейной.

Наиболее распространенная модель этой группы — модель SLIM (Л. Патнам) для крупных проектов.

Её базовое утверждение: основные затраты на разработку ПО распределяются по времени согласно функции, графики которой — кривые Патнама–Нордена–Рэлея

Уравнение модели имеет вид:

$$S = E^{1/3} \times t^{4/3} \times P \times B^{1/3},$$

где S — размер, E — объём работы, t — общее время разработки, P — производительность, B — масштабирующий коэффициент.

Размер проекта измеряется в эффективных строках кода: K × LOC, где LOC — количество строк кода, K зависит типа кода: для нового K = 1; для повторного K = 0,55; для сгенерированного K = 0,40.

Зависимость объема работы от количества строк кода:

$$E = S^3 \times P^{-3} \times B \times t^{-4},$$

Для повышения качества прогноза SLIM предлагает вести калибровку на основе хронологической информации.

Модель не приобрела широкой популярности и распространения в силу своей специфичности (большие проекты).

5. Основные методы оценки программных проектов. Экспертные методы

Экспертные методы оценки используют опыт экспертов, которые либо работали над программными проектами, либо участвовали в их оценке.

Привлекательность в простоте, и при хороших экспертах качество оценки может быть высоким.

Но риск велик: эксперт может не знать особенностей именно данного проекта; может ошибиться в масштабе; может неверно оценить возможности проектной группы; может быть предвзятым.

Поэтому стараются минимизировать фактор субъективности (несколько экспертов, плюсы и минусы).

Метод Делфи. Интуитивное обоснование метода — группа независимых экспертов лучше оценивает и предсказывает результат, чем рабочая команда проекта.

Эксперты делают независимые заключения по обсуждаемому вопросу.

Ответы обобщаются, обрабатываются и возвращаются экспертам, которые делают новые оценки.

После формирования на их основе некоторого общего мнения получается окончательная оценка.

Метод достаточно хорош, если по обсуждаемой проблеме слишком мало объективных данных.

Вариация метода — Wideband Delphi. порядок его этапов:

- Планирование: ставится задача и определяется группа;
- начальное собрание, эксперты знакомятся с задачей;
- Предварительная работа, где участники независимо составляет список задач и предлагает свои оценки;
- Собрание для обсуждения материалов и выработки списка задач и оценок;
- Объединение задач: руководитель проекта объединяет результаты и доводит их до сведения экспертов;
- Просмотр результатов: завершающий этап, оценка;
- Если критерии удовлетворены, оценка завершается.

Далее на основании полученных оценок определяется ожидаемая, которая задаётся интервалом.

Достоинства метода:

- невысокие затраты при получении экспертных оценок;
- возможность использования предыдущего опыта;
- использование математических моделей для улучшения прогностических оценок;
- возможность проводить опрос удаленно;
- исключение группового влияния, возникающего при совместной работе.

Недостатки метода:

- для получения качественных оценок эксперты должны обладать высоким уровнем мотивации;
- нередко принимается мнение большинства;
- стремление экспертов попасть в группу большинства;
- существует возможность манипуляции организационной группой над мнениями экспертов;
- анализ занимает длительное время.

Метод декомпозиции работ. PERT

- Компании Remington Rand и Du Pont предложили метод критического пути (CPM);
- В США в рамках проекта Polaris создан метод анализа и оценки длительности реализации работ проекта — Program Evaluation & Review Technique (PERT);
- метод позволил закончить проект Polaris на два года раньше срока;
- В основе метода лежит декомпозиция работ на простые компоненты, каждый из которых поддаётся оценке;
- Сейчас произошла взаимная интеграция методов.

Для расчёта продолжительности одной операции используется аппроксимация β-распределения;

Приближённая оценка для β-распределения характеризуется тремя величинами: минимально возможной (m_e), максимально возможной (M_e) и наиболее вероятной (p_e), а ожидаемая величина:

$$te = (m_e + 4p_e + M_e) / 6,$$

Среднее квадратичное отклонение — $\sigma_e^2 = (M_e - m_e) / 6$.

При независимых величинах общая оценка вычисляется как сумма оценок.

Расчёт длительности производится на основе сетевой модели, которая используется при планировании;

Граф проекта представляет собой сеть, где начальная вершина — начало работы, конечная — завершение.

Длительность операции определяет длину дуги.

Длина критического пути в графе — минимальное время, за которое можно выполнить проект.

Средняя продолжительность проекта ТЕ — сумма средних значений для выполнения операций на критическом пути.

Зная дисперсии операций, выполняют расчет длительности.

Метод декомпозиции работ — один из самых удачных, прекрасно зарекомендовавших себя на практике.

Возможность успешного планирования работ позволила применять его для проектов в различных отраслях.

Однако применение его в программных проектах наталкивается на трудности, связанные с высокой степенью неопределенности их оценок.

Требуется проводить достаточно глубокую декомпозицию и подбирать квалифицированных экспертов

Метод оценки по аналогии ОПА. Оценка по аналогии использует эмпирические данные характеристик завершенных проектов.

Особенность метода состоит в том, что он позволяет выделить схожие проекты.

Схема оценки:

- сбор данных по разрабатываемому проекту;
- поиск и анализ проектов, аналогичных разрабатываемому по выбранным характеристикам;
- экспертная оценка разрабатываемого проекта
    - Характеристикам проекта присваиваются веса согласно значимости.
    - Для выбора наиболее близких проектов используется норма в n–мерном пространстве, где проекты и их характеристики представляются точками.
    - Затем вычисляется расстояние между точками.
    - Считается, что у проектов, имеющих наибольшее сходство, расстояние будет минимальным.

6. Основные методы оценки программных проектов. Размерно-ориентированные метрики

Предназначены для того, чтобы получить представление о свойствах программного изделия на основе прямых измерений некоторых его характеристик.

Наиболее распространенная методика основана на оценках количества строк кода программ или подобных мерах.

Считаем, что объём программного изделия измеряется в тысячах строк кода (СК). Например, 5 СК означает 5 тыс. строк исходного кода.

Англоязычное обозначение — LOC (Lines Of Code) или в тысячах KLOC.

Первоначально оценки, основанные на измерении СК, применялись, когда одной строка кода соответствовала одной команде языка.

Со временем это соответствие перестало выполняться: одна строка исходного кода может содержать несколько команд языка и наоборот.

Кроме того, большую роль стал играть стиль программирования, требующий поддержки определённой структуры программы, наличия комментариев и т.п.

Чаще всего рассматривают два вида оценки:

- количество строк кода с комментариями и пустыми строками, количество которых ограничивают какой-то долей от общего числа строк в измеряемом блоке;
- количество операторов исходного кода. Здесь обычно исключают пустые строки и комментарии.

Не существует единственного общепризнанного подхода к оценке, приемлемого для различных языков программирования и ориентированного на универсальное применение.

Варианты метрик:

- число строк, содержащих исходный код и комментарии;
- процент комментариев;
- среднее число строк для функций (методов);
- среднее число строк для модулей;
- среднее число строк для классов.

Кроме того, применяются и другие показатели: число модулей, функций/методов, классов и т.п.

Обозначим :

l — объём кода,

t — трудоёмкость (общие затраты времени на проект),

p — производительность,

c — стоимость проекта,

u — удельная стоимость.

Тогда, если известны производительность труда и удельная стоимость по объёму кода вычисляются как трудоёмкость разработки, так и стоимость работ:

$t = v \times p$, $c = v \times u$

Производительность и удельная стоимость не одинаковы для различных проектов, как и для различных компонентов одного проекта.

Тогда нужно не только определять эти параметры, но и корректировать их для разных типов проекта и разных компонентов.

Для этого формируется метрическая база характеристик выполненных проектов, по которой вычисляется производительность и удельная стоимость каждого разработанного ранее компонента:

$P\,i = v\,i / t\,i$, $u\,i = c\,i / v\,i$, где i — номер компонента

Схема процесса оценки:

- Выполняется декомпозиция проекта на компоненты.
- Для каждого компонента оценивается большее, меньшее и вероятное значение объёма, вычисляется ожидаемое количество строк кода Ni.

- Трудоёмкость вычисляется как скалярное произведение (N, p), где N — ожидаемые значения объёма, p — вектор производительности разработки аналогов. Если i-того аналога нет, берётся производительность проекта.
- Стоимость вычисляется как скалярное произведение (N, u), где u — вектор удельной стоимости разработки аналогов сили проекта, если аналога нет
- Трудоёмкость и стоимость линейно зависят от количества строк написанного кода.

Согласно мнению Ф. Брукса, «наши методы оценки ошибочно путают достигнутый прогресс с затраченными усилиями».

Мнение Р. Гласса: Нет ни одного довода в пользу того, что оценка самого количества строк сколько-нибудь легче, чем оценка затрат и сроков. И не очевидно, что есть универсальная методика преобразования LOC в денежные и временные единицы. И LOC одной программы может очень сильно отличаться от LOC другой.

Тем не менее, размерно-ориентированные метрики, возможно, самый распространённый способ оценки, хотя при его применении можно попасть в известную ловушку: если поощряются программисты, которые пишут много кода и исправляют много ошибок, легко отличиться, написав огромное количество некачественного кода и исправить в нём собственные же ошибки.

Достоинства метрик:

- простота расчёта;
- понятность как процесса оценки, так и результата, особенно для представителей заказчика;
- высокий уровень объективности оценки: слабая зависимость от экспертных оценок.

Недостатки метрик:

- нельзя или трудно получить достоверные данные до начала разработки;
- нет оценки для баз данных;
- нельзя оценить сложность интерфейса и выходных документов;
- нет возможности адекватно оценить сложные алгоритмы;
- оценки зависят от языков программирования;
- невозможно учесть различную производительность программистов.

7. Основные методы оценки программных проектов. Функционально-ориентированные метрики

Использование широкого спектра алгоритмических языков приводило к несовместимости оценок проектов и к некорректности наработанной метрической базы.

Прогноз параметров выполнения проектов невозможен.

Развитие практического программирования потребовало качественный пользовательский интерфейс и сложные структуры данных.

Тогда появился метод косвенной оценки ПО: метод функциональных точек (ФТ), на языке оригинала Function Points (FP), предложенный А. Альбрехтом.

Функциональные точки – условные элементарные единицы.

Методика основана на измерении не размера, а функциональности (полезности) продукта.

Функциональность оценивается с помощью информационных характеристик (функциональных типов, классов компонентов), которые строятся так, чтобы область их действия охватывала как внутренние характеристики приложения, так внешние.

Под транзакцией будем понимать элементарный замкнутый и неделимый процесс, переводящий систему из одного состояния в другое.

Для анализа используется пять классов компонентов.

Характеристики приложения относятся к трем категориям уровня сложности (низкий, средний, высокий).

Класс компонентов	*Англоязычное наименование*	*Описание*
Внешний ввод	External Input (EI)	перемещение данных в приложение
Внешний вывод	External Output (EO)	перемещение данных из приложения
Внутренний логический файл	Internal Logical File (ILF)	распознаваемый пользователем набор связанных данных
Внешний интерфейсный файл	External Interface File (EIF)	распознаваемый пользователем набор связанных данных, который передаётся другому приложению или получается от него
Внешний запрос	External Inquiry (EQ)	формирование данных на основе внешних интерфейсных или внутренних логических файлов без их модификации и расчётов

Для ввода, вывода и запроса сложность определяется количеством различных элементов данных и числом транзакционных файлов.

Для определения уровня сложности файлов вводится соотношение между числом файлов и компонентов, на основании которого проходит оценка в невыровненных функциональных точках по трем градациям: низкий (Н), средний (С), высокий (В).

Далее вычисляется объем продукта в невыровненных ФТ (UFP).

Для вычисления оценки (базовой меры) проекта вначале подсчитывается сумма коэффициентов регулировки сложности (каждый из которых оценивается от нуля до 5 с шагом 1 (TDI).

Затем определяется фактор выравнивания:

$$VAF = (TDI \times 0{,}01) + 0{,}65$$

№	*Системные характеристики*
1	Обмен данными
2	Распределенная обработка данных
3	Производительность
4	Ограничения по аппаратным ресурсам
5	Транзакционная нагрузка
6	Интенсивность взаимодействия с пользователем
7	Эргономичность
8	Интенсивность изменения данных ILF пользователем
9	Сложность обработки
10	Повторное использование
11	Удобство установки

12	Удобство администрирования
13	Переносимость
14	Гибкость

Окончательный расчет количества выровненных функциональных точек различен для разных типов проектов, в частности, для проектов разработки (DFP):

$$DFP = (UFP + CFP) \times VAF$$

где CFP — дополнительные функциональные точки, требуемые при установке.

1 ФТ – простая программа, выполняется за день.

10 ФТ – типичное настольное приложение, месяц.

100 ФТ – предел программиста-одиночки, 6 месяцев.

1000 ФТ – приложение для команды 10 чел., год.

10 000 ФТ – 100 человек, 1,5-5 лет, обычно не укладывается в срок.

100 000 ФТ – 100 человек, 5-8 лет, обычно безнадёжный проект.

1 ФТ – простая программа, выполняется за день.

10 ФТ – типичное настольное приложение, месяц.

100 ФТ – предел программиста-одиночки, 6 месяцев.

1000 ФТ – приложение для команды 10 чел., год.

10 000 ФТ – 100 человек, 1,5-5 лет, обычно не укладывается в срок.

100 000 ФТ – 100 человек, 5-8 лет, обычно безнадёжный проект.

Масштаб одной функциональной точки

- Ассемблер 320
- C 128
- Фортран 106
- Паскаль 90
- C++ 64
- Java 53
- Visual C++ 34
- Visual Basic 32
- Delphi 5 29
- Perl 21

Достоинства метода:

- измерения не зависят от применяемой платформы;
- метод обеспечивает единообразный подход к оценке всех проектов организации;
- применение метода основано на анализе требований, оценка трудозатрат может быть выполнена раньше;
- оценки продолжают уточняться по ходу жизненного цикла.

Недостатки метода:

- высокая сложность и трудоемкость метода;
- субъективность оценок системных характеристик;
- некорректность оценки сложных алгоритмов;
- потеря времени на повторный анализ. На начальном этапе проектирования трудно определить количество предполагаемых объектов, транзакций и операций ввода/вывода. По мере продвижения разработки приходится проводить повторный анализ и пересчитывать основные показатели.

- Если требования к системе не отражают сложности реализации, метод функциональных точек даёт заниженные результаты.
- Для оценки подобных проектов Кейперс Джонс предложил методику анализа точек свойств — Feature Points, которая позволяет проводить учет не только требований к системе, но и особенностей ее реализации.
- Основа метода состоит в оценке сложности алгоритмов и модифицирует степень значимости для расчета.

8. Основные методы оценки программных проектов. Оценка на основе статистики

Несмотря на существенные достижения рассмотренных методов оценки параметров реализации программных проектов, опытный руководитель проекта скажет: «Лучший способ узнать длину пути — это пройти его».

Далее приводятся методы оценивания на основе анализа эмпирических данных.

Разработана Барри Боэмом в 1981 году на основе анализа проектов TRW Aerospace различных типов.

Варианты модели:

- базовая – затраты как функция размера;
- промежуточная – добавляются атрибуты стоимости: оценки продукта, аппаратуры, персонала и проектной среды;
- усовершенствованная – добавляется влияние атрибутов стоимости на каждый этап процесса разработки.

Режим	*Размер*	*Уровни детализации*
Внедренный	Более 300 тыс. строк	Крупный проект, большая команда, значительный объем инноваций, значительная нестабильность внешней среды
Сблокированный	50–300 тыс. строк	Средний проект с инновациями, небольшая команда. Незначительная нестабильность внешней среды
Органичный	До 50 тыс. строк	Некрупный проект, небольшая команда, для которой нехарактерны нововведения. Стабильная внешняя среда

Базовый уровень не учитывает различия в аппаратных ограничениях, качестве и опыте команды проекта, техники и средствах разработки.

Уравнения базовой подмодели:

- затраты в человеко-месяцах: $E = a \times V^b$
- срок разработки в месяцах: $T = c \times E^b$
- количество участников: $K = E / T$,

где V — объём в строках кода, коэффициенты a, b, c, d определены для разных режимов.

Затраты нелинейно зависят от размера проекта и изменяются скачком при изменении режима.

Рост E при переходе на более высокий режим не всегда означает увеличение длительности (T) выполнения проекта.

На более высоких уровнях модель COCOMO усложняется, появляются коэффициенты для повышения точности оценок.

Модель позволяет проводить калибровку на основе исторических данных по осуществленным проектам.

Существенное расширение модели, добавлены следующие разделы:

- модель композиции приложения;
- модель раннего этапа проектирования;
- модель пост-архитектуры.

Модель адаптирована к современным методологиям разработки: учитываются различные модели разработки, метрики, уровни зрелости.

Единиц косвенного измерения — объектные точки на (количество и сложность форм, отчётов и т.п.)

Достоинства метода COCOMA II:

- модель ориентирована на начальные этапы разработки;
- по мере продвижения проекта учитывается вновь появляющаяся информация;
- коэффициенты модели основаны на реальной статистике;
- существует большое количество калькуляторов для модели.

Недостатки метода:

- субъективность при оценивании;
- основа модели – размерно-ориентированный подход с его недостатками, хотя при помощи калибровок и поправочных коэффициентов их влияние можно снизить;
- эмпирические коэффициенты были подтверждены экспертами, однако часто невозможно получить оценки с необходимым уровнем достоверности;
- сложность проведения оценки;
- неадекватность оценки, если вновь разрабатываемый проект значительно отличается по типу от используемой выборки.

Постановлением Государственного комитета СССР по труду и социальным вопросам и Секретариата ВЦСПС утверждены «Укрупнённые нормы времени на изготовление и сопровождение программных средств вычислительной техники». Приведённые в них собственные модели оценки трудоемкости и разработки ПО основаны на COCOMO, и задача оценки размера, трудоемкости и разработки программной системы была решены схожим образом.

9. Основные методы оценки программных проектов. Динамические оценки

10. Конус неопределенности и EQF фактор. Определение, свойства, области применения.

Модель динамической оценки, основанная на «конусе неопределенности» (Cone of Uncertainty, CofU), предложена Барри Боэмом. Она предназначена для описания количества неопределенности в ходе выполнения проекта.

В начале проекта у проектной команды слишком мало сведений как о предстоящей работе, так и о продукте, и предварительная оценка имеет высокую степень неопределенности. Со временем, когда уже проведены исследования и начата разработка, информации о проекте становится больше, и неопределенность снижается, а к концу проекта достигает нулевого значения.

Достоинства конуса неопределенности:

- простота и прозрачность;
- позволяют понять тенденцию оценок каждого эксперта;

- конус не влияет на улучшение оценки, но позволяет прогнозировать отклонения от плана с достаточной точностью.

Недостатки конуса неопределенности:

- обманчивые и односторонние определения;
- искажение практической оценки.

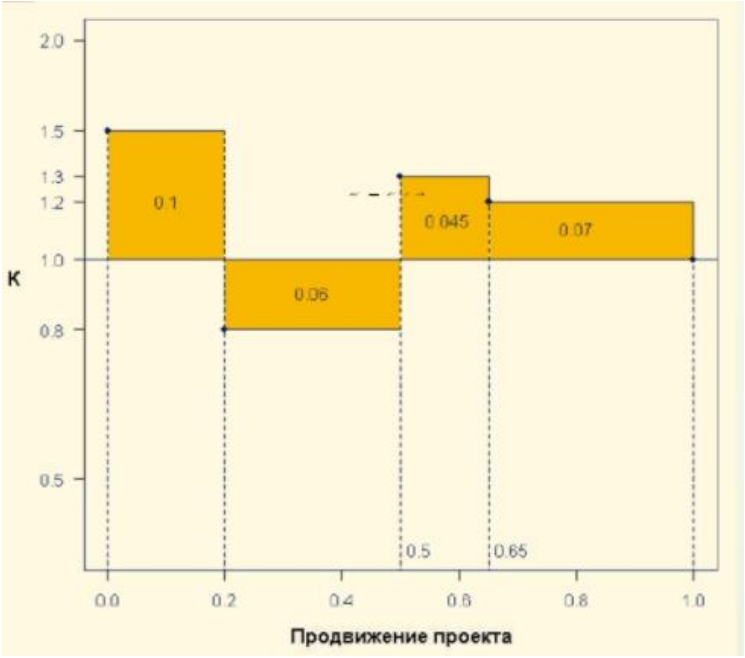
Построение конуса неопределенности, а по сути отношения прогнозируемой величины к актуальной, показывает потенциальную предвзятость эксперта, вовлеченного в прогноз.

Горизонтальная ось содержит основные вехи проекта, вертикальная ось содержит степень ошибок, которые могут быть совершены экспертом на различных этапах проекта.

- Данные оценки могут относиться ко всему содержимому проекта - размер усилий, цена, свойства или их комбинация.
- Оценки, сделанные на стадии инициации, могут быть не точными порядка четырех раз, как в большую, так и в меньшую сторону.

Прогнозируемая стоимость	**Дата прогноза**	**Продвижение проекта (%)**	**Коэффициент K=F/A (Forecast/Actual)**
30	01.01.2011	0	**1.5 =(30/20)**
16	19.02.2011	20	**0.8**
26	02.05.2011	50	**1.3**
24	**08.06.2011**	**65**	**1.2**

Для этой таблицы построим график зависимости (Forecast/Actual),) от процента продвижения проекта. Для каждой оценки посчитаем отклонения от реального значения, вычисляя площадь прямоугольника между оценкой и реальным значением.



Рассчитаем EQF как обратную величину к суммарной площади.

$$EQF=1/(0.1+0.006+0.045+0.07)=3.6$$

Исходя из определения EQF

- для любого проекта становится возможным его расчет;
- низкое значение EQF соответствует тому, что все прогнозы были низкого качества и отклонение первоначальных прогнозов велико;
- высокое значение EQF соответствует тому, что все прогнозы были высокого качества и отклонение первоначальных прогнозов минимально;

- с использованием EQF становится возможным провести качественную оценку прогнозирования проектов и сравнить полученные результаты

Плюсы и минусы конуса

- простота и прозрачность;
- позволяет понять тенденцию оценок каждого эксперта;
- обманчивые и односторонние определения;
- искажение практической оценки;
- бессмысленные данные;
- коническая форма конуса не является основанием улучшения оценки, но может быть использована как семейство распределений с ожидаемой точностью и прогнозируемым уклоном для дальнейшего прогноза.
- Построение конуса неопределенности, а по сути отношения прогнозируемой величины к актуальной (Forecast/Actual), показывает потенциальную предвзятость эксперта вовлеченного в прогноз.
- Распределение отношения прогноза к фактическому значению изменяется между организациями, по крайней мере, в трех измерениях: в точности оценки, в тенденции прогнозов сходиться и в систематической предвзятости эксперта.

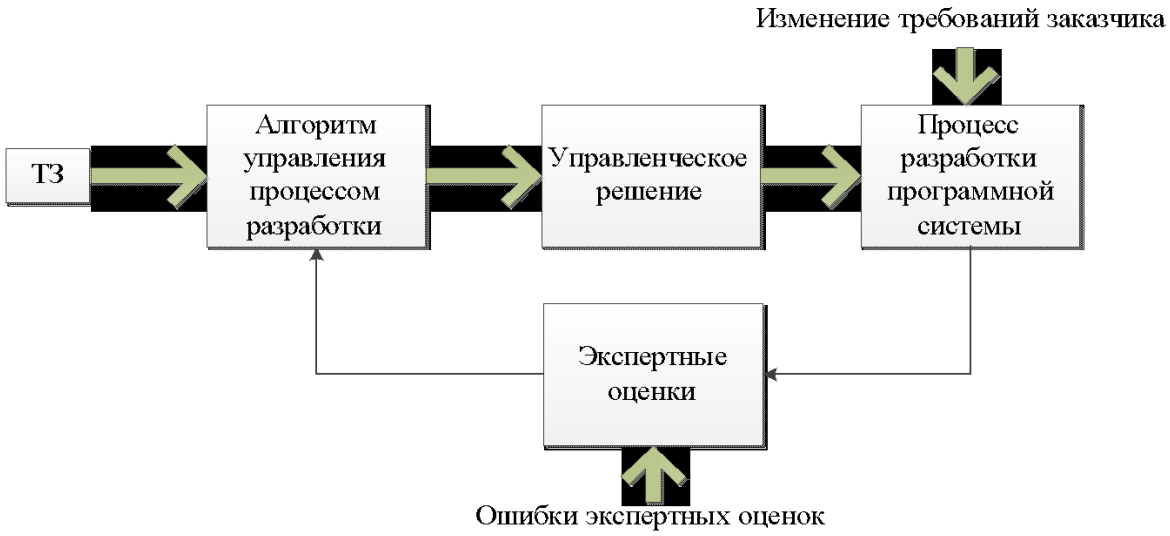
Требования к методу динамической оценки

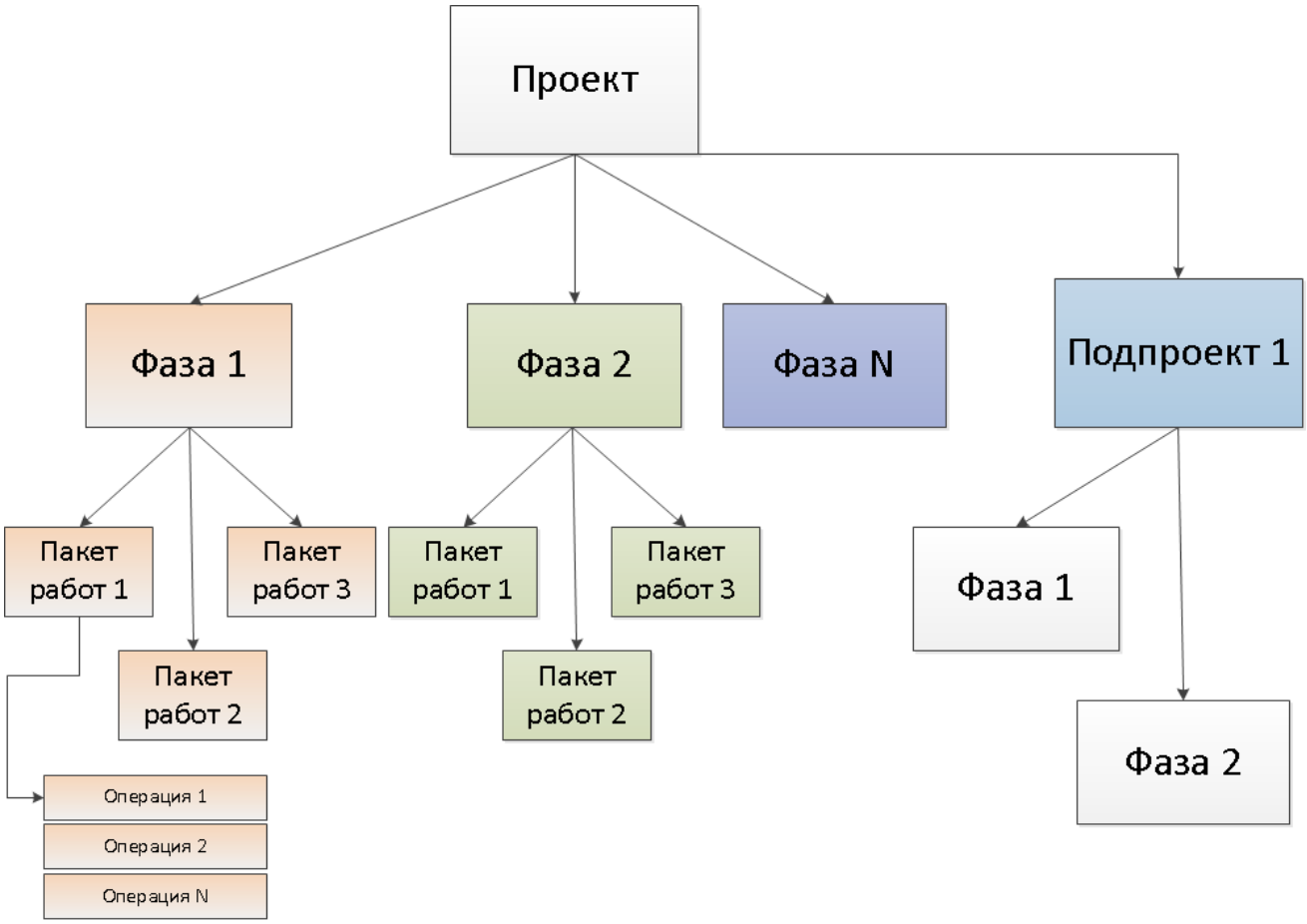
- применимость на практике;
- время и точность оценки должны удовлетворять требованиям оперативного управления;
- независимость от объёма программных проектов и выбранных методологий управления ими;
- гибкий учёт экспертной оценки;
- “прозрачность” для руководства;
- учёт достоинств и недостатков существующих моделей и методов.

Модель оценивания

- методика PERT для первого шага алгоритма;
- принцип Wideband Delphi для построения методики оценки;
- CofU и EQF фактор для анализа распределения оценок эксперта.
- MS Project в качестве базового программного обеспечения;
- алгоритм ДОПС;
- программную надстройку над MS Project, для реализации алгоритма оценки.

11. Структура проекта





Кратко об алгоритме оценивания

Этап предварительного анализа

- Разработать иерархическую структуру работ (ИСР); В данном процессе происходит декомпозиция целей проекта на более мелкие и более управляемые компоненты. Пакет работ - нижний уровень ИСР. Декомпозицию работ необходимо производить до тех пор, пока станет возможно реалистично оценить сроки и ресурсы каждого пакета;
- Определить взаимосвязь операций, используя метод диаграмм предшествования;
- Построить сетевую диаграмму;
- Разбить проект на фазы, каждую фазу разбить на пакеты работ

Определение количества предполагаемых экспертов

- Для каждого пакета работ фазы найти и определить количество предполагаемых экспертов. Для каждой фазы проекта, подпроекта, работы, находящейся на первом уровне иерархии, необходимо найти и определить количество предполагаемых экспертов.
- Стоит отметить, что сложность для оценки представляют новые уникальные операции. Для каждой задачи, находящейся на нижнем уровне иерархии, достаточно оценок 3-5 экспертов.
- По мере детализации и роста ИРС в глубину количество предполагаемых экспертов может быть скорректировано. Выбранных экспертов необходимо включить в состав проектной команды, разграничить роли и обязанности.

4. Начало работ по проекту

- 4.1. Детализировать активную фазу проекта на элементарные пакеты работ;
- 4.2. Для каждого пакета работ фазы оценить длительности для каждого эксперта;
- 4.3. Для первого завершённого пакета работ активной фазы (все работы внутри пакеты закончены) рассчитать коэффициент $K=F/A$ (отношение прогнозного значения оценок эксперта к фактической длительности);
- 4.4. Построить график зависимости K от процента продвижения пакета работ.
- 4.5. Для каждого эксперта в пакете работ фазы рассчитать EQF;
- 4.6. На основе полученных данных переопределить значение весовых коэффициентов доверия экспертов
- 4.7. На основе полученных коэффициентов переопределить оценку длительности критического пути проекта;
- 4.8. Разместить полученную информацию в базу экспертных оценок;
- Для каждого завершённого пакета работ текущей фазы сохранить параметры для базы знаний

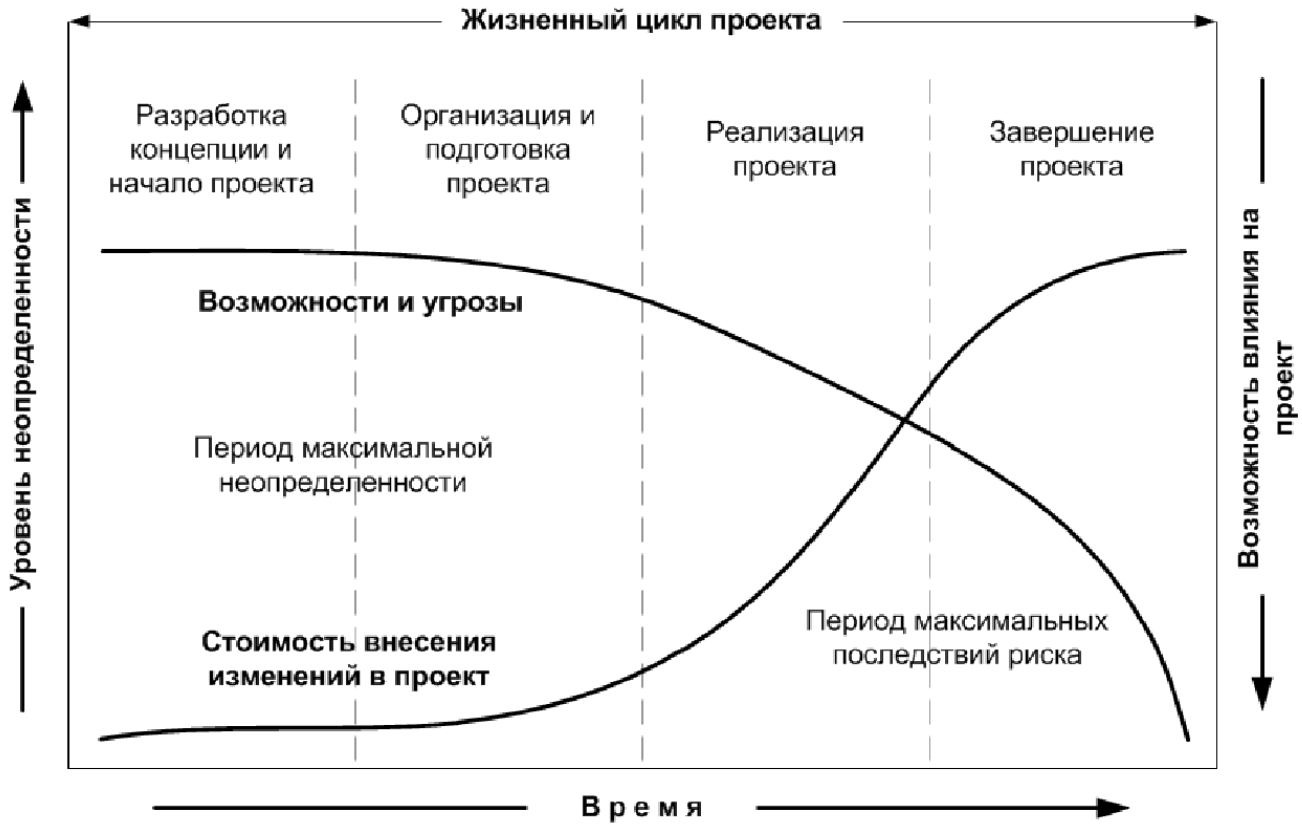
5. **Повторить шаги с 4.2 по 4.8 для всех оставшихся фаз.**

Стоимость и Функциональность могут быть оценены как функции времени.

12. Риски проекта. Определение, примеры

Риск проекта – это неопределённое возможное событие или условие, которое в случае возникновения может иметь как позитивное, так и негативное воздействие на один из KPI проекта

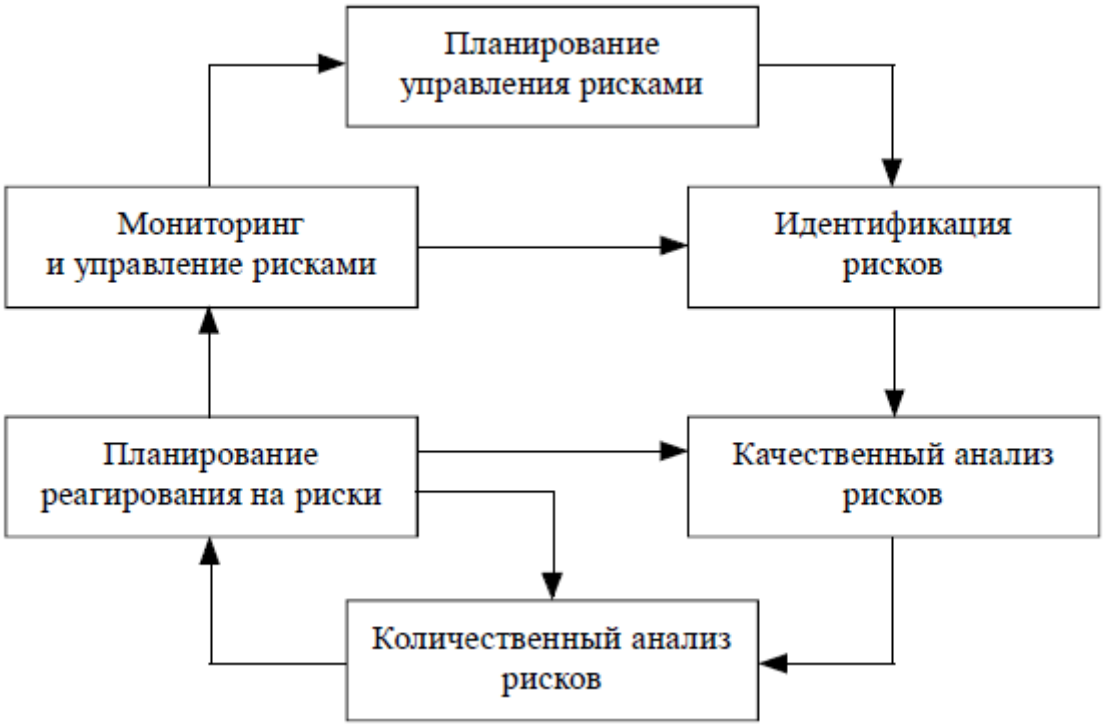
- Риск может быть вызван несколькими комбинированными причинами и в случае возникновения, чаще всего, оказывать негативное воздействие на несколько KPI
- Вероятность возникновения риска – вероятность того, что риск наступит в проекте
- Риск с нулевой вероятностью не считается риском, риск с **вероятностью 100%** является гарантированным событием, это факт, который необходимо учитывать в ходе планирования ресурсов
- Риск может иметь как негативное, так и положительное влияние на проект. Положительный риск называют “шансом”
- Управление рисками включает в себя процессы, относящиеся к планированию, анализу, идентификации и реагированию, мониторингу и управлению
- Данные процессы подлежат обновлению в ходе всего цикла проекта
- Цель управления рисками – увеличение вероятности возникновения и степени воздействия благоприятных событий и снижение вероятности возникновения и степени воздействия неблагоприятных событий
- Риски влияют на все основные ограничения проекта. Следовательно, управление рисками – это управление всеми значимыми параметрами проекта.



Процессы управления рисками проекта включают

- (**1**) планирование управления рисками
- (**2**) начальную идентификацию рисков, подготовку управления ими
- (**3**) качественный анализ рисков
- (**4**) количественный анализ рисков
- (**5**) выработку плана реагирования на риски
- (**6**) управление и постоянный мониторинг рисков

Процессы управления рисками



Ошибки при оценке риска или человеческий фактор

- Эффект «репрезентативности» - переоценка надежности малых выборок, неслучайный характер выборки
- Эффект «наглядности» – переоценка “простых”, “понятных”, частых и запоминающихся рисков
- Эффект «эгоцентризма» – ориентация на собственный накопленный опыт, а не доступные данные
- Эффект «консерватизма» – субъективный подход или сложившееся мнения о каких либо событиях
- Эффект «края» – недооценка высоко вероятных событий и переоценка маловероятных
- Эффект «Монте-Карло» – желание связать между собой два последовательных события

13. *Процесс управления рисками. Планирование управления рисками*

- **Планирование управления рисками** – процесс определения подходов и планирования операций по управлению рисками проекта
- Создание **плана управления рисками**
- Инструментарий:
  - совещания по планированию и анализу
  - анализ предыдущего опыта
  - выбор существующих методов, методологий и практик для управления рисками

План управления рисками

- Методология
- Распределение ролей и ответственности
- Разработка бюджета
- Определение сроков и частоты управления рисками
- Категории рисков + иерархическая структура рисков (Risk Breakdown Structure, RBS)
- Определение вероятности возникновения рисков и их последствий
- Матрица вероятностей и последствий
- Формы отчетности

Иерархическая структура рисков

Источники рисков

14. *Процесс управления рисками. Начальная идентификация рисков*

- Идентификация рисков – **выявление рисков**, способных повлиять на KPI проекта и документальное оформление их характеристик
- Идентификация рисков – постоянный итеративный процесс, в котором принимают участие все члены команды, это необходимо для совместного разделения ответственности за риски, а также за действия по реагированию на них
- Результат идентификации – **реестр рисков**
- **Риск с нулевой вероятностью - это факт, с ним нужно работать сразу!**

Документы для идентификации рисков

- устав проекта
- план управления рисками
- оценка длительности операций, стоимости операций
- базовый план по содержанию
- реестр участников проекта
- план управления стоимостью, содержанием, качеством
- факторы внешней среды
- предыдущий накопленный опыт
- другая документация проекта

15. *Процесс управления рисками. Методы идентификации рисков*

Методы идентификации рисков

- Мозговой штурм
- SWOT анализ (анализ сильных и слабых сторон, возможностей и угроз), PEST анализ, PESTLE анализ
- Метод Дельфи
- Методы с использованием диаграмм (Диаграммы Ишикавы, Диаграммы влияния)
- Метод номинальных групп
- Карточки Кроуфорда
- Анализ контрольных списков
- Метод аналогии
- Анализ допущений и др.

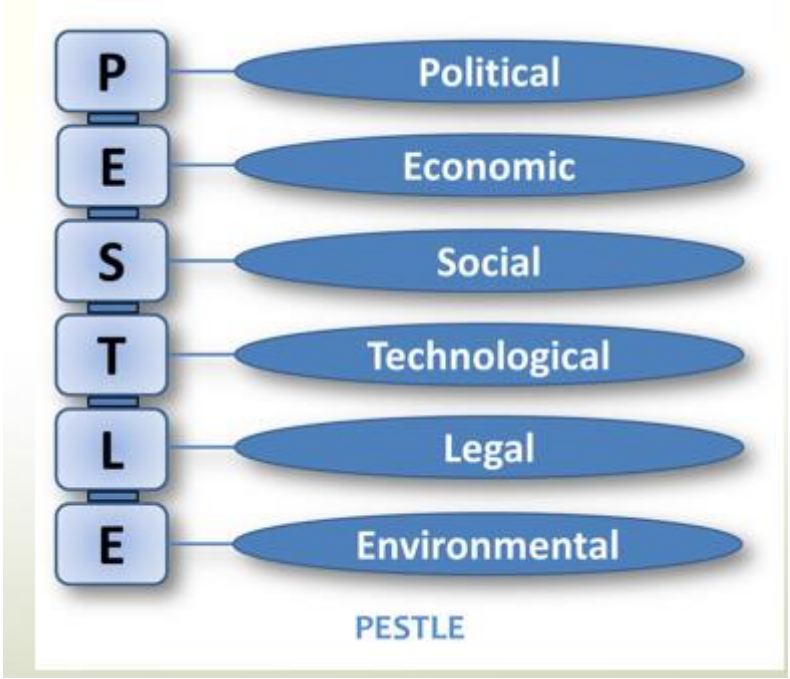
SWOT

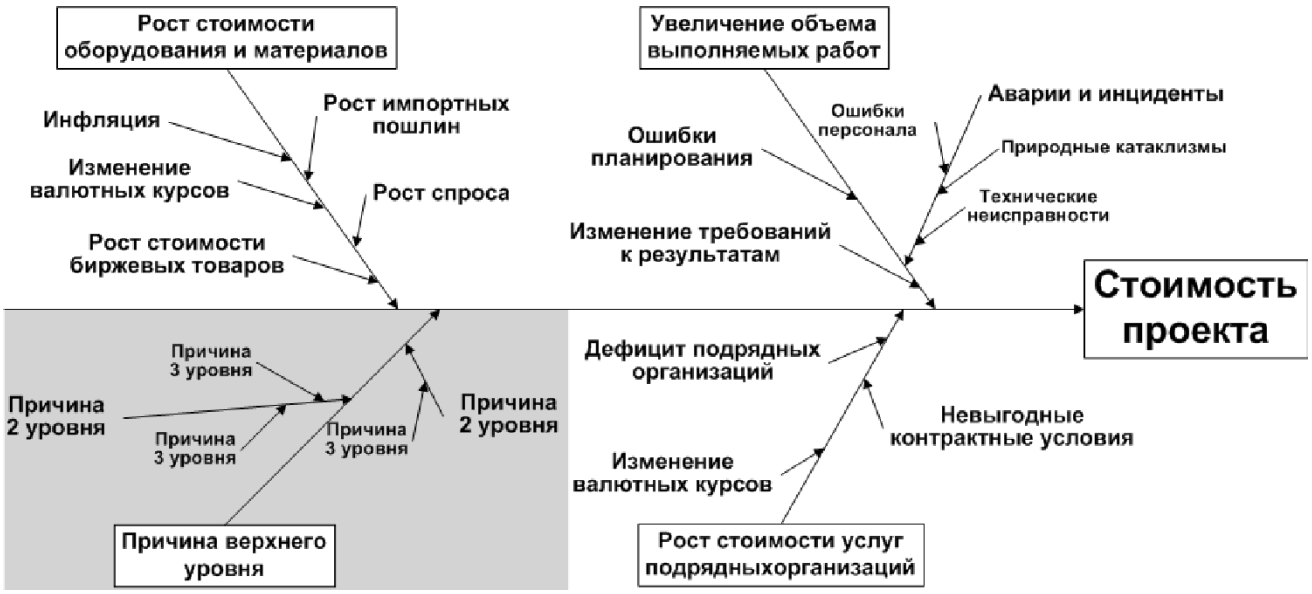
	Положительное влияние	Отрицательное влияние
Внутренняя среда	**S**trengths (свойства проекта или коллектива, дающие преимущества перед другими в отрасли)	**W**eaknesses (свойства, ослабляющие проект)

Внешняя среда	**O**pportunities (внешние вероятные факторы, дающие дополнительные возможности по достижению цели)	**T**hreats (внешние вероятные факторы, которые могут осложнить достижение цели)

Силы: • Уютное кафе, с неповторимым стилем; • Уникальная концепция; • Тихое место, недалеко от МГУ им. Ломоносова.	**Слабости:** • Слабо проходимое место, в котором расположено кафе; • Нехватка мощностей, для удовлетворения потенциального спроса.
Возможности: • Прирост потребителей вследствие разрастания района; • Появление постоянных клиентов вследствие зарабатывания имиджа и базы.	**Угрозы:** • Повышение арендной платы; • Проигрыш конкурентам доли рынка.

- **PEST-анализ** (к **STEP**) —это маркетинговый инструмент, предназначенный для выявления политических (**P**olitical), экономических (**E**conomic), социальных (**S**ocial) и технологических (**T**echnological) аспектов внешней среды, которые влияют на KPI проекта
- **PESTLE = PEST** +2 фактора (**L**egal) и (**E**nvironmental)





Реестр рисков содержит

- список идентифицированных рисков
- список действий по реагированию
- основные причины возникновения риска
- уточнённые категорий рисков

16. *Процесс управления рисками. Качественный анализ рисков*

- Цель – **приоритизация рисков**
- Включает в себя расстановку приоритетов для идентифицированных рисков. Результаты используются в ходе количественного анализа рисков и при планировании реагирования на риски
- Приоритеты идентифицированных рисков определяются на основании **вероятности** их возникновения, и **степени влияния** на KPI проекта
- Качественный анализ рисков подлежит уточнению на протяжении всего жизненного цикла проекта

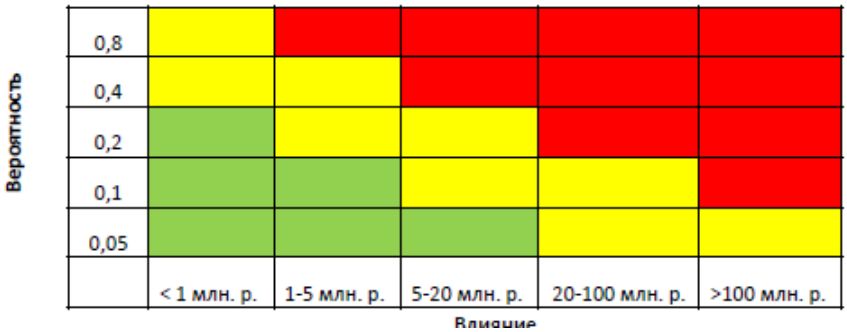
Инструменты качественной оценки рисков

- Матрица: вероятность - последствия
- Создание карты рисков
- Метод парных сравнений Саати
- FMEA-анализ

Качественные оценки

Балльные оценки

Количественные оценки



Проект Цель	Показаны значения по относительной и числовой шкалам				
	Очень низкая / 0,05	Низкая / 0,10	Умеренная / 0,20	Высокая / 0,40	Очень высокая / 0,80
Стоимость	Незначительное увеличение стоимости	Увеличение стоимости <10%	Увеличение стоимости 10-20%	Увеличение стоимости 20-40%	Увеличение стоимости >40%
Сроки	Незначительное увеличение времени	Увеличение времени <5%	Увеличение времени 5-10%	Увеличение времени 10-20%	Увеличение времени >20%
Содержание	Едва заметное уменьшение содержания	Затронуты второстепенные области содержания	Затронуты основные области содержания	Уменьшение содержания неприемлемо для спонсора	Конечный продукт проекта фактически бесполезен
Качество	Едва заметное понижение качества	Затронуты только самые трудоемкие приложения	Для понижения качества требуется одобрение спонсора	Понижение качества неприемлемо для спонсора	Конечный продукт проекта фактически бесполезен
В этой таблице представлены примеры определения воздействия риска на каждую из четырех различных целей проекта. Каждое из воздействий следует в ходе процесса планирования управления рисками адаптировать к конкретному проекту согласно принятым в организации порогам рисков. Подобным же образом можно разработать определение воздействия для благоприятных возможностей.					

17. Процесс управления рисками. Количественный анализ рисков

- Цель - оценка потенциального **воздействия всех идентифицированных рисков на KPI проекта**
- Проводится в отношении рисков, которые в процессе качественного анализа рисков были квалифицированы как потенциально или существенным образом влияющие на KPI
- Данный анализ представляет количественный подход к принятию решений в условиях неопределенности.
- Инструменты:
  - анализ чувствительности,
  - сценарный анализ,
  - моделирование Монте-Карло и др.

18. *Процесс управления рисками. Стратегии реагирования на риски*

- Цель – разработка возможных действий, способствующих **повышению благоприятных возможностей и снижению угроз** для достижения KPI проекта
- Включает в себя определение и назначение ответственных лиц, в обязанности которых входит реагирование на риск.
- Запланированные мероприятия должны соответствовать степени риска, быть экономически эффективными, своевременными и реалистичными в контексте проекта
- Часто требуется выбор наилучшего способа реагирования на риски из нескольких возможных вариантов.

Угрозы

- Уклонение
- Передача
- Снижение
- Принятие

Возможности

- Использование
- Совместное использование
- Усиление
- Принятие

Стратегии реагирования на риски Угрозы-уклонение

- По сути это исключение опасности. Включает все мероприятия, для защиты KPI проектов
- Возможно, придется изменить сами цели — смягчить требования, узнать дополнительную информацию
- Например, если появляется риск сорвать сроки проекта, можно попробовать упростить продукт, сократить количество задач, улучшить коммуникацию

Как реагировать: прекращение проекта, отказ от части работ, изменение тройственного ограничения, смена подрядчиков, технологий и т.д.

Стратегии реагирования на риски угрозы-передача

- подразумевает передачу ответственности за риск третьей стороне, при этом риск не устраняется
- Передача ответственности за риск является наиболее эффективной в отношении финансовых рисков. Передача риска практически всегда предполагает выплату премии за риск стороне, принимающей на себя риск (страховки, гарантии выполнения контракта, аутсорсинг, хеджирование т. д.)

Как реагировать: платить за страховки, разделять риски, включать гарантийные обязательства в проект

Стратегии реагирования на риски угрозы-снижение

- Снижение рисков предполагает понижение вероятности возникновения и степени его последствий до приемлемых значений за счет “профилактических мер”. Принятие предупредительных мер наиболее эффективно, нежели усилия, потраченные на устранение последствий
- Примеры: покрыть все автотестами, выбирать более дорогого, но проверенного подрядчика, разработать прототип, 2 ЦОДа.
- Если невозможно снизить вероятность, ослабление риска должно быть направлено на последствия риска, а именно на те связи, которые определяют их серьезность.

Как реагировать: диверсификация, диссипация, локализация, прототипирование, повышение операционного контроля, отбор оборудования

Стратегии реагирования на риски угрозы/возможности – принятие

- Реагирование на последствия рисков без вмешательства в сам проект. Когда исключить или снизить риски проекта невозможно, их приходится принимать. Таким образом, приходится работать с негативными или позитивными событиями уже после того, как они произошли
- Принятие может быть пассивным и активным. Пассивное представляет собой игнорирование событий риска и экстренные меры по устранению последствий. Активное принятие — создание резерва ресурсов (денег, времени, людей) и самострахование.

Как реагировать: закладывать менеджерский коэффициент, надеяться на удачу ☺

Стратегии реагирования на позитивные риски Возможности-использование

- Предназначена для устранения неопределенностей, связанных с риском, при помощи мер, обеспечивающих наиболее вероятное появление данной благоприятной возможности
- Пример: привлечение к участию в проекте более талантливого\опытного сотрудника, с тем, чтобы сократить время, необходимое для его завершения, завышение качества, запаса по прочности, нежели было предусмотрено первоначальным планом

Как реагировать: увеличивать вероятность благоприятных для проекта событий

Стратегии реагирования на позитивные риски Возможности-совместное использование

- Предусматривает передачу ответственности третьей стороне, способной наилучшим образом воспользоваться представившейся благоприятной возможностью в интересах проекта.
- Пример: образование партнерств с командами с совместной ответственностью за положительные риски, образование специализированных компаний для управления благоприятными возможностями;

Как реагировать: увеличивать вероятность благоприятных для проекта событий + привлекать партнеров для взаимной выгоды

Стратегии реагирования на позитивные риски Возможности – Усиление

- При применении этой стратегии изменяется размер благоприятной возможности путем повышения вероятности возникновения или положительного воздействия
- Происходит выявление и максимизация основных источников этих позитивных рисков
- Для повышения данной вероятности можно попытаться укрепить причину, вызывающую благоприятную возможность, и целенаправленно усилить условия ее появления.

Как реагировать: необходимо повлиять на источники воздействия, максимизируя чувствительность проекта к этой благоприятной возможности

<table>
<tr><th>Вероятность потерь
Уровень потерь</th><th>Близкая к «0»</th><th>Низкая</th><th>Небольшая</th><th>Средняя</th><th>Высокая</th><th>Близкая к «1»</th></tr>
<tr><td>Незначительный</td><td colspan="4">Принятие риска</td><td colspan="2">Создание резервов или принятие риска</td></tr>
<tr><td>Малый</td><td colspan="3">Принятие риска</td><td colspan="3">Создание резервов</td></tr>
<tr><td>Допустимый</td><td colspan="2">Создание резервов</td><td colspan="3">Внешнее страхование и (или) разделение между участниками</td><td>Избежание риска</td></tr>
<tr><td>Средний</td><td colspan="4">Внешнее страхование и (или) разделение между участниками, диверсификация</td><td colspan="2">Избежание риска (отказ от проекта)</td></tr>
<tr><td>Высокий</td><td colspan="3">Внешнее страхование и (или) разделение между участниками, диверсификация</td><td colspan="3">Избежание риска (отказ от проекта)</td></tr>
<tr><td>Катастрофический</td><td colspan="2">Внешнее страхование и (или) разделение между участниками, диверсификация</td><td colspan="4">Избежание риска (отказ от проекта)</td></tr>
</table>

- Управление и мониторинг рисков
- Цель — управление и наблюдение за прогрессом выполнения принятых планов по предотвращению рисков и смягчению их вероятных последствий
- Во время мониторинга команда проекта выполняет планы по предотвращению рисков
- Обеспечивается своевременное исполнение превентивных мер по смягчению последствий с помощью "**симптомов риска**", указывающих на возможность того, что события риска произойдут в ближайшее время

Инструменты: мониторинг и контроль!

19. *Качество ПО. Проблемы качества ИТ проектов*

- Возникла давно, трактовалась по-разному
- точность при ограничении на ресурсы
- надёжное хранение информации
- Доступные вычислительные ресурсы – новые проблемы
- сложность взаимодействия
- слабая защищённость
- перегруженность функциями
- низкая надёжность

Что позволит улучшить качество?

20. *Качество ПО. Управление качеством ИТ проектов*

Первоочередные меры по настройке системы управления качеством на предприятии (Шухарт)

- математически обоснованные требования к изготовлению каждой части изделия
- контроль – инструмент разделения системных и случайных ошибок
- оговаривается максимальное количество случайных ошибок
- совокупное количество дефектов в готовом изделии изначально учитывается в технологии производства

Управление качеством по Демингу – теория нравственности (1)

Избранные принципы Деминга (всего их 14)

3. Покончите с зависимостью от массового контроля
5. Улучшайте каждый процесс
6. Введите в практику подготовку и переподготовку кадров
7. Учредите лидерство
8. Изгоняйте страхи
12. Дайте работникам возможность гордиться своим трудом

Смертельные болезни и препятствия

• Никто не может преуспеть, используя только количественные критерии...

• Тот, кто управляет компанией, основываясь только лишь на точных числах, в скором времени останется и без компании, и без чисел...

• Наиболее важные для управления величины неизвестны и неопределимы количественно

Инженерный подход (1)

Систематический, структурированный, количественный подход к разработке, функционированию и сопровождению программных средств; способ применения инженерных методов в разработке программных продуктов

Сторонники подхода: организованная группа средних специалистов со знанием правильного процесса может легко переиграть команду суперпрограммистов, ориентирующихся на мастерство

Противники подхода: при быстрой разработке гибких, адаптируемых систем инженерные процессы непригодны, системы получаются громоздкими, неуклюжими и дорогими

Что понимается под качеством

Точки зрения

Заказчик: снята производственная проблема за невысокую цену и без дополнительных трудностей.

Пользователь: удобство работы с системой.

Конкуренция: небольшое время реализации.

Сопровождающий программист: легко адаптировать к изменяющимся условиям эксплуатации.

Качество и стандарты

• Стандарт ISO 8402:94. Качество – весь объём признаков и характеристик программ, который относится к их способности удовлетворять установленным или предполагаемым потребностям.

• ГОСТ РВ 51987–2002. Качество – совокупность свойств, обусловливающих их пригодность в соответствии с целевым назначением.

• ISO/IEC 25000:2014. Качество – способность программного продукта при заданных условиях удовлетворять установленным или предполагаемым потребностям.

21. Качество ПО. Модели качества.

Стандарт ISO 8402:94. Используется 21 характеристика качества, которые объединены в 6 групп: функциональная пригодность, надёжность, применимость, эффективность, сопровождаемость, переносимость.

ISO/IEC 25000:2014. Модель включает 8 характеристик: функциональная пригодность, надёжность, удобство использования, производительность, сопровождаемость, переносимость, совместимость, защищённость

Целевое назначение и пригодность

Заказчик: своевременная поставка, быстрая окупаемость, функциональная полнота, непротиворечивость информации, приемлемая производительность, надёжность, защищённость.

Пользователь: наличие эксплуатационной документации, удобство взаимодействия, снижение сложности выполняемых операций, уменьшение утомляемости.

Сопровождающий программист: единый стиль программного кода, нормализованная база данных, тестируемость, модифицируемость.

Специалисты о качестве

Р. Гласс переносимость, надёжность, эффективность, удобство работы, тестируемость, понятность, модифицируемость

А. Купер качество – простота взаимодействия пользователя с системой

Дж. Рейнвотер программное изделие качественное, если пользователь им удовлетворён

А.Н.Терехов изделие некачественное, если
- неадекватно функционирует,
- есть отказы при применении по назначению,
- нет эффективности по времени отклика,
- нарушается конфиденциальность информации,
- есть несоответствие хранимых данных и вводимой информации.

Для оценки и сравнения характеристик качества требуются измерительные алгоритмы на основе объективных критериев. Стандарты рекомендуют предусмотреть возможность объективного и воспроизводимого измерения каждой из характеристик качества для сопоставления с техническим заданием с учётом нормы допустимых ошибок измерения. В этом случае необходима процедура определения числовых значений параметров проекта.

Характеристики качества делятся на три уровня детализации показателей:
- количественные, которые можно измерить и численно сопоставить с требованиями;
- категорийные, которые представляются шкалой свойств;
- качественные, которые характеризуются порядковой или точечной шкалой (есть — нет, хорошо — плохо) и определяются экспертным методом.

Ошибки условно можно разделить на две категории: ошибки проектирования и ошибки реализации. Первый тип ошибок выявляется тестированием системы на соответствие ТЗ. Их наличие говорит о нарушении функциональной пригодности. Ошибки второго типа определяют надёжность, которая измеряется степенью покрытия тестами программных компонентов и системы в целом. Мерой может служить относительное количество протестированных функций и маршрутов, но при этом следует учитывать уровень потенциальной опасности отдельных элементов, оценка которого часто сложна и не однозначна.

22. Качество ПО. Риски качества

Разделим риски качества условно на четыре группы: 1) внешние, объективные; 2) проектные, но не связанные с разработчиками; 3) связанные с разработчиками, но не с программистами; 4) связанные с программистами.

Объективные риски

Связаны с событиями, внешними по отношению к проекту: • появление на рынке конкурирующих продуктов, • появление новой технология, • неожиданные правительственные решения, например, введение или изменение законов, вынуждающих срочно изменять техническое задание.

Проектные риски

В основном, ошибки в управлении проектом. Например, известно, когда будет противостояние Земли и Марса, легко вычисляется время старта корабля на Марс, следовательно, и срок, к которому должно быть готово программное обеспечение. Примерно можно определить период разработки качественного программного обеспечения. Но если сроки назначаются нереальные, корабль, немного полетав вокруг Земли, падает в океан.

Риски, связанные с разработчиками

Возникают на этапах, предшествующих программированию. Особо тяжёлый случай – некачественно проведённый анализ требований. Причины могут быть разными, результат печальный: качеству будет нанесён непоправимый урон. К значительному снижению качества приводит неудачное проектирование данных. Критичная проблема – взаимодействие процессов, особенно в реальном времени.

Риски программирования

Области риска: 1. Квалификация программиста 2. Сложность программы (объективная сложность, беспорядочный код, плохой стиль, неверное использование прототипа) 3. Некачественная отладка. 4. Переход на новые методы разработки. 5. Некорректная обработка нештатных ситуаций. 6. Неверная оценка компонентов при компонентном подходе.

23. Качество ПО. Модель CMMI

Системы качества

Комплексные системы управления качеством продукции формировались в США, Европе, Японии, Советском Союзе в 1970-е годы. Начало их внедрения – Саратовская система бездефектного изготовления продукции (середина 1950-х годов). В 1970-х годах во Львове была разработана комплексная система управления качеством продукции (КС УКП). Она позволила объединить элементы управления качеством в единую систему.

Модель технологической зрелости CMMI (Capability Maturity Model Integration)

Цель – определить надёжных разработчиков программного обеспечения
Профессиональная зрелость коллектива
характеризуется прочными связями между участниками, которые возникают на основе общих ценностных ориентаций, позитивных неформальных отношений

Эволюция организации на пути к зрелому производственному процессу

Уровень 1 начальный. Ключевых процессов нет.

Процессы носят случайный характер. Успех проекта зависит только от руководителя и исполнителя. Работа представляется как черный ящик.

Уровень 2 – повторяемый.

Есть контрольные точки внутри процесса. Процессами можно управлять по аналогии. Вводится понятие эффективности. Работа представляется как цепочка черных ящиков.

Группы ключевых процессов

1) управление требованиями 2) планирование проекта 3) мониторинг и контроль проекта 4) управление контрактами 5) обеспечение качества процесса и продукта 6) управление конфигурацией

Уровень 3 – определенный

Процессы документируемые, представляют собой единую технологическую среду. В проектах используется адаптированная версия стандартного производственного процесса

Группы ключевых процессов
7) определение производственного процесса 8) координация производственного процесса 9) обучение 10) интегрированное управление разработкой 11) инженерия разработки 12) межгрупповая координация 13) экспертные оценки

Уровень 4 – управляемый
Собираются количественные показатели производственного процесса и качества продукта. Они оцениваются и контролируются с количественной точки зрения. Все процессы проходят стабильно

Группы ключевых процессов
14) количественное управление процессом 15) управление качеством программного обеспечения

Уровень 5 – оптимизирующий

Количественная обратная связь с процессом. Реализация передовых технологий, проверка их на пилотных проектах и применение в остальных проектах. Выявляются дефекты, оцениваются их причины и предупреждается их появление.

Группы ключевых процессов

16) предотвращение дефектов 17) управление технологическими изменениями 18) управление изменениями процесса

Недостатки CMMI

Модель управляет проектом, а не процессом создания программного продукта; • нет анализа рисков, невозможно своевременно обнаружить проблемы; • для небольших компаний модель слишком бюрократична.

24. Качество ПО. Методика CBP

Методика критически важных практических навыков CBP

Лучший практический навык тот, который решит наибольшее число проблем

Цели методики

Внедрить лучшие навыки разработки ПО; тратить ресурсы на разработку, а не наλ формальное следование инструкциям; использовать лучшие практические навыки сλ учётом корпоративной культуры; дать возможность эффективно учитьсяλ методикам.

Общие рекомендации к методике
Делать упор на коллективный характерλ завершения разработки; придумать быструю стратегию реализации;λ измерять продвижение к цели;λ измерять активность разработки.λ

Принципы управления проектом
Ошибки выявляются и исправляются сразу; планирование проводится на основеλ правильных показателей; минимизируется количествоλ неконтролируемых изменений; эффективно используются сотрудникиλ

Лучшие навыки

1.Формальное управление рисками.
2.Соглашение об интерфейсе.
3.Формальная проверка проектов.
4.Управление проектами на основе метрик.
5.Контроль качества продукта на глубоком уровне.
6.Общедоступная информация о ходе проекта.
7.Исправление причин ошибок для достижения качества.
8.Управление конфигурацией.
9.Управление персоналом.

25. Жизненный цикл проекта. Фазы жизненного цикла проекта.

Пять фаз, обычно, следуют друг за другом по порядку, но бывают и исключения. Например, в случае изменений требований в ходе выполнения проекта вы можете вернетесь к фазе планирования, для учета изменений.

1. Инициализация (Initialization);

- на данном этапе закладывается фундамент будущего проекта, необходимо убедиться, что проект действительно можно осуществить, прежде чем вкладывать силы в планирование и последующие задачи
- включает в себя описание проекта, создание экономического обоснования, выявление ключевых участников и утверждение проекта соответствующими сторонами
- создается устав проекта, где будут отражены ваши основные цели, объем работ и ограничения
- По статистике только половина команды четко представляют себе бизнес-цели проекта

2. Планирование (Planning);

- на фазе планирования, создаётся и формализуются план проекта и проектная документация:
  - календарный план
  - матрица ролей и зон ответственности
  - KPI проекта
  - матрица рисков
  - Формируются правила внутрикомандного взаимодействия
- План проекта представляется команде на установочной встрече (kick –off проекта).
- План проекта становится ключевым документом в котором отражены ключевые цели и формализованы внтутрикомандное взаимодействие

3. Выполнение (Executing);

- после установочного совещания проект переходит в стадию Выполнения
- необходимо выполнять план проекта, следить за продвижением, в случае отставания принимать меры к его минимизации
- распределите, если есть, дополнительные ресурсы, подумаете как на каждом этапе сэкономить время (без потери качества), тем самым создав себе подушку безопасности

4. Контроль и мониторинг (Controlling and Monitoring);

- Необходимо проводить периодические контрольные совещания для отслеживания хода работ и корректировки плана проекта

- Цель встреч – текущий статус и получение обратной связи для оперативного сбора отзывов, чтобы не запускать проблемные или конфликтные ситуации
- Использование платформы управления проектами, например, MS Project, Primavera, spider project и друге
- Контроль и отслеживание наличия необходимых ресурсов. Важно действовать на опережение, по статистике, каждый 5 проект срывается из – за недостающих или ограниченных ресурсов.

5. Завершение (Closing).

    - сдача проекта клиенту и\или команде поддержки
    - размещение проектных документов и артефактов в централизованном хранилище (чтобы обратиться к ним в будущем, если потребуется)
    - «разбор полетов», что получилось, что не получилось, как этого избежать в следующих проектах
    - Праздник! Завершение проекта стоило немалых сил, и это достижение, которое нужно отметить. Признание заслуг имеет очень большое значение. По статистике примерно 60% сотрудников считают, что их вклад недооценен

Жизненный цикл — взаимосвязанные процессы создания и последовательного изменения состояния продукции от формирования исходных требований к ней до окончания её эксплуатации.

В определении стандарта ISO/IEC 12207 (ГОСТ Р ИСО/МЭК 12207-99) вместо совокупности процессов фигурирует период времени.

Общепринятого разбиения жизненного цикла программной системы на этапы нет.

Этап может выделяться в отдельный пункт, может входить как составная части в более крупный этап.

Характер ЖЦ определяется методологией.

Но цель всегда одна: провести разработку от замысла до готового продукта.

Традиционные этапы жизненного цикла:

- анализ — определение того, что должна делать система;
- проектирование — определение того, как система будет делать то, что она должна;
- программирование — создание функциональных компонентов и подсистем, соединение их в единое целое в соответствии с проектом;
- тестирование — проверка соответствия системы показателям, определенным ранее;
- ввод системы в действие;
- сопровождение — обеспечение штатного процесса эксплуатации системы у заказчика.

Структура жизненного цикла

Жизненный цикл строится в соответствии с принципом нисходящего проектирования.

Обычно реализованные этапы циклически повторяются в соответствии с изменениями требований и внешних условий.

На каждом этапе порождается набор технических решений и документов, для каждого этапа исходными служат документы и решения, принятые на предыдущем этапе.

В общем случае структура жизненного цикла зависит от назначения системы и режима её функционирования

Полный жизненный цикл

В реальном жизненном цикле есть этапы, предшествующие разработке, а сопровождение занимает гораздо больше времени, чем всё остальное.

- возникла необходимость в программной системе;
- появилась возможность её создать;
- система создана;

- система соответствует потребностям заказчика;
- система изменяется в соответствии с эволюцией объекта;
- система стареет, теряет способность к развитию;
- система гибнет из-за несоответствия потребностям заказчика и заменяется на новую.

Этапы ЖЦ стандарта ГОСТ Р ИСО/МЭК 12207

Стандарт описывает структуру процессов ЖЦ, но не конкретизирует в деталях, как выполнить действия и задачи, включенные в них.

Этапы:

- формирование требований к системе; 1
- проектирование; 2
- реализация; 3
- тестирование4;
- ввод в действие;5
- эксплуатация и сопровождение;6
- снятие с эксплуатации.7

- Формализуется назначение программной системы
- Определяются требования, которые представляет собой описание необходимого или желаемого свойства системы
- Различают функциональные и эксплуатационные требования
- Функциональные требования - требования к функциональности программной системы
- Эксплуатационные требования определяют особенности его функционирования и эксплуатации
- Этап заканчивается разработкой **технического задания (ТЗ)**

- Формулируется содержательная постановка задачи, выбирается математический аппарат, строится модель предметной области, определяются подзадачи и выбираются или разрабатываются методы их решения
- Спецификация – формализованное описание функций и ограничений разрабатываемой программной системы
- Совокупность спецификаций формирует общую логическую модель проектируемой программной системы
- Здесь же формируется модель тестирования, согласуется методология тестирования

Основная задача – определение подробных спецификаций разрабатываемой программной системы

Различают процессы архитектурного, информационного (БД), функционального проектирования, проектирование взаимодействия (интерфейс) и выходных документов

Процесс проектирования программного наполнения содержит

Проектирование структуры

Определение компонентов и их взаимосвязи

Декомпозиции компонентов и построение иерархий

Проектирования компонентов

Результат проектирования – модель разрабатываемой программной системы с набором её спецификацией

Принято различать логическое и физическое проектирование

логическое проектирование, которое включает те проектные операции, которые непосредственно не зависят от имеющихся технических и программных средств, составляющих среду функционирования будущего программного продукта;

- физическое проектирование – привязку к конкретным техническим и программным средствам среды функционирования, т.е. учёт ограничений, определённых в спецификациях.

Реализация – процесс поэтапного создания моделей, программ и подпрограмм на выбранном языке программирования и их отладка

Программирование считают обычно главной деятельностью: Код не подвластен риторической силе и логике. На код нельзя воздействовать ученой степенью, общественным признанием и высоким окладом (К.Бек)

Но, по мнению специалистов, на отладку уходит – 90%-70% времени программирования. Поэтому экономить надо не на написании кода, а на отладке, создавая хорошо структурированный код.

Тестирование – отдельный этап ЖЦ. На этапе реализации оно присутствует как элемент отладки, определяющий симптомы ошибки. Его выполняет программист в формате «белого ящика»

Тестирование – процесс выполнения программы на наборе данных, для которого известен её результат или её поведение.

Тест – атомарная проверка одной функциональной части программы ***или*** набор данных, на котором производится тестирование

Цель тестирования – пропустить как можно меньше программных ошибок, среди них не должно быть наносящих существенный ущерб

На этом этапе тестированием занимается тестировщик (не автор!)

При тестировании могут возникать спорные ситуации. Для их разрешения в команде необходим ответственный за функциональность, способный скорректировать спецификацию

Результат – повышение надёжности программ и уточнение спецификации.

- **Сопровождение** – процесс создания и внедрения новых версий программного продукта. Занимает 40-80% времени ЖЦ

Виды сопровождения:

- Совершенствующее (настройка приобретённых программ, реализация отложенных функций, добавление новых возможностей) – 60%
- Корректирующее (исправление ошибок) – 17%
- Адаптирующее (перенос в новую среду) – 18%
- Превентивное – 5%

Технология внесения изменений

- Причина и суть изменений
- Проект изменений
- Модификация кода или структуры данных
- Тестирование
- Ввод в эксплуатацию

- Состав проекта по фазам

Начальная стадия – техническое задание: анализ требований, осуществимости, рисков. 1

- Проектирование – технический проект: проект БД, описание функционала, макет интерфейса, входные и выходные документы. 2
- Реализация – рабочий проект: описание программного изделия. 3
- Ввод в действие – готовое изделие и эксплуатационная документация. 4

- неформальная постановка задачи (технические требования): тема, ограничения;
- конкретизация функций системы;
- уточнение характера информации;
- выяснение контингента пользователей и их информационных потребностей;
- определение ограничений по представлению информации, объёмам, производительности, доступу к системе.

выполнено формирование требований к проекту;
создана начальная модель системы в терминах вариантов использования (готовность 10-20%);
сформирован начальный словарь;
сформирован начальный план;
выявлены дополнительные требования;
разработан план выполнения проекта по стадиям и итерациям;
реализовано необходимое количество прототипов.

- 2

проект базы данных: сущности, их ключи, связи, ограничения на атрибуты, их типы;

- проект функционального наполнения: множество функций с указанием их взаимозависимости;
- описание каждой функции;
- макет пользовательского интерфейса;
- выходные документы: сведения, которые естественно ожидать от системы.

- создана модель системы в терминах вариантов использования (готовность 80%);
- выявлены дополнительные требования;
- создана базовая архитектура;
- реализован работающий прототип системы;
- идентифицированы все серьёзные риски;
- разработан план выполнения проекта: определены итерации, критерии их оценки, даты начала и конца каждой итерации.

- 3

Итерационная разработка проекта;

- итерации зависимы – выполняются как последовательная цепочка;
- итерации независимы – могут выполняться параллельно.

Результат

- создано программное обеспечение;
- степень реализации достаточна для работы пользователя;
- описана реализация системы;
- есть результаты тестовых испытаний;
- разработаны руководства пользователя;
- разработано описание текущей реализации (версии продукта).
-

4

бета-тестирование;

- параллельное функционирование с существующими системами;
- конвертирование баз данных;
- оптимизация производительности;
- обучение пользователей и начальное сопровождение.

Результат – новая версия системы

26. Модели жизненного цикла. Последовательные модели

Последовательная модель предложена У.Ройсом в 1970 г.

Основная идея – процесс разработки ПО делится на определённые фазы, выполняемые строго последовательно

Классическая каскадная модель или модель водопада включает следующие этапы:

разработка требований

анализ и дизайн

реализация

валидация и тестирование

развёртывание системы

КАСКАДНАЯ

Модель предполагает переход на следующий этап после полного завершения работ предыдущего.

Каждый этап заканчивается результатом, который служит исходной позицией для следующего. Исходные требования к системе фиксируются в начале работы над проектом и не изменяются.

Принципиальные свойства каскадной модели:

- фиксация требований к системе до её сдачи;
- переход на следующую стадию проекта только после завершения предыдущей.

Достоинства:

- на каждом этапе готовится пакет документов, отвечающий критериям полноты и согласованности;
- порядок работы позволяет планировать сроки завершения работ и затраты.

Недостатки:

- система может не удовлетворять потребностям;
- позднее обнаружение проблем;
- быстро устаревающая документация;
- нарушение графика работ.

Пусть i-тый этап ЖЦ выполняется за время t_i, вся работа выполняется за время $T=\Sigma_{i=1}^{n}(t_i)$, где n — количество этапов разработки.

Пусть s —степень «полезности» системы с точки зрения заказчика. Время окончания i-го этапа обозначим $T_i = \Sigma_{j=1}^{i}(t_j)$, $i = 1\ldots n$, $T_n = T$.

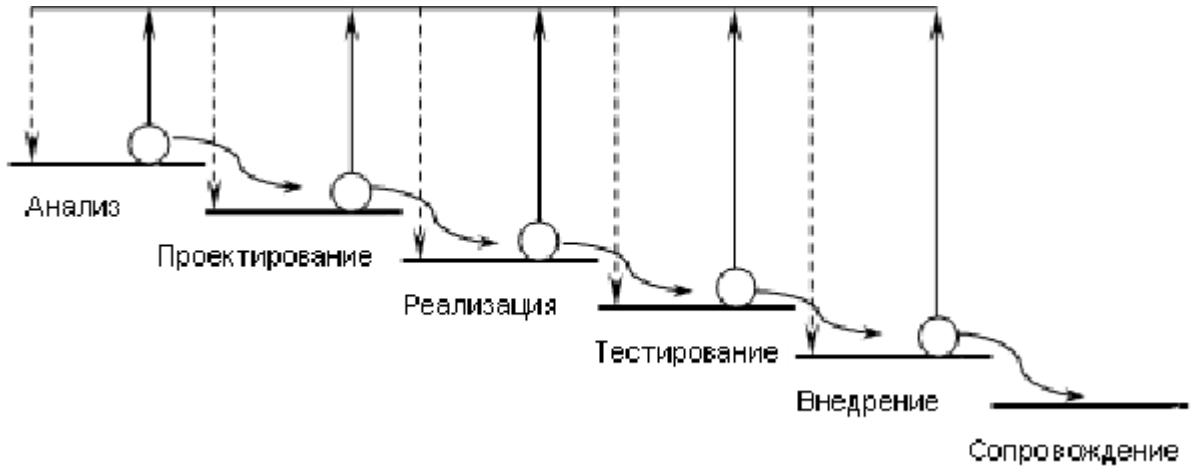
Если в точке t_0 предполагаемая польза — s_0, после первого этапа польза становится $s_1 < s_0$, дефект пользы на первом этапе $\Delta_1' = s_0 - s_1$. Для каждого i-го этапа $\Delta_i' = s_{i-1} - s$, и общий дефект пользы системы в результате ошибок разработки $\Delta' = \Sigma_{i=1}^{n}(\Delta_i')$.

В течение времени разработки изменяется и представление заказчика о пользе, которую могла бы принести система.

Обозначим приращение пользы на каждом этапе как Δ_i'', тогда общее приращение от дополнительных требований будет $\Delta'' = \Sigma_{i=1}^{n}(\Delta_i'')$.

ПОЭТАПНАЯ ИТЕРАЦИОННАЯ МОДЕЛЬ

- Недостатки каскадной модели привели к модели, в которой предполагается наличие циклов обратной связи между этапами. Для этого используется механизм межэтапных корректировок
- Каждый этап работы заканчивается обсуждением результата в присутствии заказчика. Как только выявляется отклонение проекта от ожидаемого результата, проект дорабатывается, на что выделяется дополнительное время
- Из-за внешнего сходства с каскадной её иногда называют каскадно-возвратной, но это название противоречиво: каскад не возвращается



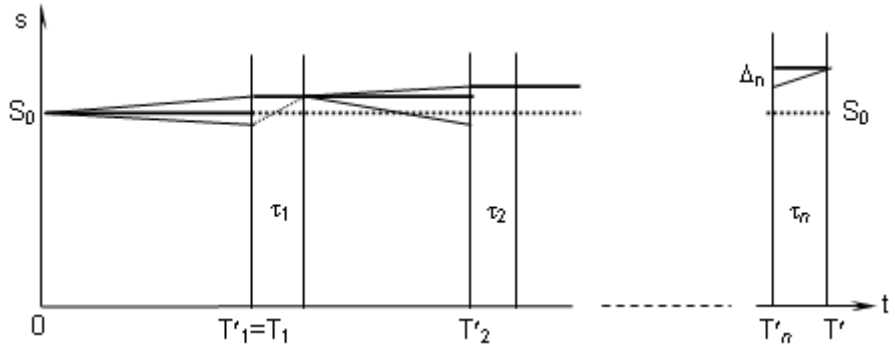
Используем те же обозначения, что и для каскадной модели, добавим время на процедуры перехода от этапа к этапу.

Во время обсуждения результатов этапа выявляется дефект Δ_i' и дополнительные пожелания Δ_i''.

Выделяется время τ_i на ликвидацию дефекта, после его ликвидации выполняют следующий этап.

К моменту системы общее несоответствие составляет $\Delta_n = \Delta_n' + \Delta_n''$, оно ликвидируется в период опытной эксплуатации.

Общее время с учётом доработок $T' = T + \Sigma_{i=1}^{n}(\tau_i)$.



Достоинство:

результат обычно соответствует ожиданиям заказчика, и риск провала проекта невелик.

Недостаток:

из-за попыток сделать лучше, длительность любого этапа может растянуться на неопределённое время. Таким образом, главный *недостаток* модели — трудность планирования времени разработки и затрат..

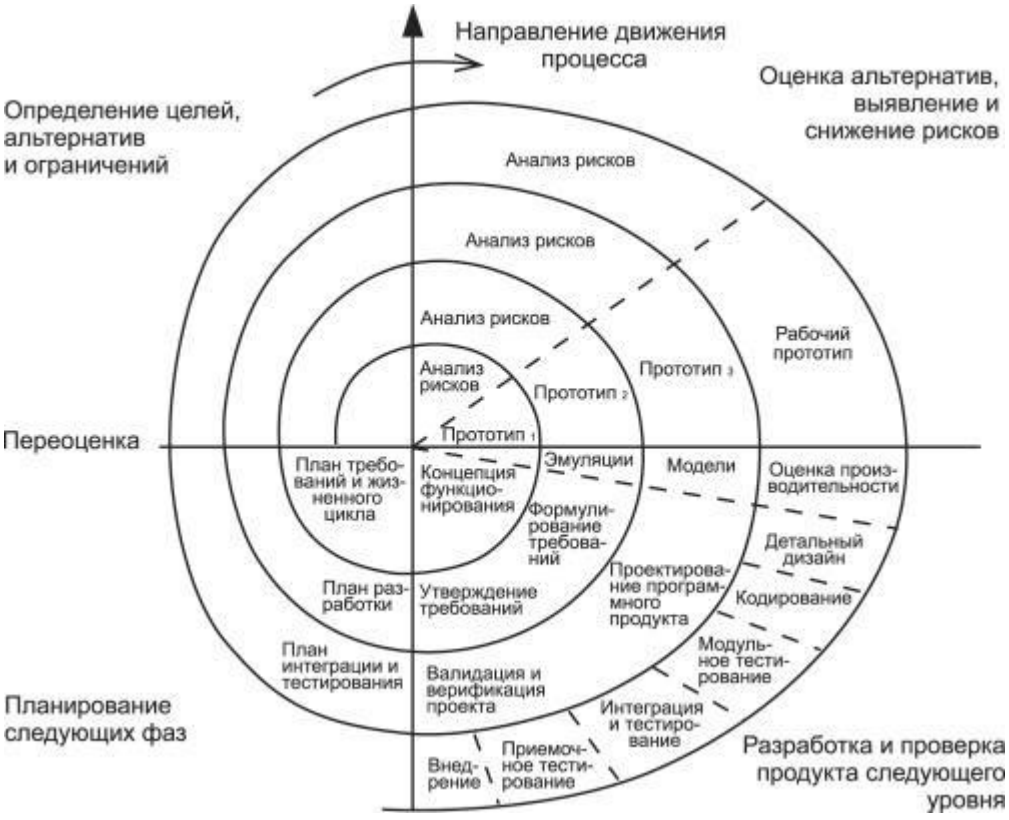
- Последовательным развитием каскадной и водопадной моделей была v

Вариант поэтапной модели – это V-образная модель

- Применение V-образной модели позволило существенно повысить качество разработки ПО за счёт ориентации на тестирование. Она позволяет решить задачу соответствия созданного продукта выдвигаемым требованиям за счет процедур верификации и аттестации на ранних этапах разработки
- V-образная модель – модификация каскадной, обладает многими её недостатками, например, слабо приспособлена к возможным изменениям требований заказчика
- последовательные модели хорошо работают в проектах, где требования могут быть заранее чётко определены и зафиксированы

27. *Модели жизненного цикла. Спиральная модели*

- Спиральная модель, предложенная Б.Боэмом в 1988 г., объединила итерационный процесс проектирования и классический каскадный подход на основе идеи создания прототипов
- Спиральная модель ориентирована на проектирование, именно в этой части последовательно выполняется несколько итераций проектирования на основе создания прототипов
- Разработка ПО происходит на последнем витке спирали по классической каскадной модели



реальном прикладном программировании часто приходится сталкиваться с «нетерпеливым заказчиком», которому нужно начать работу с системой, пусть с ограниченными возможностями.

Предполагается, что мощность системы можно нарастить параллельно с эксплуатацией первой версии.

Такая ситуация породила версионную модель разработки, в которой система представляется в виде совокупности функций с определённым приоритетом ввода в действие.

Вначале тщательно проектируется и реализуется очень простая, но полезная версия системы.

Как только первая версия сдана, начинается работа над второй, включающей функции второго приоритета.

Аналогично разрабатываются и следующие версии.

Модель такого типа относится к спиральным, у неё подобное графическое представление. Разработка представляется в полярных координатах, где угол символизирует время, радиус — объём работы.

Полный оборот соответствует разработке и вводу в эксплуатацию очередной версии.

Спиральная модель позволяет уменьшить неопределённость поэтапной модели.

Основная идея — разработка системы короткими, но законченными версиями, каждая из которых не удовлетворяет полностью требованиям заказчика, но последовательность версий итеративно приближается к ним.

В отличие от инкрементной, спиральная модель предполагает ввод в эксплуатацию не новой подсистемы, а очередной версии системы целиком.

Технология работы по спиральной модели допускает распараллеливание.

Достоинства:

- раннее получение результата;
- быстрое получение реакции пользователя;
- предсказуемость поведения системы.

Недостатки:

- сложность планирования;
- сложность адаптации пользователя к множеству версий;
- напряжённость работы разработчиков;
- сложность смены (сопровождения) множества версий.

28. *Модели жизненного цикла. Итеративные модели*

В крупных системах планировать время выполнения отдельных этапов трудно.

Планируемое время разработки i-го этапа t_i велико, расхождение $\Delta_i = \Delta_i' + \Delta_i''$ успевает стать заметным, и время τ_i становится сравнимым с t_i .

Возникает риск «вечной разработки».

Подобный эффект возникает и в случае большой неопределённости (для компенсации риска требуется увеличить время t_i) или с высокой изменчивостью, когда за время выполнения этапа значительно увеличивается расхождение Δ_i''

Б.Боэм: эволюционная модель – «модель, стадии которой состоят из расширяющихся приращений оперативного программного обеспечения, с направлением эволюции, определяемым опытом работы».

В отличие от предыдущих, инкрементная модель даёт возможность заказчику получать пользу даже от не полностью готового продукта, а разработчику — использовать обратную связь для улучшения качество следующих подсистем.

Подсистема сдаётся с полной документацией и с возможностью сопровождения, она эволюционирует по законам обычной системы.

Если она пришла в состояние, когда её сопровождение стало нерентабельным, она заменяется новой в рамках существующей системы

- Модель итеративной (инкрементной) разработки – эволюционное развитие каскадной модели
- Процесс разработки состоит из последовательности повторяющихся итераций, каждая из которых представляет собой полноценный проект

- В результате каждой итерации продукт получает улучшения своей функциональности или новую компоненту. После завершения последней итерации проект получает полный набор функции и требований, определенный границами проекта
- Исходя из требований заказчика, команда проекта выбирает результат итерации:
  - промышленную систему с ограниченными функциями
  - архитектурные или функциональные прототипы

Стратегия реализации эволюционных моделей Т. Джилба:

- Представь нечто реальному пользователю.
- Согласуй добавленное значение с пользователем по всем критичным направлениям.
- Откорректируй проект и требования на основании того, что существует на самом деле.

Разработка:

1. Выполняется этап, на котором определяются системные цели, формулируются требования, определяются ресурсы.
2. На основе требований разрабатывается архитектура, в которой система разбита на относительно независимые замкнутые подсистемы, определяются приоритеты, в соответствии с ними планируется разработка.
3. Разрабатывается каждая подсистема.

Плюсы и минусы

инкрементная модель, за счёт раннего ввода подсистем в эксплуатацию, даёт возможность заказчику лучше осознать свои потребности и своевременно внести корректировки в требования к не разработанным подсистемам.

Поскольку каждая подсистема проще системы в целом, разработчику проще планировать необходимые ресурсы. И это несомненное *достоинство* модели.

Недостаток – трудность достижения единого стиля системы. Обычно успешно реализуют интерфейсы между подсистемами и общую направленность проекта. Унифицировать в рамках системы характер взаимодействия с пользователем обычно не удаётся

29. Модели жизненного цикла. Жизненный цикл RUP.

- Во второй половине 1990-х годов в компании Rational Software была создана **итеративно-инкрементная модель RUP.**
- RUP описывает общий абстрактный процесс, на основе которого команда проекта должна создать специализированный адаптированный процесс, который ориентирован на внутренние потребности
- Основные характеристики процессов RUP
  - **Разработка требований.** На основании прецедентов использования описываются требования к системе, полный набор которых образует модель прецедентов. Прецедент использования – описание сценария взаимодействия пользователя с системой, который полностью описывает пользовательскую задачу. Согласно методологии RUP, функциональные требования (ФТ) должны быть представлены в виде прецедентов использования

- Основные характеристики процессов RUP
  - **Итеративная разработка.** В методологии RUP проект состоит из последовательности итераций с продолжительностью от двух до шести недель. Прецедент использования – основная единица планирования итераций. Перед началом итерации определяется список прецедентов для реализации в ней. В ходе реализации проекта могут вноситься необходимые изменения в требования, архитектуру, реализацию и проектные решения.
  - **Архитектура RUP.** RUP – ориентированная на архитектуру методология. В ней реализация и тестирование архитектуры начинается на ранних этапах. В RUP используется понятие исполняемой архитектуры, которая позволяет в первую очередь реализовать архитектурно значимые прецеденты

Дисциплина — элемент технологического процесса, последовательности действий, который приводит к получению значимого результата

1. Построение бизнес-моделей.
2. Определение требований.

3. Анализ и проектирование.
4. Реализация.
5. Тестирование.
6. Развёртывание.
7. Управление конфигурацией и изменениями.
8. Управление проектом.
9. Создание инфраструктуры.

30. *Модели жизненного цикла. Гибкие методологии, eXtreme Programming*

В начале 2000-х годов группой экспертов по легковесным методологиям были сформулированы принципы гибкой разработки ПО - **Agile Manifesto**

- Общие особенности гибких методологий:
  - ориентированность на разработчиков и заказчиков. Основная идея – собрать в проектной команде профессионалов, которые определят успешность проекта в большей степени, чем процессы и технологии
  - вместо формальных спецификаций использовать устные обсуждения, которые служат главным способом коммуникации внутри проектной команды
  - итеративная разработка с минимальной длительностью итерации. В результате каждой итерации происходит выпуск полноценной работающей версии продукта
  - В начале проекта команда не пытается зафиксировать требования и затем следовать жёстко определённому плану. Изменения могут быть сделаны на любом этапе проекта.

- Гибкая методология разработки ПО созданная К.Беком в 1996 г. в результате работы над проектом в компании Chrysler.
- Методология наследует общие принципы гибких методологий, достигая при их помощи инженерных практик:
  - в команде проекта постоянно присутствует заказчик, обладающий детальной информацией о необходимой функциональности, который определяет приоритеты задач и оценивает качество создаваемой системы
  - пользовательские истории и приёмочные тесты служат средством спецификации требований, по сути это короткие неформальные описания прецедентов использования системы
  - весь создаваемый код автоматически покрывается тестами

- Продолжаем:
  - максимально простая архитектура системы. Методология XP не рекомендует проектирование в расчёте на будущее системы. Текущая архитектура должна поддерживать существующую функциональность.
  - изменение архитектуры требует постоянной переработки кода. XP поощряет коллективное владение кодом.
  - сделанные разработчиками изменения после автоматического тестирования попадают в репозиторий, этап интеграции отсутствует
  - парное программирование и сорокачасовая рабочая неделя

Базовые принципы • быстрая обратная связь; • приемлемая простота; • постепенное изменение; • приемлемое изменение; • качественная работа.
Виды деятельности • кодирование; • тестирование; • слушание (внимание); • проектирование.

Двенадцать правил

1. Краткосрочное планирование.
2. Небольшие версии.
3. Метафора.
4. Простой дизайн.
5. Тестирование.

6. Переработка кода.
7. Программирование парами.
8. Коллективное владение.
9. Постоянная интеграция.
10. 40-часовая рабочая неделя.
11. Заказчик на месте разработки.
12. Стандарты кодирования.

Механизмы обеспечения качества Уделять внимание только тем вопросам,λ которые в данный момент действительно важны; добиваться получения заказчиками иλ руководителями оперативной информации о результатах работы и о процессах.

31. Модели жизненного цикла. Гибкие методологии, SCRUM

Цель — максимально быстрая поставка продукта на рынок. Для этого используется понятие «минимально ценный продукт» (minimum viable product, MVP)

Аргументы • рынок должен получить данный продукт до того, как тот успеет устареть; • до выпуска продукта на рынок есть лишь смутные предположения о потребностях пользователя, и только выпуск продукта на рынок даст возможность их уточнить. Проблемы решаются качественным анализом, но в Scram его нет

Жизненный цикл разработки разбивается на спринты, каждый из которых длится от одной недели до месяца. Цель спринта — выпуск работающей версии продукта, реализующей какую-то функциональность будущей системы.

Качество • Время спринта оценивается произвольно: нет стабильно качественной отладки; • нет полноценного анализа: ошибки в функциональности; • накапливается технический долг. Вывод: для производства качественных продуктов методология не пригодна.

- **Методология SCRUM** представляет эмпирический подход к разработке ПО. Данный процесс адаптивен и основан на повторяющихся циклах.
- **Основные принципы Scrum**:
  - постоянное взаимодействие с заказчиком важнее договорных обязательств
  - гибкий и индивидуальный подход важнее строгих процессов и методов
  - следованию плану может быть изменено при необходимости
  - работающее ПО важнее проектной документации

- Scrum — это каркас разработки. В его основе лежат короткие ежедневные встречи и цикличные «спринты»
- Спринты, каждый из которых длится от одной недели до месяца, составляют жизненный цикл разработки.
- Спринт – итерация в SCRUM, в ходе которой создается инкремент бизнес-продукта
  - Цель спринта — выпуск работающей версии продукта, реализующей какую-то функциональность будущей системы