

## 1. История и основные понятия программной инженерии [3]

Программная инженерия есть применение определенного систематического измеримого подхода при разработке, эксплуатации и поддержке программного обеспечения.

Термин программное обеспечение ввел в 1958 году всемирно известный статистик Джон Тьюкей. Термин программная инженерия впервые появился в названии конференции НАТО, состоявшейся в Германии в 1968 году и посвященной так называемому кризису программного обеспечения. С 1990 по 1995 год велась работа над международным стандартом, который должен был дать единое представление о процессах разработки программного обеспечения. В 2004 году в отрасли был создан основополагающий труд «Руководство к своду знаний по программной инженерии» (SWEBOOK), в котором были собраны основные теоретические и практические знания, накопленные в этой отрасли.

**Программирование** – процесс отображения определенного множества целей на множество машинных команд и данных, интерпретация которых на компьютере или вычислительном комплексе обеспечивает достижение поставленных целей.

Цели могут быть любые: воспроизведение звука в динамике ПК, расчет траектории полета космического аппарата на Марс, печать годового балансового отчета и т.д. Важно то, что они должны быть определены. Это звучит банально, но сколько бы раз об этом не твердили ранее, по-прежнему, приходится сталкиваться с программными проектами, в которых отсутствуют какие-либо определенные цели. Это отображение может быть очень простым, например, перфорирование машинных команд и данных на перфокартах. А может быть многоступенчатым и очень сложным.

**Профессиональное программирование** – деятельность, направленная на получение доходов при помощи программирования. Принципиальным отличием от просто программирования является то, что имеется или, по крайней мере, предполагается некоторый потребитель, который готов платить за использование программного продукта. Отсюда следует важный вывод о том, что профессиональное производство программ это всегда коллективная деятельность, в которой участвуют минимум два человека: программист и потребитель.

**Профессиональный программист** – человек, который занимается профессиональным программированием. Профессионального программиста следует отличать от профессионала (мастера в программировании). Разброс профессионального мастерства в программировании достаточно широк и далеко не каждый, кто зарабатывает на жизнь программированием, является мастером.

**Программный продукт** – совокупность программ и сопроводительной документации по их установке, настройке, использованию и доработке. Согласно стандарту жизненный цикл программы, программной системы, программного продукта включает в себя разработку, развертывание, поддержку и сопровождение. Сопровождение включает в себя устранение критических неисправностей в системе и реализуется часто не как проект, а как процессная деятельность. Поддержка заключается в разработке новой функциональности, переработке уже существующей функциональности, в связи с изменением требований, и улучшением продукта, а также устранение некритических замечаний к программному обеспечению, выявленных при его эксплуатации. Жизненный цикл программного продукта завершается выводом продукта из эксплуатации и снятием его с поддержки и сопровождения.

**Модель процесса разработки программного обеспечения** – формализованное представление процесса разработки программного обеспечения.

## 2. Допущения и ограничения [3]

Исходные допущения и ограничения тесно связаны с управлением рисками. В разработке программного обеспечения зачастую риски формулируют в виде допущений. Например, оценивая проект разработки и внедрения по схеме с фиксированной ценой, в

допущения записывают предположение о том, что стоимость лицензий на стороннее ПО не изменится до завершения проекта.

Ограничения, как правило, сокращают возможности проектной команды в выборе решений и могут содержать:

1. Специфические нормативные требования. Например, обязательная сертификация продукта, услуги на соответствие определенным стандартам.
2. Специфические технические требования. Например, разработка под заданную программно-аппаратную платформу.
3. Специфические требования к защите информации.
4. Требования к системе, которые могут ожидать заказчики по умолчанию, но которые не включаются в рамки данного проекта. Например, в данный раздел может быть включен пункт о том, что разработка программного интерфейса для будущей интеграции с другими системами заказчика не входит в задачи данного проекта.

/\* GPT \*/

В программной инженерии допущения и ограничения играют критически важную роль, определяя рамки и границы, в которых осуществляется разработка программного обеспечения. Допущения - это утверждения, которые считаются истинными без доказательств на момент начала проекта. Они могут касаться различных аспектов проекта, таких как доступность ресурсов, функциональность системы, внешние зависимости и многое другое. Например, разработчики могут предполагать, что определённые библиотеки или фреймворки будут доступны на всех целевых платформах, или что пользователи будут иметь определённый уровень знаний и навыков. Допущения часто служат основой для планирования и принятия решений, но они также могут нести риски, так как могут оказаться неверными в процессе реализации проекта.

Ограничения, с другой стороны, представляют собой реальные ограничения и барьеры, с которыми команда сталкивается в процессе разработки. Они могут быть техническими, организационными, экономическими или нормативными. Технические ограничения могут включать ограничения по производительности, объёму памяти, пропускной способности сети или совместимости с определёнными системами и устройствами. Организационные ограничения могут касаться наличия квалифицированных специалистов, сроков выполнения проекта или структуры команды. Экономические ограничения часто связаны с бюджетом проекта, который ограничивает возможности использования определённых технологий или сервисов. Нормативные ограничения могут включать соблюдение стандартов безопасности, конфиденциальности данных, а также законодательных требований различных стран и регионов.

Допущения и ограничения взаимосвязаны и могут оказывать влияние друг на друга. Например, если одно из допущений оказывается неверным, это может привести к необходимости пересмотра ограничений проекта. Также, если ограничения изменяются, это может потребовать корректировки допущений и, соответственно, плана проекта. Управление допущениями и ограничениями требует постоянного мониторинга и анализа. Команда должна быть готова оперативно реагировать на изменения, обновляя документацию и внося корректировки в процесс разработки. Важно также вовлекать всех заинтересованных лиц в процесс определения и пересмотра допущений и ограничений, чтобы обеспечить их понимание и согласие.

Таким образом, допущения и ограничения являются фундаментальными аспектами, которые формируют контекст разработки программного обеспечения. Они помогают определить реалистичные цели, установить приоритеты и управлять рисками, обеспечивая более предсказуемое и контролируемое выполнение проекта.

### **3. Отличия программной инженерии от других отраслей [3]**

Standish Group, анализируя ежегодно успешность работы крупных корпораций и выполнения ими крупных проектов, пришла к выводу, что только треть из них завершается успешно и в срок. Остальные же проекты либо завершаются с опозданием, либо превышают запланированные расходы, либо вовсе проваливаются.

То, что производят программисты нематериально – это коллективные мысли и идеи, выраженные на языке программирования. Производство программного обеспечения суперсложная интеллектуальная деятельность. Время вхождения в профессию сильно меньше, чем в других инженерных дисциплинах. Разрабатывать программное обеспечение не сложнее, чем делать ракеты.

Творчество – это интеллектуальная деятельность человека, законы которой нам неизвестны. Если бы мы знали законы творчества, то и картины, и стихи, и музыку, и программы уже давно бы создавали компьютеры. Творческое начало это то, что роднит программирование с наукой и искусством. Творчество в программировании начинается с определения целей программы и заканчивается только тогда, когда в ее коде, написанном на каком-либо языке программирования, поставлена последняя точка. Попытки разделять программистов на творческую элиту, архитекторов и проектировщиков, и нетворческих программистов-кодеров не имеют под собой объективных оснований. Даже если алгоритм программы строго определен математически, два разных программиста его закодируют по-разному, и полученная программа будет иметь разные потребительские качества.

Программирование – не искусство, в том смысле, что оно не является творческим отражением и воспроизведением действительности в художественных образах. Об искусстве в программировании можно и должно говорить только в смысле умения, мастерства, знания дела, как и в любой другой профессии. И как в любой другой профессии программистское мастерство может доставлять истинное эстетическое наслаждение, но только для людей, причастных к этой профессии.

Программирование – это не наука. Наработки математиков в области логики, теории информации, численных методов, реляционной алгебры, теории графов и некоторых других дисциплинах на долю процента не покрывают сложность программистских задач. В программировании нет системы знаний о закономерностях создания программ. Даже выдающиеся программисты не возьмут на себя смелость утверждать об архитектуре новой программной системы то, что она будет успешной. Хотя в программировании уже накоплен определенный опыт провалов, который может позволить

искусственному программисту увидеть в архитектуре новой системы антипаттерны - источники серьезных будущих проблем. Но не более того. Программист работает с абстракциями, но ему приходится держать в голове гораздо больше, чем любому ученому. Абстракции сопутствуют программисту на всех уровнях разработки программы от описания ее целей до машинного кода. И этих уровней могут быть большое количество. И на каждом уровне абстракций их деталей становится все больше и больше. Дополнительно к абстрактному мышлению, программист должен обладать сильно выраженным системным мышлением. Еще одной сложностью является то, что все эти абстракции и взаимосвязи между ними изменяются во времени, и программист должен учитывать эту динамику. Кроме того, программист должен обладать усидчивостью, сосредоточенностью и упорством. Проработка должна быть абсолютно точной и не должна содержать ни одной ошибки, неправильного, лишнего или отсутствующего символа исходного кода (а это порой миллионы строк).

#### **4. Управление содержанием проекта**

Человечество пока не придумало ничего более эффективного для решения сложной задачи, чем анализ и ее декомпозиция на более простые подзадачи, которые, в свою очередь, могут быть разделены на еще более простые подзадачи и так далее. Получается некоторая иерархическая структура, дерево, в корне которого находится проект, а на листьях элементарные задачи или работы, которые надо выполнить, чтобы завершить проект в условиях заданных ограничений.

Иерархическая структура работ (Work Breakdown Structure, WBS) – ориентированная на результат иерархическая декомпозиция работ, выполняемых командой проекта для достижения целей проекта и необходимых результатов. С ее помощью структурируется и определяется все содержание проекта. Каждый следующий уровень иерархии отражает более детальное определение элементов проекта. Основой для разработки иерархической структуры работ служит концепция проекта, которая определяет продукты проекта и их основные характеристики. Иерархическая структура работ обеспечивает выявление всех работ, необходимых для достижения целей проекта. Многие проекты проваливаются не от того, что у них нет плана, а от того что в этом плане забыты важные работы, например, тестирование и исправление ошибок, и продукты проекта, например, пользовательская документация. Поэтому, если иерархическая структура работ составлена корректно, то любая работа, которая в нее не вошла не может считаться работой по проекту.

Иерархическая структура работ является одним из основных инструментов в механизме управления проектом, с помощью которого измеряется степень достижения результатов проекта. Важнейшая ее функция – это обеспечить консистентное представление всех участников проекта относительно того, как будет делаться проект. В последующем базовый план будет служить ориентиром для сравнения с текущим исполнением проекта

и выявления отклонений для целей управления.

Сразу, как только удалось стабилизировать и согласовать иерархическую структуру работ, необходимо разработать план управления содержанием проекта. Для этого следует:

1. Определить источники запросов на изменение.
2. Установить порядок анализа, оценки и утверждения или отклонения изменения содержания.
3. Определить порядок документирования изменений содержания.
4. Определить порядок информирования об изменении содержания. Первая задача, которую необходимо решить при анализе запроса на изменения - выявить объекты изменений: требования,

архитектура, структуры данных, исходные коды, сценарии тестирования, пользовательская документация, проч. Затем требуется спроектировать и детально описать изменения во всех выявленных объектах. И наконец, следует оценить затраты на внесение изменений, тестирование изменений и регрессионное тестирование продукта и их влияние на сроки проекта. Эта работа, которая потребует затрат рабочего времени и порой значительных разных специалистов: аналитиков, проектировщиков, разработчиков, тестировщиков и менеджера проекта. Поэтому эта работа должна обязательно быть учтена в плане.

## 5. Уточнение содержания и состава работ [4] (GPT)

Уточнение содержания и состава работ в программной инженерии является важным процессом, направленным на детальную проработку всех аспектов проекта, что помогает обеспечить его успешное выполнение. Этот процесс начинается с определения общей концепции проекта и заканчивается формированием конкретного набора задач, необходимых для достижения поставленных целей.

На начальном этапе уточнения содержания проводится анализ требований. Этот анализ включает в себя взаимодействие с заказчиками и заинтересованными сторонами для сбора их ожиданий и потребностей. Важно понять, что именно требуется от программного продукта, какие функции и характеристики он должен иметь, а также какие ограничения и условия должны быть соблюдены. Результатом этого этапа является документ требований, который описывает все аспекты будущей системы и служит основой для дальнейшей работы.

После того как требования собраны и задокументированы, начинается этап их детализации. На этом этапе требования преобразуются в конкретные задачи и действия, которые необходимо выполнить. Этот процесс может включать в себя разработку функциональных и нефункциональных требований, создание пользовательских сценариев и кейсов использования, а также определение технических характеристик и спецификаций. Все эти элементы помогают более четко понять, что именно должно быть разработано и как это должно работать.

Одним из ключевых инструментов на этапе уточнения содержания и состава работ является структура декомпозиции работ (SDW). SDW представляет собой иерархическую модель, которая разбивает проект на более мелкие и управляемые компоненты. Этот процесс декомпозиции позволяет определить все задачи и этапы проекта, а также их взаимосвязи и последовательность выполнения. Каждая задача в SDW описывается детально, что помогает более точно оценивать необходимые ресурсы, сроки и затраты.

Детализация задач и работ также включает в себя определение критериев приемки и качества. Эти критерии помогают установить, как будет оцениваться успешность выполнения каждой задачи и всего проекта в целом. Критерии приемки могут включать в себя функциональные требования, такие как наличие определенных функций и возможностей, а также нефункциональные требования, такие как производительность, надежность и безопасность системы. Критерии качества могут включать в себя стандарты кодирования, тестирования и документирования.

Важным аспектом уточнения содержания и состава работ является управление изменениями. В процессе выполнения проекта требования могут изменяться, и необходимо иметь механизмы для управления этими изменениями. Это включает в себя процедуры для анализа и оценки изменений, их утверждения и интеграции в общий план проекта. Управление изменениями помогает избежать неожиданностей и поддерживать контроль над проектом, обеспечивая его выполнение в рамках установленных сроков и бюджета.

Еще один важный аспект уточнения содержания и состава работ – это постоянная коммуникация и взаимодействие с заинтересованными сторонами. Регулярные встречи, обзоры и проверки помогают

убедиться, что все стороны понимают цели и задачи проекта одинаково, а также позволяют своевременно выявлять и устранять любые проблемы или недоразумения.

Завершающим этапом уточнения содержания и состава работ является документирование всех аспектов проекта. Это включает в себя создание технической документации, планов проекта, графиков и отчетов. Документация помогает поддерживать ясность и прозрачность проекта, а также служит основой для последующего анализа и оценки. Документирование всех этапов и результатов проекта также позволяет накопить знания и опыт, которые могут быть использованы в будущих проектах.

Таким образом, уточнение содержания и состава работ в программной инженерии является комплексным и многоэтапным процессом, который требует тщательного планирования, детального анализа и постоянной коммуникации. Этот процесс помогает определить все необходимые задачи и действия, установить критерии приемки и качества, а также обеспечить управление изменениями и документацию. Эффективное уточнение содержания и состава работ позволяет достичь поставленных целей и обеспечить успешное выполнение проекта.

## **6. Эволюция подходов к управлению программными проектами [4] (GPT)**

Эволюция подходов к управлению программными проектами в программной инженерии отражает значительные изменения и улучшения в методах, процессах и инструментах, используемых для планирования, исполнения и контроля проектов. Исторически, эти подходы развивались в ответ на растущую сложность программных систем, изменяющиеся потребности бизнеса и технические инновации.

На ранних этапах развития программной инженерии применялись традиционные методы управления проектами, заимствованные из строительной и производственной отраслей. Эти методы, такие как каскадная модель (waterfall model), предполагали последовательное выполнение этапов проекта: от сбора и анализа требований к проектированию, реализации, тестированию и сопровождению. Каждый этап начинался только после завершения предыдущего, что обеспечивало структурированный и предсказуемый процесс. Однако, каскадная модель имела свои ограничения, особенно в условиях быстрого изменения требований и необходимости гибкости. Её жесткость и недостаточная адаптивность часто приводили к тому, что к моменту завершения проекта требования заказчика уже успевали измениться, что создавало значительные проблемы.

С развитием информационных технологий и увеличением сложности программных систем возникла необходимость в более гибких и адаптивных подходах. В 1990-х годах начали активно развиваться гибкие методологии, такие как Agile. Основные принципы Agile, изложенные в Манифесте Agile, включают в себя приоритет взаимодействия людей над процессами и инструментами, работающий продукт над исчерпывающей документацией, сотрудничество с заказчиком над контрактными переговорами и готовность к изменениям над следованием первоначальному плану. Agile подходы, такие как Scrum и Kanban, позволяют командам быстро адаптироваться к изменениям, поддерживать тесное взаимодействие с заказчиком и регулярно поставлять работающий софт. Эти методологии основываются на итеративном и инкрементальном подходе, что позволяет гибко реагировать на изменения и постепенно совершенствовать продукт.

Вместе с Agile, в конце 1990-х и начале 2000-х годов получили развитие и другие гибкие методологии, такие как экстремальное программирование (XP) и Lean. Эти подходы акцентируют внимание на улучшении качества кода, автоматизации тестирования, непрерывной интеграции и быстрой обратной связи. В экстремальном программировании, например, широко применяются такие практики, как парное программирование, частые релизы и постоянное улучшение кода, что помогает снизить риски и повысить качество конечного продукта.

В последние годы на управление программными проектами значительное влияние оказали принципы DevOps, которые интегрируют процессы разработки и эксплуатации. DevOps направлен на

улучшение сотрудничества между разработчиками и операционными командами, автоматизацию процессов и непрерывное развертывание (Continuous Deployment). Применение DevOps позволяет значительно ускорить выпуск новых версий программного обеспечения, улучшить качество и стабильность продуктов, а также быстрее реагировать на изменения и инциденты. Это стало возможным благодаря использованию таких инструментов, как контейнеризация (например, Docker), оркестрация контейнеров (Kubernetes), системы управления конфигурацией (Ansible, Puppet, Chef) и CI/CD-пайплайны (Jenkins, GitLab CI/CD).

Современные подходы к управлению программными проектами также всё больше опираются на данные и аналитику. Использование методов и инструментов для анализа больших данных, машинного обучения и искусственного интеллекта позволяет проектным командам более точно прогнозировать сроки и затраты, выявлять риски и оптимизировать ресурсы. Например, применение предиктивной аналитики помогает выявлять потенциальные проблемы на ранних стадиях и принимать проактивные меры по их устранению.

Эволюция подходов к управлению программными проектами также включает интеграцию лучших практик из различных методологий. Например, гибридные подходы комбинируют элементы Agile и традиционных методов, чтобы наилучшим образом соответствовать специфике конкретного проекта. Такие подходы позволяют сохранять гибкость и адаптивность Agile, одновременно используя структурированные и предсказуемые элементы традиционных моделей.

Таким образом, эволюция подходов к управлению программными проектами отражает переход от жёстких и предсказуемых моделей к более гибким, адаптивным и интегрированным методологиям. Этот переход был обусловлен потребностью в быстрой реакции на изменения, улучшении качества и сокращении времени на разработку. Современные подходы, такие как Agile, DevOps и гибридные методологии, обеспечивают более тесное взаимодействие с заказчиком, быструю обратную связь и непрерывное улучшение продукта, что помогает успешно справляться с вызовами современного мира программной инженерии.

## **7. Организация проектной команды [3]**

Командная работа играет важнейшую роль в разработке отличного программного обеспечения и что великие команды олицетворяют понятие «мы», а не «я». Нет ничего ценнее, чем работать вместе с заинтересованными участниками команды над созданием действительно стоящего продукта.

Несмотря на общность ценностей, формулы идеальной команды не существует. Одни команды внедряют чистый Agile, другие используют MSF. Согласно классической теории необходимо, чтобы участники находились в непосредственной близости друг с другом, но иногда реалии бизнеса требуют географического распределения команды. Большинство команд имеют все необходимые навыки, но иногда для выполнения конкретной работы требуется помощь узких специалистов. Создав команду необходимо помнить, что команды похожи на людей: для их роста нужно время. По мере своего развития команда проходит через четыре основные стадии (Рис. 12).



Рис. 12. – Этапы формирования команды

**На этапе формирования** команды приходится иметь дело с совокупностью отдельных личностей. При этом происходят следующие процессы между участниками команды:

1. Знакомство между членами команды.
2. Оценивание компетентности и полномочий лидера.
3. Появление сомнений в навыках и перспективах работы в команде.
4. Наблюдение за коллегами.
5. Попытка найти «свое» место в команде.
6. Совершенствование знаний и умений.

По причине защитного механизма в поведение у человека в этот момент в команде может наблюдаться как прогресс, так и его отсутствие. На данном этапе члены команды пытаются много общаться, но могут не полностью понимать друг друга.

**На этапе притирания** члены команды осознают, что для достижения целей им предстоит пройти длинный путь. Характерные черты действий команды:

1. Появление чувств неудовлетворенности, раздражения, разочарования и неуверенности в собственных силах.
2. Соперничество «за корону» в важности для команды.
3. Соперничество за внимание со стороны лидера.
4. Недоверие.
5. Раздражительная реакция на задачи.
6. Сомнения в полезности для общего успеха.

Лидер на данном этапе должен выполнять функцию направления участников команды и команды в целом. Лидер команды должен обратить серьезное внимание на возможное чувство недовольства в группе, не принимая его на свой счет и спокойно обсуждая подобные ситуации, не занимая оборонительную позицию.



**На этапе стабилизации** участники команды постепенно привыкают друг к другу, появляется доверие и уважение. Характерные черты действий членов команды:

1. Общение и взаимодействие протекают более эффективно
2. Возникает чувство сплоченности и командный дух
3. Общение между членами команды становятся конструктивными
4. Работа становится на первое место, конфликты сходят на нет.

Стиль лидерства на данном этапе – поддерживающий. Лидер обязан максимально сконцентрироваться на поддержке и оценке усилий каждого участника, совершенствовании рабочей атмосферы, коммуникациях, сотрудничестве и дальнейшем улучшении отношений в группе.

**На этапе расцвета** видна настоящая эффективность команды с творческой рабочей атмосферой. Характерные черты действий команды:

1. Поддержание внутрикомандного климата.
2. Развитие механизмов самоуправления.
3. Поддержка связей между участниками команды.

Члены команды используют сильные и слабые стороны друг друга и чувствуют вовлеченность в командную работу. На этом этапе стиль лидерства - консультативный, сопряженный с непрерывным процессом поиском улучшения взаимодействия.

Если руководитель (лидер) не будет прилагать усилий, то команда, рано или поздно, начнет терять свою эффективность и погружаться в состояние застоя и стагнации. Задача менеджера: поддерживать требуемый уровень мотивации, быть лидером, искать новые пути и открывать новые возможности. Четыре стадии развития команды должны циклически повторяться, чтобы обеспечить непрерывный рост производительности

---

---

**Айшан**

## 8. Ресурсы [2]

В управлении программными проектами ресурсы играют ключевую роль, так как от их эффективного использования зависит успех проекта. Ресурсы можно классифицировать на несколько категорий: человеческие, материальные, финансовые, временные и информационные. Рассмотрим их более подробно.

### 1. Человеческие ресурсы

**Описание:**

Люди, участвующие в проекте, их навыки, знания и опыт.

**Важные аспекты:**

- **Командная работа:** Эффективное взаимодействие между членами команды.
- **Навыки и квалификация:** Наличие необходимых технических и управленческих навыков.
- **Обучение и развитие:** Постоянное повышение квалификации сотрудников.

#### Примеры:

- Разработчики, тестировщики, аналитики, проектные менеджеры, UX/UI дизайнеры.

## 2. Материальные ресурсы

#### Описание:

Физические объекты и инфраструктура, используемые в проекте.

#### Важные аспекты:

- **Оборудование и техника:** Компьютеры, серверы, оборудование для тестирования.
- **Инфраструктура:** Офисные помещения, рабочие места.
- **Программное обеспечение:** Лицензии на ПО, инструментальные средства.

#### Примеры:

- Компьютеры, серверы, сети, оборудование для проведения тестов, офисные помещения.

## 3. Финансовые ресурсы

#### Описание:

Денежные средства, необходимые для выполнения проекта.

#### Важные аспекты:

- **Бюджетирование:** Планирование и контроль расходов.
- **Финансирование:** Источники финансирования, управление доходами и расходами.
- **Контроль затрат:** Мониторинг использования финансовых ресурсов.

#### Примеры:

- Зарплаты сотрудников, затраты на оборудование и ПО, операционные расходы.

## 4. Временные ресурсы

#### Описание:

Время, доступное для выполнения проекта и его задач.

#### Важные аспекты:

- **График проекта:** Планирование сроков выполнения задач.
- **Контроль времени:** Отслеживание выполнения задач в соответствии с графиком.
- **Соблюдение дедлайнов:** Обеспечение завершения этапов проекта в установленные сроки.

#### Примеры:

- График выполнения задач, сроки выполнения этапов, контрольные точки проекта.

## 5. Информационные ресурсы

#### Описание:

Данные и информация, необходимые для успешного выполнения проекта.

#### Важные аспекты:

- **Документация:** Наличие актуальных документов, спецификаций и требований.

- **Базы данных:** Доступ к необходимой информации и данным.
- **Коммуникация:** Эффективный обмен информацией между членами команды и заинтересованными сторонами.

#### Примеры:

- Спецификации, техническая документация, базы данных, информационные системы.

### Примеры использования ресурсов:

#### 1. Человеческие ресурсы:

- Проектная команда включает разработчиков, тестировщиков и менеджеров, которые регулярно проводят встречи для координации работы и решения возникающих проблем.

#### 2. Материальные ресурсы:

- Для разработки и тестирования ПО используется современное оборудование и программное обеспечение, обеспечивающее высокую производительность и надежность.

#### 3. Финансовые ресурсы:

- Бюджет проекта включает затраты на зарплаты сотрудников, приобретение оборудования, аренду офисных помещений и лицензии на программное обеспечение.

#### 4. Временные ресурсы:

- График проекта разработан таким образом, чтобы учитывать все этапы разработки, тестирования и внедрения, с учетом возможных задержек и непредвиденных обстоятельств.

#### 5. Информационные ресурсы:

- Вся проектная документация хранится в единой информационной системе, обеспечивающей доступ к актуальной информации для всех членов команды.

Эффективное управление ресурсами включает их планирование, распределение, мониторинг и контроль, что позволяет обеспечить успешное выполнение проекта в установленные сроки, в рамках бюджета и с необходимым качеством.

## 9. Подходы к разработке управления программными проектами.

### «Водопад» или каскадная модель [3]

Управление программными проектами требует различных подходов, которые могут быть выбраны в зависимости от характера проекта, требований к продукту и предпочтений команды. Основные подходы включают водопадную модель, гибкое управление, метод «как получится» и метод частых поставок. Рассмотрим каждый из них более подробно.

### Водопадная или каскадная модель

#### Основные характеристики:

##### 1. Последовательность этапов:

- Проект делится на последовательные этапы: анализ требований, проектирование, реализация, тестирование, внедрение и поддержка.

##### 2. Жесткий контроль:

- Переход к следующему этапу возможен только после завершения предыдущего.

### 3. Документирование:

- Подробная документация создается на каждом этапе, что упрощает контроль и проверку.

#### Преимущества:

- **Простота и ясность:** Четкие и последовательные этапы делают процесс понятным.
- **Строгий контроль:** Хорошо подходит для проектов с четко определенными требованиями.
- **Документирование:** Подробная документация помогает в управлении и отслеживании прогресса.

#### Недостатки:

- **Низкая гибкость:** Трудно вносить изменения на поздних этапах.
- **Задержки:** Возможны значительные задержки при обнаружении ошибок на поздних этапах.
- **Риски:** Высокие риски из-за недостатка возможности адаптации к изменениям.

## 10. Подходы к разработке управления программными проектами. Гибкое управление

### Гибкое управление (Agile)

#### Основные характеристики:

#### 1. Итеративность:

- Проект разбивается на небольшие итерации (спринты), каждый из которых включает полный цикл разработки: планирование, разработка, тестирование и внедрение.

#### 2. Гибкость:

- Возможность адаптироваться к изменениям требований на каждом этапе.

#### 3. Коллаборация:

- Постоянное взаимодействие между командой, заказчиками и пользователями.

#### Преимущества:

- **Гибкость:** Легко адаптироваться к изменениям в требованиях и приоритетах.
- **Быстрая поставка:** Частые релизы позволяют получать обратную связь и вносить коррективы.
- **Улучшенное качество:** Постоянное тестирование и интеграция повышают качество продукта.

#### Недостатки:

- **Неопределенность:** Возможны неопределенности в сроках и бюджете.
- **Требования к команде:** Требуется высокой квалификации и самоорганизации команды.
- **Сложности в масштабировании:** Может быть сложно управлять большими проектами с многочисленными командами.

## 11. Подходы к разработке управления программными проектами. Как получится [2]

### Подход "Как получится"

#### Основные характеристики:

1. **Отсутствие четкой структуры:**
  - Процесс разработки не имеет четко определенной структуры или последовательности.
2. **Импровизация:**
  - Решения принимаются по мере необходимости, без предварительного планирования.
3. **Минимальные процессы:**
  - Минимизация формальностей и процессов.

#### Преимущества:

- **Гибкость:** Максимальная адаптация к текущим условиям и ситуациям.
- **Быстрое реагирование:** Быстрое принятие решений и реализация изменений.

#### Недостатки:

- **Неопределенность:** Высокая степень неопределенности в сроках и результатах.
- **Риски:** Высокие риски из-за отсутствия планирования и структуры.
- **Зависимость от команды:** Требуется высокой квалификации и опыта от команды для минимизации хаоса.

## 12. Подходы к разработке управления программными проектами. Метод частых поставок [4]

### Метод частых поставок (Continuous Delivery)

#### Основные характеристики:

1. **Непрерывная интеграция и поставка:**
  - Постоянное внедрение изменений и улучшений в продукт.
2. **Автоматизация:**
  - Широкое использование автоматизированных инструментов для тестирования, сборки и деплоя.
3. **Постоянная готовность к выпуску:**
  - Продукт всегда находится в состоянии, готовом к выпуску.

#### Преимущества:

- **Быстрая доставка:**
  - Постоянные обновления и улучшения продукта.
- **Высокое качество:**

- Постоянное тестирование и интеграция повышают качество и стабильность.
- **Улучшенная обратная связь:**
  - Частые релизы позволяют получать постоянную обратную связь от пользователей.

#### Недостатки:

- **Сложности с автоматизацией:**
  - Требуется значительных усилий по автоматизации процессов.
- **Требования к инфраструктуре:**
  - Необходимость в надежной и масштабируемой инфраструктуре для поддержки непрерывной интеграции и доставки.
- **Интенсивная работа:**
  - Постоянная работа над релизами может требовать высокой интенсивности труда от команды.

### 13. Мониторинг и контроль рисков [4]

### 14. Управление рисками проекта. Мониторинг и контроль рисков [3]

Как бы не организованно и правильно разрабатывался проект, рано или поздно возникнут проблемы, которые могут увести проект в не то русло. Исходя из этого руководитель должен заранее предугадывать эти проблемы, которые по сути и являются рисками.

Риск проекта – это неопределенное событие, которое каким-либо образом влияет на цели проекта. Например, цель – закрыть проект, не превышая тот бюджет, который был заложен изначально. Тогда любые события с непредвиденными расходами будут считаться риском. Или цель – создать качественный продукт быстрее конкурента. Тогда есть риск опоздать с запуском проекта или потерять своего лучшего поставщика и проиграть в качестве. Как только у проекта появляется цель, то стоит подумать о факторах, которые могут ей помешать.

Риск – это проблема, которая еще не существует, а проблема – это риск, который материализовался. Риск можно описать следующими свойствами:

1. Причина или источник возникновения риска. Явление, которое обуславливает наступление риска.
2. Признаки риска, указание на то, что событие риска произошло или произойдет в ближайшее время. Причина возникновения риска может быть не наблюдаема, например, заразились вирусом. Но можно наблюдать некоторые симптомы – изменение температуры тела.
3. Последствия риска. Проблема, которая может быть реализована в проекте в результате произошедшего риска.
4. Влияние риска. Влияние реализовавшегося риска на цели проекта. Воздействие обычно касается основных показателей, таких как стоимость, график и технические характеристики разрабатываемого продукта. Многие риски происходят частично и оказывают соразмерное отрицательное или положительное воздействие на проект.

Согласно стандарту PMBOK управление рисками происходит в 4 этапа (Рис. 13): 1. Идентификация. Выявить риски, которые могут помешать целям проекта. 2. Анализ. Определить, какие из выявленных рисков наиболее опасны. 3. Планирование. Спланировать наиболее опасные риски. 4. Мониторинг и контроль. Поддерживать план проекта и список рисков в актуальном состоянии

-----

Управление рисками проекта — это процесс выявления, оценки и реагирования на риски, которые могут повлиять на успешное выполнение проекта. Он включает несколько ключевых этапов, таких как идентификация рисков, их анализ, планирование ответных мер, мониторинг и контроль.

## **Мониторинг и контроль рисков**

Мониторинг и контроль рисков — это процесс отслеживания выявленных рисков, выявления новых рисков, выполнения планов по реагированию на риски и оценки их эффективности на протяжении всего проекта. Этот процесс помогает обеспечить, что риски находятся под контролем и что запланированные меры по их минимизации выполняются.

### **Основные шаги мониторинга и контроля рисков:**

1. **Отслеживание выявленных рисков:**
  - Регулярно проверяйте статус известных рисков и анализируйте их влияние на проект.
  - Используйте журналы рисков для документирования всех рисков, их оценки и принятых мер.
2. **Выявление новых рисков:**
  - Проактивно проводите обсуждения в команде, анализируя текущую ситуацию проекта и выявляя новые потенциальные риски.
  - Обеспечьте, чтобы все члены команды знали о процессе выявления рисков и могли внести свои предложения.
3. **Анализ и оценка эффективности планов по реагированию:**
  - Оцените, насколько эффективно работают текущие меры по управлению рисками.
  - Внесите коррективы в планы по реагированию, если это необходимо.
4. **Отчетность и коммуникация:**
  - Регулярно отчитывайтесь перед заинтересованными сторонами о статусе рисков и принятых мерах.
  - Обеспечьте, чтобы все заинтересованные стороны были в курсе текущего состояния рисков и понимали возможные последствия для проекта.
5. **Адаптация и обновление планов управления рисками:**
  - Постоянно обновляйте план управления рисками, включайте в него новые риски и корректируйте стратегии по управлению существующими рисками.
  - Учитесь на опыте и применяйте полученные знания для улучшения будущих проектов.

### **Методы и инструменты мониторинга и контроля рисков:**

1. **Журналы рисков:**
  - Используйте таблицы или специализированные программные инструменты для ведения журналов рисков, чтобы отслеживать их статус, вероятности и влияние.
2. **Регулярные встречи и отчеты:**
  - Проводите регулярные встречи команды для обсуждения текущих рисков и результатов выполненных мер.
  - Создавайте периодические отчеты для руководства и заинтересованных сторон.

### 3. **Ключевые показатели риска (KRI):**

- Установите ключевые показатели риска, которые помогут вам отслеживать изменения в уровне риска.

### 4. **Аудиты и обзоры:**

- Проводите внутренние и внешние аудиты, чтобы проверить эффективность вашего процесса управления рисками.

## **Пример практического применения:**

### 1. **Проектная команда регулярно проводит совещания по рискам:**

- Каждую неделю команда обсуждает текущие риски, обновляет их статус и планирует новые действия.

### 2. **Использование программного обеспечения для управления проектами:**

- Ведется журнал рисков в специализированной системе, где можно отслеживать их развитие и действия по их минимизации.

### 3. **Создание отчетов для руководства:**

- Ежемесячно создаются отчеты для руководства компании, где представлены текущие риски, оценка их воздействия и принятые меры.

Эти шаги и методы помогут обеспечить эффективное управление рисками, что в свою очередь повышает вероятность успешного завершения проекта.

## **15. Планирование реагирования на риски [2]**

Планирование реагирования на риски — это процесс разработки стратегий и планов действий для устранения или минимизации воздействия рисков на проект. Этот этап критически важен для успешного управления проектом, так как позволяет подготовиться к возможным проблемам и минимизировать их негативные последствия.

### **Основные этапы планирования реагирования на риски:**

#### **Идентификация рисков:**

Определите потенциальные риски, которые могут повлиять на проект.

Оцените вероятность их возникновения и потенциальное воздействие на проект.

#### **Приоритизация рисков:**

Классифицируйте риски по степени важности и срочности.

Определите, на какие риски следует обращать внимание в первую очередь.

#### **Разработка стратегий реагирования:**

Определите подходящие стратегии для управления каждым риском.

Стратегии могут включать избегание, смягчение, передачу или принятие риска.



### **Разработка планов действий:**

Для каждого риска разработайте конкретные шаги, которые нужно предпринять в случае его возникновения.

Установите ответственных за выполнение этих шагов и определите сроки.

### **Определение ресурсов и бюджета:**

Определите необходимые ресурсы (люди, время, деньги) для реализации планов реагирования.

Включите эти ресурсы в общий бюджет проекта.

### **Документирование плана управления рисками:**

Составьте подробный план управления рисками, включающий все выявленные риски, стратегии реагирования и планы действий.

Убедитесь, что этот план доступен всем членам команды и заинтересованным сторонам.

### **Стратегии реагирования на риски:**

#### **Избегание риска:**

Измените план проекта, чтобы исключить вероятность возникновения риска.

Пример: изменить маршрут поставки, чтобы избежать неблагоприятных погодных условий.

#### **Смягчение риска:**

Примите меры для уменьшения вероятности или воздействия риска.

Пример: провести дополнительное обучение для команды, чтобы снизить вероятность ошибок.

#### **Передача риска:**

Перенесите ответственность за управление риском на третью сторону.

Пример: застраховать проект от финансовых потерь.

#### **Принятие риска:**

Признайте риск и подготовьтесь к его последствиям, не предпринимая специальных мер для его устранения.

Пример: резервирование бюджета для покрытия возможных затрат.

Планирование реагирования на риски позволяет не только подготовиться к возможным проблемам, но и обеспечивает уверенность в том, что проект будет успешно завершен, несмотря на непредвиденные обстоятельства.

## 16. Разновидности рисков [2]

Риски можно поделить на два типа:

1. Прогнозируемые. Это те риски, которые можно идентифицировать и подвергнуть анализу. В отношении таких рисков можно спланировать действия по их предотвращению и снижению последствий на проект.
2. Непрогнозируемые. Риски, которые невозможно идентифицировать и, следовательно, спланировать ответные действия.

Непрогнозируемые риски – это непредвиденные события. Единственное, что можно в этом случае предпринять, это создать некий резерв бюджета проекта на случай неизвестных, но потенциально возможных изменений. На расходование этого резерва руководитель проекта, как правило, обязан получить одобрение своего руководства. Управленческие резервы на непредвиденные обстоятельства не входят в основной план бюджета проекта, но включаются в бюджет проекта. Они не распределяются по проекту, как бюджет, и поэтому не учитываются при расчете освоенного объема.

-----

Разновидности рисков в управлении проектами можно классифицировать по различным признакам, таким как источник возникновения, сфера воздействия и характер влияния на проект. Вот основные типы рисков:

### 1. По источнику возникновения:

#### 1. Внутренние риски:

- **Организационные риски:** связанные с организационной структурой, процессами и культурой компании.
  - Пример: неэффективное распределение ресурсов, недостаточная квалификация сотрудников.
- **Технические риски:** связанные с техническими аспектами проекта, такими как оборудование, технологии и программное обеспечение.
  - Пример: сбои в работе оборудования, проблемы с совместимостью программного обеспечения.
- **Управленческие риски:** связанные с управлением проектом, такие как планирование, контроль и коммуникация.
  - Пример: неправильное планирование сроков, недостаток коммуникации между командами.

#### 2. Внешние риски:

- **Экономические риски:** связанные с экономической ситуацией в стране или мире.
  - Пример: инфляция, колебания валютных курсов.
- **Политические риски:** связанные с политической ситуацией и изменениями в законодательстве.
  - Пример: изменение налогового законодательства, политическая нестабильность.
- **Природные риски:** связанные с природными явлениями.
  - Пример: землетрясения, наводнения, ураганы.
- **Социальные риски:** связанные с социальными изменениями и общественными настроениями.

- Пример: забастовки, изменение потребительских предпочтений.

## 2. По сфере воздействия:

### 1. Финансовые риски:

- Связаны с финансами и бюджетом проекта.
  - Пример: превышение бюджета, нехватка финансирования.

### 2. Технические риски:

- Связаны с техническими аспектами выполнения проекта.
  - Пример: проблемы с технологическими решениями, сбои в оборудовании.

### 3. Операционные риски:

- Связаны с операционной деятельностью и процессами в проекте.
  - Пример: неэффективные процессы, сбои в поставках материалов.

### 4. Юридические риски:

- Связаны с юридическими аспектами и законодательством.
  - Пример: нарушение контрактных обязательств, изменения в законодательстве.

### 5. Риски безопасности:

- Связаны с безопасностью работников и инфраструктуры.
  - Пример: несоблюдение правил техники безопасности, аварии на производстве.

## 3. По характеру влияния:

### 1. Качественные риски:

- Связаны с качеством выполняемых работ и конечного продукта.
  - Пример: несоответствие продукта техническим требованиям, проблемы с качеством материалов.

### 2. Временные риски:

- Связаны с соблюдением графика проекта.
  - Пример: задержки в поставках, длительные согласования.

### 3. Риски содержания:

- Связаны с изменениями требований и объема работ.
  - Пример: изменение требований заказчика, расширение объема работ.

## Пример классификации рисков:

### 1. Экономический риск (Внешний, Финансовый):

- Пример: Неожиданные изменения валютного курса, влияющие на стоимость импортных материалов.

### 2. Технический риск (Внутренний, Технический):

- Пример: Отказ ключевого оборудования на производственной линии.

### 3. Политический риск (Внешний, Юридический):

- **Пример:** Введение новых нормативных актов, требующих дополнительных проверок и сертификаций.

### 4. Операционный риск (Внутренний, Операционный):

- **Пример:** Низкая производительность команды из-за недостаточного обучения.

Эта классификация помогает лучше понимать природу рисков и разрабатывать эффективные стратегии для их управления.

---

Эльджан

## 17. Модели процесса разработки ПО [3]

Модели, или методологии, процессов разработки программного обеспечения принято классифицировать по количеству формализованных процессов и детальности их регламентации. Чем больше процессов документировано, чем более детально они описаны, тем больше значимость модели. Разработка программного продукта знает много достойных методологий — иначе говоря, устоявшихся best practices. Выбор зависит от специфики проекта, системы бюджетирования, субъективных предпочтений и даже темперамента руководителя. В основном существуют такие модели процесса разработки программного обеспечения как: ГОСТ, SW-CMM, RUP, MSF, PSP/TSP, Agile.

### ГОСТ

Как и для любой деятельности, которая регламентируется какими-либо стандартами, так и для описания процессов разработки программного обеспечения в Республике существуют государственные стандарты. Строгое следование этим гостам не только приводит к водопадному подходу, но и требует очень высокой степени формализованной разработки. На основе этих стандартов разрабатываются программные системы по госзаказам.

### SW-CMM

В CMM представлена схема постепенного улучшения, разделённая по 5 уровням зрелости процессов. Эти 5 уровней представляют собой шкалу для оценки уровня зрелости процессов разработки программного обеспечения в компании и для измерения их параметр.

Основные характеристики каждого уровня:

1. Начальный — процесс разработки носит хаотический характер. Определены лишь немногие из процессов, и успех проектов зависит от конкретных исполнителей.
2. Повторяемый — установлены основные процессы управления проектами: отслеживание затрат, сроков и функциональности. Упорядочены некоторые процессы,

необходимые для того, чтобы повторить предыдущие достижения на аналогичных проектах.

3. Определенный – процессы разработки ПО и управления проектами описаны и внедрены в единую систему процессов компании.

4. Управляемый – собираются детальные количественные данные по функционированию процессов разработки и качеству конечного продукта.

5. Оптимизируемый – постоянное улучшение процессов основывается на количественных данных по процессам и на пробном внедрении новых идей и технологий.

## **RUP**

Рациональный унифицированный процесс (Rational Unified Process, RUP) – одна из спиральных методологий разработки программного обеспечения. Методология поддерживается компанией Rational Software. В качестве языка моделирования в общей базе знаний используется язык Unified Modelling Language (UML). Итерационная разработка программного обеспечения в RUP предполагает разделение проекта на несколько мелких проектов, которые выполняются последовательно, и каждая итерация разработки четко определена набором целей, которые должны быть достигнуты в конце итерации.

## **MSF**

Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной и спиральной. Процесс MSF ориентирован на «вехи» (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата.

## **PSP/TSP**

Индивидуальный процесс разработки (Personal software process, PSP) – процесс разработки программного обеспечения, помогающий разработчикам понимать и улучшать собственную производительность. Создан для применения принципов модели зрелости процессов к практике одного разработчика.

## **Agile**

Методология Agile – это итеративный подход к управлению проектами и разработке программного обеспечения, позволяющий командам ускорить доставку ценности клиентам и избежать лишней головной боли. Вместо того чтобы выпускать весь продукт целиком, Agile-команда выполняет работу в рамках небольших, но удобных инкрементов. Требования, планы и результаты постоянно проходят проверку на актуальность, благодаря чему команды могут быстро реагировать на изменения. При использовании традиционного каскадного подхода к разработке один специалист

заканчивает работу над проектом и передает эстафету следующему, самоустраняясь от участия в дальнейшем процессе. В отличие от этой модели Agile предполагает активное взаимодействие между участниками многофункциональных команд. В основе Agile лежат открытое общение, совместная работа, адаптация и доверительные отношения между участниками команды. Agile не сводится к ряду собраний и конкретных приемов разработки. Agile – это группа методологий, в каждой из которых прослеживается стремление к безостановочному выполнению циклов обратной связи и непрерывному совершенствованию.

Не существует единственного правильного процесса разработки программного обеспечения, в каждом новом проекте процесс должен определяться каждый раз заново, в зависимости от проекта, продукта и персонала. Совершенно разные процессы должны применяться в проектах, в которых участвуют 5 человек, и в проектах, в которых участвуют 500 человек. Если продуктом проекта является критическое программное обеспечение, например, система управления атомной электростанцией, то процесс разработки должен сильно отличаться от разработки сайта. И, наконец, по-разному следует организовывать процесс разработки в команде вчерашних студентов и в команде состоявшихся профессионалов.

## **18. Модели процесса разработки программного обеспечения. Agile [3]**

Методология Agile – это итеративный подход к управлению проектами и разработке программного обеспечения, позволяющий командам ускорить доставку ценности клиентам и избежать лишней головной боли. Вместо того чтобы выпускать весь продукт целиком, Agile-команда выполняет работу в рамках небольших, но удобных инкрементов. Требования, планы и результаты постоянно проходят проверку на актуальность, благодаря чему команды могут быстро реагировать на изменения. При использовании традиционного каскадного подхода к разработке один специалист заканчивает работу над проектом и передает эстафету следующему, самоустраняясь от участия в дальнейшем процессе. В отличие от этой модели Agile предполагает активное взаимодействие между участниками многофункциональных команд. В основе Agile лежат открытое общение, совместная работа, адаптация и доверительные отношения между участниками команды. Agile не сводится к ряду собраний и конкретных приемов разработки. Agile – это группа методологий, в каждой из которых прослеживается стремление к безостановочному выполнению циклов обратной связи и непрерывному совершенствованию.

Agile состоит из 4-х ценностей. Ценности Agile родились в 2001 году в Agile-манифесте – в результате обобщения многих тогдашних «методологий разработки» их авторами. Ценности – это то общее, что определяет приоритеты в работе, независимо от конкретного процесса и предмета работы. Каждая из 4-х ценностей Agile

сформулирована в виде «X важнее Y», где X – это: 1. Люди. 2. Работающий продукт, 3. Сотрудничество с заказчиком, 4. Готовность к изменениям.

### **1. Люди и их взаимодействие важнее процессов и инструментов.**

Чтобы люди работали эффективнее, процессы и инструменты не должны их ограничивать. В Agile ни процесс, ни тем более программный инструмент не диктует, что людям делать. Более того, они сами решают, как менять процессы/инструменты своей работы. Чтобы ускорить процесс разработки, люди также должны взаимодействовать напрямую (без посредников в виде документов или других людей), активно общаться между собой лично, а не письменно.

### **2. Работающий продукт важнее исчерпывающей документации.**

Чтобы клиенты были довольны, им нужен именно работающий продукт. Поэтому разработчики продукта должны фокусироваться именно на том, чтобы продуктом можно было как можно скорее воспользоваться, а не на составлении списков, диаграмм, требований, отчетов перед заказчиком. Чтобы укладываться в сжатые сроки с минимумом затрат, зачастую не стоит связывать себя документацией. Поддержка документации в адекватном продукту состоянии нередко замедляет разработку и требует неоправданно больших затрат.

### **3. Сотрудничество с заказчиком важнее согласований условий контракта.**

Чтобы на выходе получить продукт, действительно ценный для заказчика, стоит отказаться от излишних деталей в контракте между подрядчиком и заказчиком (равно как и в требованиях внутреннего заказчика к внутреннему разработчику продукта). Будучи жестко заданы на старте, детали контракта мешают учитывать новые данные и приоритеты, появляющиеся лишь во время разработки.

### **4. Готовность к изменениям важнее, чем следование плану.**

Чтобы не откладывать риски проектов на последние стадии разработки (когда будет уже поздно уменьшать содержание работы, сдвигать срок или усиливать команду), Agile предлагает не только итеративность работы, но и готовность к изменениям на всех стадиях. Чтобы в первую очередь делалось самое ценное, текущее видение бизнес-ценности и позиционирования продукта должно быть прозрачно для разработчиков, а процесс их работы должен позволять вносить существенные изменения в прежние планы. В том числе, разработчики должны быть готовы добавлять в продукт незапланированные новые возможности, если они стали ценными в изменившейся ситуации

Таким образом, Agile – это методология разработки, система ценностей и принципов, помогающих разработчикам делать новые продукты быстрее и с большим эффектом для бизнеса: – за счет более эффективного взаимодействия с заказчиком и друг с другом, которое не ограничивается жестким контрактом или жестким

внутренним процессом; – за счет быстрой реакции на изменения, причем с обеих сторон; – за счет фокуса на работающий продукт, а не на вспомогательные вещи вроде документации.

## **19. Модели процесса разработки программного обеспечения.**

### **MSF [3]**

В 1994 году, стремясь достичь максимальной отдачи от IT-проектов, компания Microsoft выпустила в свет пакет руководств по эффективному проектированию, разработке, внедрению и сопровождению продукта. Эти знания базируются на опыте, полученном компанией при работе над большими проектами по разработке и сопровождению программного обеспечения, опыте консультантов и лучшем из того, что накопила на данный момент IT-индустрия. Всё это представлено в виде двух взаимосвязанных и хорошо дополняющих друг друга областей знаний: Microsoft Solutions Framework (MSF) и Microsoft Operations Framework (MOF).

Модель процессов MSF представляет общую методологию разработки и внедрения IT решений. Особенность этой модели состоит в том, что благодаря своей гибкости и отсутствию жестко навязываемых процедур она может быть применена при разработке весьма широкого круга IT проектов. Эта модель сочетает в себе свойства двух стандартных производственных моделей: каскадной и спиральной. Процесс MSF ориентирован на «вехи» (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата.

Модель процессов MSF учитывает постоянные изменения проектных требований. Она исходит из того, что разработка решения должна состоять из коротких циклов, создающих поступательное движение от простейших версий решения к его окончательному виду. В рамках MSF программный код, документация, дизайн, планы и другие рабочие материалы создаются, как правило, итеративными методами. MSF рекомендует начинать разработку решения с построения, тестирования и внедрения его базовой функциональности. Затем к решению добавляются все новые и новые возможности. Такая стратегия именуется стратегией версионирования. Несмотря на то, что для малых проектов может быть достаточным выпуск одной версии, рекомендуется не упускать возможности создания для одного решения ряда версий. С созданием новых версий эволюционирует функциональность решения.

В рамках MSF предлагается ряд шаблонов стандартных документов, которые являются артефактами каждой стадии разработки продукта и могут быть использованы для планирования и контроля процесса разработки. Решение не представляет бизнес-ценности, пока оно не внедрено. Именно по этой причине



модель процессов MSF содержит весь жизненный цикл создания решения, включая его внедрение – вплоть до момента, когда решение начинает давать отдачу.

## **20. Модели процесса разработки программного обеспечения. PSP/TSP**

Индивидуальный процесс разработки (Personal software process, PSP) – процесс разработки программного обеспечения, помогающий разработчикам понимать и улучшать собственную производительность. Создан для применения принципов модели зрелости процессов к практике одного разработчика. Предоставляет разработчикам описания методов планирования и оценки, показывает, как измерять собственную продуктивность и соотносить её с существующим планом.

Согласно этой модели, каждый программист должен уметь: 1) *учитывать время, затраченное на работу над проектом;* 2) *учитывать найденные дефекты;* 3) *классифицировать типы дефектов;* 4) *оценивать размер задачи;* 5) *осуществлять систематический подход к описанию результатов тестирования;* 6) *планировать программные задачи;* 7) *распределять их по времени и составлять график работы;* 8) *выполнять индивидуальную проверку проекта и архитектуры;* 9) *осуществлять индивидуальную проверку кода;* 10) *выполнять регрессионное тестирование.*

TSP призван повысить уровень качества и продуктивности проекта разработки программного обеспечения группы, чтобы помочь им лучше справляться с затратами и графиком разработки системы программного обеспечения. Основной целью TSP является создание рабочей среды для создания и поддержания самостоятельной группы и поддержки дисциплинированной индивидуальной работы как основы PSP. Самонаведенная команда означает, что команда управляет собой, планирует и отслеживает свою работу, управляет качеством своей работы и упреждающе работает для достижения целей команды. TSP имеет два основных компонента: командообразование и командная работа. Командообразование – это процесс, определяющий роли каждого члена группы и устанавливающий в рамках TSP запуск и периодическую. Командная работа – это процесс, который занимается инженерными процессами и практиками, разработчиками. TSP предоставляет инженерам и менеджерам способ создания и управления их командой для производства высококачественного программного обеспечения в соответствии с графиком и бюджетом.

Прежде чем инженеры смогут участвовать в TSP, необходимо, чтобы они уже узнали о PSP, чтобы TSP мог эффективно работать. Обучение также требуется для других членов группы и руководства. Цикл разработки программного обеспечения TSP начинается с процесса планирования, называемого запуском, под руководством тренера, который прошел специальную подготовку. Роль тренера сосредоточена на

поддержке команды и отдельных сотрудников в команде в качестве эксперта по процессам, при этом являясь независимым от прямой ответственности за управление проектом.

## 21. Модели процесса разработки программного обеспечения. RUP [3]

Рациональный унифицированный процесс (Rational Unified Process, RUP) – одна из спиральных методологий разработки программного обеспечения. Методология поддерживается компанией Rational Software. В качестве языка моделирования в общей базе знаний используется язык Unified Modelling Language (UML). Итерационная разработка программного обеспечения в RUP предполагает разделение проекта на несколько мелких проектов, которые выполняются последовательно, и каждая итерация разработки четко определена набором целей, которые должны быть достигнуты в конце итерации.

Методология RUP основана на следующих основных потоках, являющихся элементами итерации жизненного цикла программного обеспечения:

1. **Бизнес-анализ** – предполагает анализ требований на данной итерации жизненного цикла, определение желаемых параметров системы и нужд пользователей.
2. **Требования** – формализация образа системы. Предполагает сбор требований и управление требованиями, перевод требований в функциональные спецификации.
3. **Анализ и моделирование** – предполагает перевод собранных требований в формализованную программную модель. Результатом является описание системы на фазе реализации – это документы уровня разработчиков системы. В процессе итеративной разработки эволюционировать будет продукт именно этого потока – модель проекта.
4. **Реализация** – предполагает собственно написание кода. Элементы кода в RUP уже созданы на этапе анализа и дизайна, так как средство реализации UML позволяет создавать элементы кода на нескольких языках программирования. Методология – объектно-ориентированное программирование;
5. **Тестирование** – предполагает тестирование продукта на данной итерации.
6. **Внедрение** – предполагает установку продукта на полигоне заказчика, подготовку персонала, запуск системы и передача материалов отделу продаж.

RUP достаточно хорошо формализован, и наибольшее внимание уделяется начальным стадиям разработки проекта – анализу и моделированию. Таким образом, эта методология направлена на снижение коммерческих рисков посредством обнаружения ошибок на ранних стадиях разработки.

## 22. Модели процесса разработки программного обеспечения. SW-CMM [2]

Первая версия CMM была пересмотрена участниками рабочей встречи, в которой принимали участие около 200 специалистов в области программного обеспечения и членами общества разработчиков. В результате был выпущен стандарт CMM, который до настоящего времени активно используется во всем мире.

Причины такого интереса к CMM вполне очевидны. Несмотря на то, что и сами разработчики программного обеспечения, и их руководство зачастую очень хорошо знают свои постоянные проблемы, они не могут прийти к единому мнению о том, какие изменения необходимы компании в первую очередь. Без выработки единой стратегии проведения улучшений руководство не может найти взаимопонимания со своими сотрудниками относительно наиболее приоритетных задач по улучшению. Для достижения максимального результата от усилий, потраченных на улучшение процессов, необходимо иметь поэтапную стратегию развития, которая позволит улучшать зрелость процессов разработки постепенно, эволюционным путем.

В CMM представлена схема такого постепенного улучшения, разделенная по 5 уровням зрелости процессов. Эти 5 уровней представляют собой шкалу для оценки уровня зрелости процессов разработки программного обеспечения в компании и для измерения их параметров.

Основные характеристики каждого уровня:

1. **Начальный** – процесс разработки носит хаотический характер. Определены лишь немногие из процессов, и успех проектов зависит от конкретных исполнителей.
2. **Повторяемый** – установлены основные процессы управления проектами: отслеживание затрат, сроков и функциональности. Упорядочены некоторые процессы, необходимые для того, чтобы повторить предыдущие достижения на аналогичных проектах.
3. **Определенный** – процессы разработки ПО и управления проектами описаны и внедрены в единую систему процессов компании.
4. **Управляемый** – собираются детальные количественные данные по функционированию процессов разработки и качеству конечного продукта.
5. **Оптимизируемый** – постоянное улучшение процессов основывается на количественных данных по процессам и на пробном внедрении новых идей и технологий.

## 23. Модели процесса разработки программного обеспечения. ГОСТ [2]

Как и для любой деятельности, которая регламентируется какими-либо стандартами, так и для описания процессов разработки программного обеспечения в Республике существуют государственные стандарты. Строгое следование этим гостам не только приводит к водопадному подходу, но и требует очень высокой степени формализованной разработки. На основе этих стандартов разрабатываются программные системы по госзаказам.

В Республике Беларусь существуют государственные стандарты:

**ГОСТ 19.201-78. «Единая система программной документации. Техническое задание. Требования к содержанию и оформлению».**

**ГОСТ 34.602-89. «Информационная технология. Комплекс стандартов на автоматизированные системы. Техническое задание на создание автоматизированной системы».**

Вот некоторые из ключевых стандартов ГОСТ, относящихся к процессу разработки программного обеспечения в России:

ГОСТ Р ИСО/МЭК 12207-2010 - "Информационная технология. Процессы жизненного цикла программных средств". Этот стандарт описывает процессы, необходимые для разработки и сопровождения программного обеспечения, включая процессы управления, процессы разработки, процессы эксплуатации и поддержки.

ГОСТ Р 57193-2016 - "Системная и программная инженерия. Обеспечение качества и тестирование систем и программного обеспечения". Стандарт описывает процессы обеспечения качества и тестирования программного обеспечения, охватывая различные методы и подходы к тестированию.

ГОСТ 34.602-89 - "Техническое задание на создание автоматизированной системы". Этот стандарт определяет требования к содержанию и оформлению технического задания на создание автоматизированной системы, что является важным этапом в процессе разработки ПО.

ГОСТ 34.601-90 - "Автоматизированные системы. Стадии создания". Стандарт описывает стадии создания автоматизированных систем, что включает анализ требований, проектирование, разработку, тестирование и внедрение.

Каждая страна может иметь свои специфические требования и стандарты, разработанные в соответствии с национальными потребностями и законодательством. Для получения точной информации о национальных стандартах следует обратиться к соответствующим органам стандартизации и сертификации в стране, таким как Государственный комитет по стандартизации, метрологии и патентам.

Проект – это основа инноваций. Он является двигателем прогресса и ключевым элементом в создании уникальных решений и продуктов, которые могут изменить рынок и повседневную жизнь людей. В современном мире, где технологии развиваются с невероятной скоростью, успех компании часто зависит от её способности быть на шаг впереди конкурентов, предлагая новаторские решения, которые еще не были реализованы.

### Значение проекта в инновациях

1. **Определение новых возможностей:** Проекты позволяют исследовать и открывать новые возможности, которые могут не быть очевидными на первый взгляд. Это может включать разработку новых продуктов, улучшение существующих процессов или внедрение передовых технологий. Идеи, которые возникают в рамках проектов, часто становятся основой для значительных изменений и улучшений.
2. **Быстрота и адаптивность:** В конкурентной среде важно действовать быстро. Проекты дают возможность оперативно разрабатывать и тестировать новые идеи, снижая время от концепции до рынка. Компании, способные быстро адаптироваться и внедрять инновации, получают значительное преимущество перед теми, кто придерживается более медленных и консервативных подходов.
3. **Качество и удовлетворение потребностей:** Современные потребители ожидают высококачественные продукты и услуги, которые удовлетворяют их потребности и превышают ожидания. Проекты позволяют сосредоточиться на качестве, используя современные методы разработки и тестирования, чтобы предложить продукты, которые действительно важны для клиентов.
4. **Создание новых потребностей:** Иногда инновации приводят к созданию продуктов, потребность в которых потребители еще не осознают. Такие проекты требуют глубокого понимания тенденций и поведения пользователей, а также смелости предлагать новые решения. Успешные примеры включают в себя продукты, которые кардинально изменили рынок, такие как смартфоны, социальные сети или электромобили.

### Примеры инновационных проектов

1. **Смартфоны:** В начале 2000-х годов мало кто мог представить, что телефон станет устройством для работы, развлечений и общения в одном. Проекты, такие как iPhone от Apple, перевернули рынок, создав новый стандарт для мобильных устройств.
2. **Социальные сети:** Facebook и другие платформы изменили способ, которым люди взаимодействуют друг с другом. Эти проекты не только удовлетворили существующие потребности в общении, но и создали новые, предложив возможности, о которых пользователи даже не думали.
3. **Электромобили:** Tesla и другие компании продвинули идею электромобилей, предлагая экологически чистые и эффективные решения, которые теперь становятся все более популярными и доступными.

Проекты играют ключевую роль в процессе инноваций, открывая новые возможности, позволяя действовать быстро и эффективно, а также создавая продукты, которые удовлетворяют и превосходят ожидания потребителей. Компании, которые ставят проекты в центр своей стратегии, имеют все шансы стать лидерами рынка и задавать тон в своей отрасли, предлагая решения, которые действительно меняют мир.

## 25. Проект и организационная структура компании

Проект и организационная структура компании тесно связаны между собой и играют ключевую роль в успехе любой организации. В условиях быстро меняющегося рынка и высокой конкуренции, способность компании эффективно управлять проектами и адаптировать свою организационную структуру под их требования становится важным фактором для достижения стратегических целей.

Организационная структура компании определяет, как распределяются роли, обязанности и коммуникации внутри организации. Существует несколько типов организационных структур, и выбор подходящей структуры может существенно повлиять на эффективность управления проектами.

1. **Функциональная структура:** В функциональной структуре компания разделена на отделы, каждый из которых специализируется на определенной функции, такой как маркетинг, финансы, производство и т.д. Проекты в такой структуре могут сталкиваться с трудностями из-за недостатка межфункционального взаимодействия, что может замедлять процесс принятия решений и адаптации.
2. **Проектная структура:** В проектной структуре основное внимание уделяется проектам. Сотрудники объединяются в команды, работающие над конкретными проектами, и каждый проект управляется независимо. Это способствует лучшему управлению ресурсами и повышению ответственности за результаты. Однако такая структура может привести к дублированию усилий и ресурсов между проектами.
3. **Матричная структура:** Матричная структура представляет собой сочетание функциональной и проектной структур. Сотрудники подчиняются как функциональным руководителям, так и менеджерам проектов. Это позволяет эффективно использовать ресурсы и навыки сотрудников, одновременно обеспечивая гибкость и адаптивность. Матричная структура требует четкого управления и координации, чтобы избежать конфликтов и путаницы.

Проекты и организационная структура компании – это два взаимосвязанных элемента, которые определяют способность организации к инновациям и



эффективному достижению своих целей. Компании, которые успешно интегрируют управление проектами в свою организационную структуру, могут быстрее адаптироваться к изменениям рынка, использовать новые возможности и достигать устойчивого роста. Важно помнить, что выбор подходящей организационной структуры и методы управления проектами должны соответствовать специфическим потребностям и целям каждой компании, чтобы обеспечить максимальную эффективность и успешность проектов.

---

## Назага

### 26. Жизненный цикл проекта. Фазы и продукты [4]

**Программный продукт** – совокупность программ и сопроводительной документации по их установке, настройке, использованию и доработке.

Согласно стандарту **жизненный цикл** программы, программной системы, программного продукта включает в себя разработку, развертывание, поддержку и сопровождение.

Если программный продукт не коробочный, а достаточно сложный, то его **развертывание** у клиентов, как правило, реализуется отдельными самостоятельными проектами внедрения. **Сопровождение** включает в себя устранение критических неисправностей в системе и реализуется часто не как проект, а как процессная деятельность. **Поддержка** заключается в **разработке** новой функциональности, переработке уже существующей функциональности, в связи с изменением требований, и улучшением продукта, а также устранение некритических замечаний к программному обеспечению, выявленных при его эксплуатации. Жизненный цикл программного продукта завершается **выводом продукта из эксплуатации и снятием его с поддержки и сопровождения**.

### 27. Концепция проекта [2]

Концепция проекта разрабатывается на основе анализа потребностей бизнеса. Главная функция документа – подтверждение и согласование единого видения целей, задач и результатов всеми участниками проекта. **Концепция определяет что и зачем делается в проекте.**

**Концепция – проекта это ключевой документ**, который используется для принятия решений в ходе всего проекта, а также на фазе приемки для подтверждения результата. Она содержит, как правило, следующие разделы:

1. Название проекта.
2. Цели проекта.
3. Результаты проекта (требования, конечные продукты).
4. Допущения и ограничения.
5. Ключевые участники и заинтересованные стороны.
6. Ресурсы проекта.

7. Сроки.

8. Риски.

9. Критерии приемки.

10. Обоснование полезности проекта.

### Цели и результаты проекта

Цели проекта – желаемый конечный результат, выраженный в конкретных, измеримых и достижимых показателях, значимых для стратегических целей компании. Четкое определение бизнес-целей важно для всех процессов и решений в проекте. Возможные цели проекта:

- 1)Изменения в компании: Повышение эффективности производственной деятельности.
- 2)Реализация стратегических планов: Завоевание значительной доли рынка новым продуктом.
- 3)Выполнение контрактов: Разработка программного обеспечения по заказу.
- 4)Разрешение специфических проблем: Доработка программного продукта для соответствия законодательству.

Результаты проекта должны быть измеримыми для оценки их соответствия оговоренным целям. Цели должны определить:

- 1)Бизнес-выгоды для заказчика.
- 2)Конкретный продукт или услугу, произведенные по окончании проекта.
- 3)Высокоуровневые требования, ключевые свойства и характеристики продукта/услуги.

### Допущения и ограничения

Исходные допущения и ограничения связаны с управлением рисками. В разработке ПО риски часто формулируются как допущения. Например, при оценке проекта с фиксированной ценой предполагается, что стоимость лицензий на стороннее ПО останется неизменной до завершения проекта.

Ограничения уменьшают возможности команды в выборе решений и могут включать:

- 1)Нормативные требования: Обязательная сертификация продукта на соответствие стандартам.
- 2)Технические требования: Разработка под заданную программно-аппаратную платформу.
- 3)Требования к защите информации.



4) Требования к системе, не включенные в проект: Например, разработка интерфейса для будущей интеграции с другими системами не входит в задачи данного проекта.

### Ключевые участники и заинтересованные стороны

На этапе инициации проекта важно выявить и описать всех его участников, которые активно участвуют в проекте или чьи интересы могут быть затронуты. Участники могут влиять на проект и его результаты. К ключевым участникам относятся:

1) Спонсор проекта: Лицо или группа, предоставляющая финансовые ресурсы.

2) Заказчик проекта: Лицо или организация, использующие продукт или услугу проекта (не всегда совпадает со спонсором).

3) Пользователи результатов проекта.

4) Куратор проекта: Представитель исполнителя, уполномоченный принимать решения о ресурсах и изменениях.

5) Руководитель проекта: Ответственный за реализацию проекта в срок, в пределах бюджета и с заданным качеством.

6) Соисполнители проекта: Субподрядчики и поставщики.

### Ресурсы проекта

Для оценки стоимости проекта необходимо определить и оценить ресурсы, нужные для его выполнения:

1) Людские ресурсы и квалификация персонала.

2) Оборудование, услуги, расходные материалы, лицензии на ПО, критические компьютерные ресурсы.

3) Бюджет проекта: План расходов и предполагаемых доходов с разбивкой по статьям и этапам проекта.

В программных проектах основную часть затрат составляют людские ресурсы. Остальные затраты, как правило, незначительны по сравнению с затратами на персонал. Хорошей считается оценка трудозатрат с точностью от -50% до +100% на этапе инициации. Программирование составляет лишь четверть всех затрат, а другие процессы требуют ресурсов соответствующей квалификации. Перед определением численности и состава команды необходимо оценить трудоемкость разработки программных средств в человеко-часах.

### Сроки проекта

Невозможно завершить проект быстрее, чем за 3/4 оптимального расчетного графика, независимо от количества специалистов. Необходимо также определить этапы проекта – контрольные точки (вехи), для переоценки на основе достигнутых показателей.

Контрольная точка – важный момент или событие в расписании проекта, отмечающее достижение заданного результата или начало/завершение определенного объема работы. Каждая контрольная точка имеет дату и объективные критерии достижения.

Для программного проекта контрольные точки должны соответствовать выпуску каждой промежуточной версии ПО, реализующей и тестирующей определенную часть функциональности. Продолжительность итерации может составлять от 2 до 4 недель в зависимости от сложности и масштаба проекта.

### Риски проекта

Риск – неопределенное событие или условие, которое может негативно или положительно повлиять на цели проекта. Негативные риски обычно увеличивают стоимость проекта и вызывают задержки в выполнении мероприятий.

На этапе инициации, из-за отсутствия данных для детального анализа, часто проводится качественная оценка общего уровня рисков:

Низкий риск: Минимальное влияние на проект.

Средний риск: Умеренное влияние, возможны незначительные задержки или увеличения стоимости.

Высокий риск: Существенное влияние, вероятны значительные задержки и увеличение стоимости.

### Критерии приемки проекта

Критерии приемки должны устанавливать числовые значения характеристик системы, которые необходимо продемонстрировать при приемо-сдаточных испытаниях или опытной эксплуатации. Эти критерии должны однозначно подтверждать достижение целей проекта.

### Обоснование полезности проекта

Этот раздел концепции должен содержать краткое технико-экономическое обоснование проекта:

- 1) Для кого предназначены результаты проекта: Описание текущей ситуации и существующих проблем у потенциального заказчика.
- 2) Решение проблем: Каким образом результаты проекта решают эти проблемы.
- 3) Значимость решения: Оценка экономического эффекта для клиента, показывающая, насколько важно решение данных проблем.
- 4) Преимущества для компании-исполнителя: Описание выгод и преимуществ, которые компания-исполнитель может получить от реализации проекта.

### 28. Критерии успешности проекта [3]

Задача проекта – достижение конкретной бизнес-цели, при соблюдении ограничений «железного треугольника». Это означает, что ни один из углов треугольника не может быть изменен без оказания влияния на другие. Например, чтобы уменьшить время, потребуется увеличить стоимость и/или сократить содержание.

Согласно текущей редакции стандарта PMBOK, проект считается успешным, если удовлетворены все требования заказчика и участников проекта. Поэтому у проекта разработки ПО сегодня не три, а четыре фактора успеха:

- 1) Выполнен в соответствии со спецификациями.
- 2) Выполнен в срок.
- 3) Выполнен в пределах бюджета.
- 4) Каждый участник команды уходил с работы с чувством успеха.

Этот четвертый фактор успеха должен стать воспроизводимым, если предприятие хочет быть эффективным. Для успешного проекта характерно постоянное ощущение его участниками чувства удовлетворения и гордости за результаты своей работы, чувства оптимизма. Нет ничего более губительного для проекта, чем равнодушие или уныние его участников.



Рис. 10. «Железный треугольник» проекта

## Эффективность

Эффективность – это отношение полученного результата к произведённым затратам. Нельзя оценивать эффективность только по результативности: чем больше производится, тем выше эффективность. Такой подход может быть ошибочным, так как затраты не следует путать с инвестициями.

Примеры затрат: оплата аренды, электроэнергии, коммунальные платежи.

Примеры инвестиций: создание и закрепление эффективной команды; обучение участников проекта.

Вложение в людей увеличивает числитель в формуле эффективности. Уход профессионалов после проекта, выполненного любой ценой, приводит к значительным и трудно восполнимым затратам.

Современная конкуренция указывает, что персонал – это инвестиции и активы, которые нужно уметь наращивать, управлять и сохранять. Сегодня люди – это капитал.

Предприятие должно относиться к своим работникам так же, как к лучшим клиентам, поскольку знания, которыми они обладают, неотъемлемы от их носителей – людей.

Современные предприятия должны предоставлять услуги и продукты клиентам, а также рабочие места для профессионалов. Принципы "Одно предприятие на всю жизнь" и "Работай продуктивно, а предприятие о тебе позаботится" уходят в прошлое. В IT-секторе правила устанавливают профессионалы.

## 29. Цели и результаты проекта [4]

Цель – желаемый информационный образ конечного продукта. Цели должны убеждать, для чего нужен проект, что конкретно он производит, что надо изменить, как должно быть. Цели должны быть:

- 1) значимыми (направленными на достижение стратегических целей компании), конкретными (специфичными для данного проекта),
- 2) измеримыми (иметь проверяемые количественные оценки),
- 3) реальными (до-стижимыми).

Четкое определение бизнес-целей важно, поскольку существенно влияет на все процессы и решения в проекте. Целями проекта могут быть:

1. Изменения в компании. Например, повышение эффективности основной производственной деятельности.
2. Реализация стратегических планов. Например, завоевание значительной доли растущего рынка за счет вывода на него нового продукта.
3. Выполнение контрактов. Например, обеспечение разработки программного обеспечения по заказу.
4. Разрешение специфических проблем. Например, обеспечение доработки программного продукта в целях приведения его в соответствие с изменениями в законодательстве.

Результаты проекта должны быть измеримыми, т. е. при их оценке должна быть возможность сделать заключение, достигнуты оговоренные в концепции результаты или нет. Цели должны определять: Какие именно бизнес-выгоды получит заказчик в результате проекта. Какой продукт или услуга. Что конкретно будет произведено по окончании проекта. Высокоуровневые требования. Краткое описание и, при необходимости, ключевые свойства и/или характеристики продукта/услуги.

## 30. Организационная структура проекта [2]

Организационная структура – это согласованное и утвержденное распределение ролей, обязанностей и целей деятельности ключевых участников проекта. Она в обязательном порядке должна включать в себя систему рабочих взаимоотношений между рабочими группами проекта, систему отчетности, оценки хода выполнения проекта и систему принятия решений. Следует помнить, что организационная структура проекта – «живой» организм. Она начинает складываться на стадии планирования и должна меняться по ходу проекта.

Нестабильность организационной структуры – частая смена исполнителей может стать серьезной проблемой в управлении проектом, поскольку, существует цена замены, которая определяется временем вхождения нового участника в контекст проекта.

31. Инициация проекта [2] (Этот ответ можно написать в вопросы 27, 28, 33)

Классическое управление проектами выделяет два вида организации деятельности: операционная и проектная.

Операционная деятельность применяется, когда внешние условия хорошо известны и стабильны, когда производственные операции хорошо изучены и неоднократно испытаны, а функции исполнителей определены и постоянны. В этом случае основой эффективности служат узкая специализация и повышение компетенции. «Если водитель трамвая начнет искать новые пути, жди беды».

Там, где разрабатывается новый продукт, внешние условия и требования к которому постоянно меняются, где применяемые производственные технологии используются впервые, где постоянно требуются поиск новых возможностей, интеллектуальные усилия и творчество, там требуются проекты.

Проект – временное предприятие, предназначенное для создания уникальных продуктов, услуг или результатов.

У операционной и проектной деятельности есть ряд общих характеристик:

1) выполняются людьми,

2) ограничены доступностью ресурсов,

3) планируются,

4) исполняются и управляются.

Операционная деятельность и проекты различаются, главным образом, тем, что операционная деятельность – это продолжающийся во времени и повторяющийся процесс, в то время как проекты являются временными и уникальными.

Ограничение по срокам означает, что у любого проекта есть четкое начало и четкое завершение. Завершение наступает, когда достигнуты цели проекта; или осознано, что цели проекта не будут или не могут быть достигнуты; или исчезла необходимость в проекте, и он прекращается.

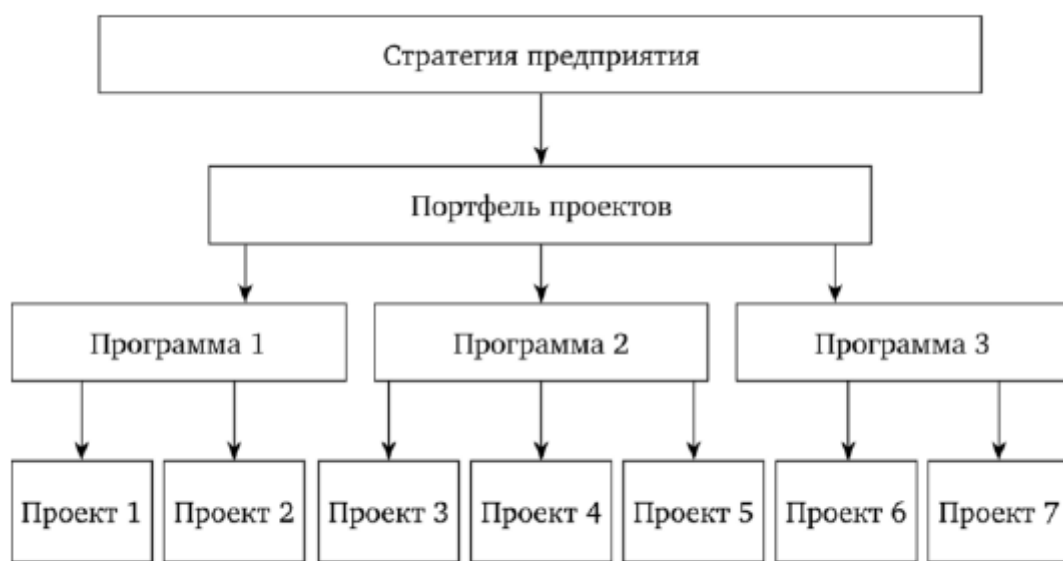
Уникальность так же важное отличие проектной деятельности от операционной. Если бы результаты проекта не носили уникальный характер, работу по их достижению можно было бы четко регламентировать, установить производственные нормативы и реализовывать в рамках операционной деятельности. Задача проекта – достижение конкретной бизнес-цели. Задача операционной деятельности – обеспечение нормального течения бизнеса.

Проект – это средство стратегического развития (Рисунок 9). Цель – описание того, что мы хотим достичь. Стратегия – констатация того, каким образом мы собираемся эти цели достигать. Проекты преобразуют стратегии в действия, а цели в реальность.

Таким образом, каждая работа, которую выполняет конкретный сотрудник, привязывается к достижению стратегических целей организации.

Проекты объединяются в программы. Программа – ряд связанных друг с другом проектов, управление которыми координируется для достижения преимуществ и степени управляемости, недоступных при управлении ими по отдельности.

Проекты и программы объединяются в портфели. Портфель – набор проектов или программ и других работ, объединенных вместе с целью эффективного управления данными работами для достижения стратегических целей.



**Рис. 9. – Структура проектной деятельности**

Проекты и управление ими существовали всегда. В качестве самостоятельной области знаний управление проектами начало формироваться в начале 20 века. В этой дисциплине пока нет единых международных стандартов. Наиболее известные центры компетенции:

– PMI, Project Management Institute, PMBOK – американский национальный стандарт ANSI/PMI 99-001-2004.

– IPMA, International Project Management Association.

Проект – это основа инноваций. Сделать то, до чего другие компании еще не додумались, сделать это как можно быстрее, иначе это сделают другие. Предложить потребителю более качественный продукт или такой продукт, потребность в котором потребитель даже не может пока осознать.



## 32. Содержание проекта [3]

«Если не получается проглотить слона целиком, то его надо порезать на отбивные». Человечество пока не придумало ничего более эффективного для решения сложной задачи, чем анализ и ее декомпозиция на более простые подзадачи, которые, в свою очередь, могут быть разделены на еще более простые подзадачи и так далее. Получается некоторая иерархическая структура, дерево, в корне которого находится проект, а на листьях элементарные задачи или работы, которые надо выполнить, чтобы завершить проект в условиях заданных ограничений.

Иерархическая структура работ (Work Breakdown Structure, WBS) – ориентированная на результат иерархическая декомпозиция работ, выполняемых командой проекта для достижения целей проекта и необходимых результатов. С ее помощью структурируется и определяется все содержание проекта. Каждый следующий уровень иерархии отражает более детальное определение элементов проекта.

Основой для разработки иерархической структуры работ служит концепция проекта, которая определяет продукты проекта и их основные характеристики. Иерархическая структура работ обеспечивает выявление всех работ, необходимых для достижения целей проекта. Многие проекты проваливаются не от того, что у них нет плана, а от того, что в этом плане забыты важные работы, например, тестирование и исправление ошибок, и продукты проекта, например, пользовательская документация. Поэтому, если иерархическая структура работ составлена корректно, то любая работа, которая в нее не вошла, не может считаться работой по проекту.

Иерархическая структура работ является одним из основных инструментов в механизме управления проектом, с помощью которого измеряется степень достижения результатов проекта. Важнейшая ее функция – это обеспечить консистентное представление всех участников проекта относительно того, как будет делаться проект. В последующем базовый план будет служить ориентиром для сравнения с текущим исполнением проекта и выявления отклонений для целей управления.

## 33. Управление приоритетами проектов [3]

Приоритет любого проекта определяется на основе трех основных характеристик: **финансовой ценности, стратегической значимости и уровня риска.**

Финансовая ценность проекта оценивается по следующей шкале:

- 1) Высокая: окупаемость до 1 года, доходы превышают расходы более чем в 1.5 раза.
- 2) Выше среднего: окупаемость от 1 года до 3 лет, доходы превышают расходы более чем в 1.3 раза.
- 3) Средняя: улучшение производственной эффективности на 30%, информационная ценность или улучшение контроля над бизнесом.
- 4) Низкая: незначительное снижение расходов на 10% и улучшение производительности.

Стратегическая ценность проекта оценивается по следующей шкале:

- 1) Высокая: обеспечивает стратегическое преимущество, увеличивает рынок или открывает новые сегменты, решает значимые проблемы клиентов.
- 2) Выше среднего: создает временные конкурентные преимущества, удержание важных клиентов.
- 3) Средняя: поддерживает доверие рынка, улучшает мнение клиентов.
- 4) Низкая: стратегическое воздействие минимально, легко повторяемо конкурентами.

Уровень риска проекта оценивается по следующей шкале:

- 1) Низкий: четкие цели и требования, доступные ресурсы.
- 2) Средний: нечеткие цели, ограниченные ресурсы.
- 3) Выше среднего: неопределенные цели, ограниченные ресурсы и новая технологическая платформа.
- 4) Высокий: неопределенные цели, отсутствие ресурсов и стабильности технологической платформы.

Управление приоритетами проектов критично для избежания перегрузки и повышения эффективности производственной деятельности компании.

---

Джамиля

## 34. Анализ рисков

Анализ рисков (risk analysis) – это процесс идентификации и оценки потенциальных рисков, которые могут повлиять на достижение целей организации или проекта. Анализ рисков можно разделить на 2 этапа: качественный анализ и количественный анализ.

### 1. Качественный анализ рисков

**Цель:** Идентифицировать и описать потенциальные риски, а также их влияние на проект или бизнес.

Качественный анализ рисков фокусируется на описании и оценке рисков в качественных терминах. Основные шаги и методы включают:

- **Идентификация рисков:** Этот шаг был упомянут ранее, но не включен в ваш список. Идентификация рисков заключается в выявлении всех потенциальных рисков, которые могут повлиять на проект или организацию.
- **Определение вероятности реализации рисков:** Оценка того, насколько вероятно, что каждый риск наступит. Это часто делается с использованием качественных категорий, таких как "низкий", "средний" и "высокий" риск.
- **Определение тяжести последствий реализации рисков:** Оценка возможного воздействия каждого риска на проект или организацию. Это также делается с использованием качественных категорий, таких как "незначительное", "умеренное" и "серьезное" воздействие.
- **Определение ранга риска по матрице «вероятность – последствия»:** Построение матрицы рисков, где каждый риск оценивается по двум осям: вероятность и тяжесть последствий. Это позволяет ранжировать риски и определить, какие из них требуют наибольшего внимания.



- **Определение близости наступления риска:** Оценка времени, в течение которого риск может наступить. Это может включать определение рисков, которые могут возникнуть в краткосрочной, среднесрочной или долгосрочной перспективе.
- **Оценка качества использованной информации:** Оценка надежности и достоверности информации, использованной для анализа рисков. Это может включать проверку источников данных, степень неопределенности и наличие допущений.

## **2. Количественный анализ рисков**

**Цель:** Измерить и ранжировать риски по степени их важности.

Количественный анализ рисков предполагает использование количественных данных и методов для точного определения вероятности и последствий рисков. Основные шаги и методы включают:

### **1. Анализ чувствительности**

Анализ чувствительности – это метод, который оценивает, как изменения в одной или нескольких переменных влияют на результаты проекта. Цель анализа чувствительности – определить, какие переменные оказывают наибольшее влияние на результаты и, следовательно, на риски.

**Шаги анализа чувствительности:**

1. Определение ключевых переменных проекта.
2. Изменение значений этих переменных в заданных пределах.
3. Оценка влияния изменений на результаты проекта.
4. Определение наиболее чувствительных переменных, которые оказывают наибольшее влияние на результаты.

### **2. Анализ дерева решений**

Анализ дерева решений – это метод, который используется для принятия решений в условиях неопределенности. Дерево решений представляет собой графическое представление решений и их возможных последствий, включая вероятности и ценность результатов.

**Шаги анализа дерева решений:**

1. Определение решения или проблемы.
2. Построение дерева решений, включающего узлы решений (точки выбора) и узлы событий (точки неопределенности).
3. Присвоение вероятностей и значений каждому исходу.
4. Оценка каждого пути в дереве на основе вероятностей и ценностей.
5. Выбор оптимального решения на основе анализа дерева.

### **3. Моделирование и имитация**

Моделирование и имитация – это методы, которые используют математические модели для оценки и анализа рисков. Одним из наиболее распространенных методов является метод Монте-Карло, который позволяет моделировать различные сценарии и оценивать их вероятности и последствия.

**Шаги моделирования и имитации:**

1. Определение ключевых переменных и их распределений вероятностей.
2. Построение математической модели проекта или процесса.
3. Проведение большого числа симуляций с использованием случайных значений для переменных.
4. Оценка результатов симуляций для определения вероятностей и последствий различных сценариев.
5. Анализ распределения результатов для оценки рисков и принятия решений.

Критерии оценки качества информации, используемой при анализе рисков, важны для обеспечения точности и надежности анализа. Рассмотрим каждый из упомянутых критериев подробнее:

## 1. Степень понимания риска

**Степень понимания риска** отражает, насколько хорошо команда проекта или организации понимает природу, причины и возможные последствия риска. Чем лучше риск понят, тем точнее можно его оценить и управлять им.

### Основные аспекты:

- **Ясность определения риска:** Насколько четко и подробно описан риск.
- **Понимание причин и факторов риска:** Знание всех факторов, которые могут способствовать наступлению риска.
- **Оценка возможных последствий:** Насколько точно и подробно описаны потенциальные последствия риска.
- **Исторические данные и опыт:** Наличие и использование предыдущего опыта и данных для понимания риска.

## 2. Доступность и полнота информации о риске

**Доступность и полнота информации о риске** определяет, насколько полно и доступно необходимая информация для оценки риска. Это включает наличие всех данных, необходимых для полного понимания и оценки риска.

### Основные аспекты:

- **Полнота данных:** Насколько полными являются данные, покрывают ли они все аспекты риска.
- **Доступность данных:** Насколько легко доступна информация для анализа (например, в базах данных, отчетах, документах).
- **Своевременность данных:** Насколько актуальны и своевременны данные, используемые для оценки риска.
- **Объем данных:** Наличие достаточного объема данных для проведения точного анализа.

## 3. Надежность, целостность и достоверность источников данных

**Надежность, целостность и достоверность источников данных** отражает качество и надежность источников, откуда получена информация о рисках. Это включает проверку источников на предмет их достоверности и целостности данных.

### Основные аспекты:

- **Надежность источников:** Репутация и авторитетность источников данных.
- **Целостность данных:** Убедительность и непротиворечивость данных, отсутствие пропущенных или искаженных данных.

- **Достоверность данных:** Проверка на точность и правильность информации, соответствие реальным фактам.
- **Источники информации:** Проверка на использование первичных (надежных и проверенных) источников информации.

## 35. Анализ рисков и оценка его последствий [2]

Анализ рисков и оценка его последствий – это два взаимосвязанных процесса, которые являются ключевыми элементами управления рисками.

### Оценка последствий:

**Цель:** Определение и измерение потенциального ущерба, который может быть нанесён каждым риском.

Оценка последствий рисков может быть выполнена с использованием различных методов и подходов. Давайте подробнее рассмотрим каждый из перечисленных методов:

### 1. Качественный анализ

**Качественный анализ** рисков фокусируется на описании и оценке рисков в качественных терминах. Это включает оценку вероятности и последствий рисков с использованием нечисловых категорий (например, низкий, средний, высокий).

#### Основные шаги:

- Идентификация рисков.
- Оценка вероятности и последствий с использованием качественных категорий.
- Ранжирование рисков на основе их качественной оценки.
- Использование матрицы рисков для визуализации и приоритизации рисков.

### 2. Количественный анализ

**Количественный анализ** рисков использует числовые данные и методы для точной оценки вероятности и последствий рисков. Это включает использование статистических данных и математических моделей.

#### Основные шаги:

- Сбор и анализ данных для количественной оценки рисков.
- Использование методов, таких как анализ Монте-Карло, для моделирования различных сценариев.
- Расчет вероятностей и последствий в числовых выражениях.
- Оценка совокупного воздействия рисков на проект или организацию.

### 3. Анализ чувствительности

**Анализ чувствительности** оценивает, как изменения в одной или нескольких переменных влияют на результаты проекта или оценки рисков. Это помогает выявить наиболее критические переменные, которые оказывают наибольшее влияние на результаты.

#### Основные шаги:

- Определение ключевых переменных проекта.

- Изменение значений этих переменных в заданных пределах.
- Оценка влияния изменений на результаты проекта.
- Определение наиболее чувствительных переменных.

#### 4. Моделирование рисков

**Моделирование рисков** включает создание моделей для оценки и анализа различных сценариев риска. Один из распространенных методов моделирования – метод Монте-Карло, который использует случайные значения для оценки вероятности и последствий рисков.

##### Основные шаги:

- Определение ключевых переменных и их распределений вероятностей.
- Построение математической модели проекта или процесса.
- Проведение большого числа симуляций с использованием случайных значений для переменных.
- Оценка результатов симуляций для определения вероятностей и последствий различных сценариев.

#### 5. Анализ ожидаемой денежной стоимости (EWS)

**Анализ ожидаемой денежной стоимости (EWS)** используется для количественной оценки финансовых последствий рисков. Этот метод помогает определить ожидаемую стоимость риска, учитывая вероятность его наступления и потенциальные финансовые потери.

##### Основные шаги:

- Определение всех возможных рисков и их вероятностей.
- Оценка возможных финансовых последствий каждого риска.
- Расчет ожидаемой денежной стоимости для каждого риска (вероятность  $\times$  финансовые последствия).
- Суммирование ожидаемых денежных стоимостей для всех рисков для получения общей оценки.

### 36. Качественный анализ рисков [3]

### 37. Количественный анализ рисков [2]

#### Качественный анализ рисков

**Цель:** Идентифицировать и описать потенциальные риски, а также их влияние на проект или бизнес.

Качественный анализ рисков фокусируется на описании и оценке рисков в качественных терминах. Основные шаги и методы включают:

- **Идентификация рисков:** Этот шаг был упомянут ранее, но не включен в ваш список. Идентификация рисков заключается в выявлении всех потенциальных рисков, которые могут повлиять на проект или организацию.
- **Определение вероятности реализации рисков:** Оценка того, насколько вероятно, что каждый риск наступит. Это часто делается с использованием качественных категорий, таких как "низкий", "средний" и "высокий" риск.

- **Определение тяжести последствий реализации рисков:** Оценка возможного воздействия каждого риска на проект или организацию. Это также делается с использованием качественных категорий, таких как "незначительное", "умеренное" и "серьезное" воздействие.
- **Определение ранга риска по матрице «вероятность – последствия»:** Построение матрицы рисков, где каждый риск оценивается по двум осям: вероятность и тяжесть последствий. Это позволяет ранжировать риски и определить, какие из них требуют наибольшего внимания.
- **Определение близости наступления риска:** Оценка времени, в течение которого риск может наступить. Это может включать определение рисков, которые могут возникнуть в краткосрочной, среднесрочной или долгосрочной перспективе.
- **Оценка качества использованной информации:** Оценка надежности и достоверности информации, использованной для анализа рисков. Это может включать проверку источников данных, степень неопределенности и наличие допущений.

## **Количественный анализ рисков**

**Цель:** Измерить и ранжировать риски по степени их важности.

Количественный анализ рисков предполагает использование количественных данных и методов для точного определения вероятности и последствий рисков. Основные шаги и методы включают:

### **1. Анализ чувствительности**

Анализ чувствительности – это метод, который оценивает, как изменения в одной или нескольких переменных влияют на результаты проекта. Цель анализа чувствительности – определить, какие переменные оказывают наибольшее влияние на результаты и, следовательно, на риски.

#### **Шаги анализа чувствительности:**

5. Определение ключевых переменных проекта.
6. Изменение значений этих переменных в заданных пределах.
7. Оценка влияния изменений на результаты проекта.
8. Определение наиболее чувствительных переменных, которые оказывают наибольшее влияние на результаты.

### **2. Анализ дерева решений**

Анализ дерева решений – это метод, который используется для принятия решений в условиях неопределенности. Дерево решений представляет собой графическое представление решений и их возможных последствий, включая вероятности и ценность результатов.

#### **Шаги анализа дерева решений:**

6. Определение решения или проблемы.
7. Построение дерева решений, включающего узлы решений (точки выбора) и узлы событий (точки неопределенности).
8. Присвоение вероятностей и значений каждому исходу.
9. Оценка каждого пути в дереве на основе вероятностей и ценностей.
10. Выбор оптимального решения на основе анализа дерева.

### **3. Моделирование и имитация**

Моделирование и имитация – это методы, которые используют математические модели для оценки и анализа рисков. Одним из наиболее распространенных методов является метод Монте-Карло, который позволяет моделировать различные сценарии и оценивать их вероятности и последствия.

#### **Шаги моделирования и имитации:**

6. Определение ключевых переменных и их распределений вероятностей.
7. Построение математической модели проекта или процесса.
8. Проведение большого числа симуляций с использованием случайных значений для переменных.
9. Оценка результатов симуляций для определения вероятностей и последствий различных сценариев.
10. Анализ распределения результатов для оценки рисков и принятия решений.

### **38. Идентификация рисков [3]**

### **39. Управление рисками проекта. Идентификация рисков [4]**

Управление рисками проекта включает несколько ключевых этапов, начиная с идентификации рисков. Идентификация рисков – это процесс выявления, документирования и описания рисков, которые могут повлиять на проект. Рассмотрим этот процесс подробнее:

**Идентификация рисков** – это выявление рисков, способных каким либо образом повлиять на проект, и документирование их характеристик. Это повторяющийся процесс, который периодически происходит на всех этапах проекта, поскольку в рамках его жизненного цикла могут возникнуть новые риски.

Источниками рисков могут быть:

1. База знаний организации.
2. Информация из открытых источников, научных работ, маркетинговая аналитика и другие исследовательские работы в данной области (даже форму разработчиков, которые уже сталкивались с данными проблемами).

#### **Основные шаги процесса идентификации рисков:**

##### **Сбор информации**

Для сбора информации о рисках можно использовать следующие подходы:

1. **Опрос экспертов.** Реализуется путем интервью нужных квалифицированных специалистов. Специалисты высказывают мнение о рисках и дают им качественную оценку, исходя из своих знаний, опыта и имеющейся информации. Этот метод может дать возможность избежать повторного наступления рисков.

2. **Мозговой штурм.** К участию в мозговом штурме привлекаются также квалифицированные специалисты, которым дают возможность подготовить свои суждения по определённой категории рисков. Затем проводится собрание, на котором специалисты высказывают свои мнения о рисках. Споры и замечания при таком подходе не допускаются. Все риски записываются, группируются по типам и характеристикам, каждому риску даётся определение. **Цель** – составить первичный перечень возможных рисков для последующего отбора и анализа.

**3. Метод Дельфи.** Данный подход похож на мозговой штурм, за исключением того, что опрос происходит анонимно. Далее проводится анкетирование, после чего данные обрабатываются и отсылаются повторно экспертам для дальнейшего уточнения рисков.

**4. Карточки Кроуфорда.** Собирается группа экспертов 7–10 человек. Каждому участнику опроса выдается по десять карточек. Куратор задает вопрос: "Какой риск является наиболее важным в этом проекте?" Все респонденты должны записать наиболее, по их мнению, важный риск в данном проекте. При этом никакого обмена мнениями не происходит. Ведущий делает паузу, после чего вопрос повторяется. Участник не может повторять в ответе один и тот же риск. После того как вопрос прозвучит десять раз, в распоряжении ведущего появятся от 70 до 100 карточек с ответами. Если группа с различными точками зрения, то вероятность того, что участники эксперимента укажут большинство значимых для проекта рисков, весьма высока. Остаётся составить список названных рисков и раздать его участникам для внесения изменений и дополнений.

#### • **Определение рисков:**

- **Анализ предпосылок и ограничений проекта:** Определение факторов, которые могут ограничивать или способствовать успеху проекта, таких как ограничения по времени, бюджету, ресурсам и технологиям.

**Использование контрольных списков:** Применение контрольных списков типичных рисков для проектов аналогичного типа, чтобы убедиться, что никакие важные риски не упущены.

#### • **Классификация рисков:**

- **Технические риски:** Риски, связанные с техническими аспектами проекта, включая сложности в разработке, интеграции и тестировании.

**Управленческие риски:** Риски, связанные с управлением проектом, такие как недостаток ресурсов, плохое планирование и управление временем.

**Организационные риски:** Риски, связанные с организационной структурой и культурой, включая изменения в руководстве и конфликтные приоритеты.

**Внешние риски:** Риски, возникающие из-за внешних факторов, таких как экономические условия, изменения в законодательстве и политическая нестабильность.

#### • **Документация рисков:**

- **Реестр рисков:** Создание реестра рисков, который включает описание каждого риска, его категорию, потенциальные последствия и вероятность возникновения.

**Шаблоны и формы:** Использование стандартных шаблонов и форм для документирования рисков, что обеспечивает консистентность и полноту информации.

Идентификация рисков – это важный этап в управлении рисками проекта, который помогает выявить и понять все возможные риски, которые могут повлиять на успех проекта. Правильная идентификация рисков позволяет разработать эффективные стратегии управления рисками и минимизировать их негативное влияние на проект.

## 40. **Риски [2]** (*chat gpt, если не нравится, то читать Управление рисками проекта. Основные понятия так же ответ на этот вопрос*)

Риски в проекте представляют собой потенциальные события или обстоятельства, которые могут негативно повлиять на его выполнение, достижение целей, бюджет или сроки. Рассмотрим основные типы рисков и методы их управления:

### Основные типы рисков:

1. **\*\*Технические риски:\*\*** Связаны с техническими аспектами проекта, такими как сложность технических решений, несовместимость систем и технические проблемы.
2. **\*\*Управленческие риски:\*\*** Связаны с управлением проектом, включая недостаточное планирование, неэффективное распределение ресурсов и неправильное принятие решений.
3. **\*\*Организационные риски:\*\*** Связаны с организационной структурой и культурой, включая изменения в руководстве, конфликты интересов и недостаточную поддержку со стороны стейкхолдеров.
4. **\*\*Внешние риски:\*\*** Обусловлены внешними факторами, такими как изменения в законодательстве, экономические условия, конкуренция на рынке и природные катаклизмы.

### Методы управления рисками:

1. **\*\*Идентификация рисков:\*\*** Определение и документирование потенциальных рисков, которые могут возникнуть в проекте.
2. **\*\*Оценка рисков:\*\*** Оценка вероятности возникновения рисков и их потенциального воздействия на проект.
3. **\*\*Анализ рисков:\*\*** Анализ вероятности и воздействия рисков на проект, приоритизация рисков для определения наиболее критических.
4. **\*\*Управление рисками:\*\*** Разработка и применение стратегий для снижения вероятности и воздействия рисков, а также разработка планов действий в случае реализации рисков.
5. **\*\*Мониторинг и контроль рисков:\*\*** Постоянное наблюдение за рисками, оценка эффективности принятых мер по управлению рисками и корректировка стратегий при необходимости.

### Примеры рисков:

- **\*\*Технический риск:\*\*** Непредвиденные проблемы с программным обеспечением, которые могут привести к задержкам в разработке.
- **\*\*Управленческий риск:\*\*** Недостаточное распределение ресурсов или неэффективное управление временем, что может привести к переработкам и увеличению бюджета проекта.
- **\*\*Организационный риск:\*\*** Изменения в руководстве проекта или конфликты в команде, что может замедлить прогресс и повлиять на работу команды.
- **\*\*Внешний риск:\*\*** Изменения в законодательстве, которые могут потребовать изменения в проекте и дополнительных затрат.

Управление рисками включает в себя систематический подход к выявлению, анализу и управлению рисками с целью минимизации их воздействия на проект и обеспечения его успешного завершения.



## 41. Управление рисками проекта.

## 42. Управление рисками проекта. Основные понятия [2]

*(так же ответ на Риски)*

**Риск проекта** – это неопределенное событие, которое каким-либо образом влияет на цели проекта. Например, **цель** – закрыть проект, не превышая тот бюджет, который был заложен изначально. Тогда любые события с непредвиденными расходами будут считаться риском. Или цель – создать качественный продукт быстрее конкурента. Тогда есть риск опоздать с запуском проекта или потерять своего лучшего поставщика и проиграть в качестве. Как только у проекта появляется цель, то стоит подумать о факторах, которые могут ей помешать.

**Риск** – это проблема, которая еще не существует, а проблема – это риск, который материализовался. Риск можно описать следующими свойствами:

1. **Причина или источник возникновения риска.** Явление, которое обуславливает наступление риска.
2. **Признаки риска, указание на то, что событие риска произошло или произойдет в ближайшее время.** Причина возникновения риска может быть не наблюдаема, например, заразились вирусом. Но можно наблюдать некоторые симптомы – изменение температуры тела.
3. **Последствия риска.** Проблема, которая может быть реализована в проекте в результате произошедшего риска.
4. **Влияние риска.** Влияние реализовавшегося риска на цели проекта. Воздействие обычно касается основных показателей, таких как стоимость, график и технические характеристики разрабатываемого продукта. Многие риски происходят частично и оказывают соразмерное отрицательное или положительное воздействие на проект.

**Риски можно поделить на два типа:**

1. **Прогнозируемые.** Это те риски, которые можно идентифицировать и подвергнуть анализу. В отношении таких рисков можно спланировать действия по их предотвращению и снижению последствий на проект.
2. **Непрогнозируемые.** Риски, которые невозможно идентифицировать и, следовательно, спланировать ответные действия.

Непрогнозируемые риски – это непредвиденные события. Единственное, что можно в этом случае предпринять, это создать некий резерв бюджета проекта на случай неизвестных, но потенциально возможных изменений. На расходование этого резерва руководитель проекта, как правило, обязан получать одобрение своего руководства. Управленческие резервы на непредвиденные обстоятельства не входят в основной план бюджета проекта, но включаются в бюджет проекта. Они не распределяются по проекту, как бюджет, и поэтому не учитываются при расчете освоенного объема. Согласно стандарту PMBOK управление рисками происходит в 4 этапа (Рис. 13):

1. **Идентификация.** Выявить риски, которые могут помешать целям проекта.
2. **Анализ.** Определить, какие из выявленных рисков наиболее опасны.
3. **Планирование.** Спланировать наиболее опасные риски.
4. **Мониторинг и контроль.** Поддерживать план проекта и список рисков в актуальном состоянии

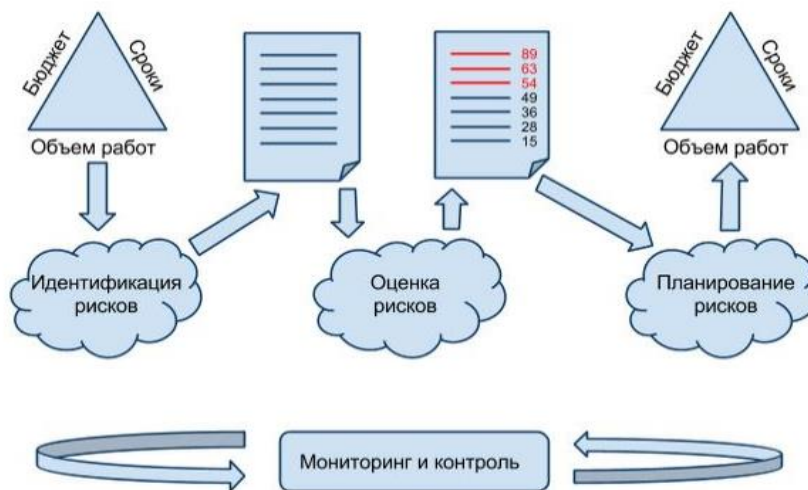


Рис. 13. – Жизненный цикл рисков проекта

## 43. Планирование рисков [2]

Фактически, на этапе планирования рисков происходит управление проектом. Для каждого риска, из списка критичных, необходимо придумать стратегию, которая обезопасит проект от этих рисков. Всего стратегий используется три:

**Transfer.** Переносим ответственность за последствия риска на третью сторону (заказчика, компанию партнера, страховую компанию и так далее). Применять эту стратегию есть смысл, если самим невозможно повлиять на риск и есть на кого эту ответственность переложить.

**Accept.** Принимаем ответственность за последствия риска на себя, но ничего не делаем, оставляем все как есть. Применять этот подход есть смысл только когда с риском ничего нельзя поделать, а делать трансфер на третью сторону неоправданно дорого.

**Mitigate.** Борьба с риском, принимая ответственность на него на себя. Для борьбы с риском нужно иметь несколько планов. Основной, для того, чтобы риск подавить, и отходной, на случай если риск всё-таки случился и влияет на проект:

- **Основной план** необходимо внедрять сразу, до того, как риск случился. Он должен понижать либо вероятность, либо последствия риска. Тут нужно использовать запись рисков в формате «причина-риск-эффект». Чтобы понизить вероятность риска, нужно бороться с его причиной. Чтобы побороть последствия, нужно защищать предмет его воздействия.
- **Отходной план** внедряется в случае, если меры по борьбе с риском не принесли результатов, риск случился и стал проблемой.

Айдын

## 44. Главные риски программных проектов и способы реагирования [5]

Риски программных проектов можно разделить на несколько категорий, каждая из которых имеет свои особенности и способы управления. Вот основные риски и методы их смягчения:

### 1. Технические риски

— Выбор неподходящей технологии, технические сложности.

## 2. Управленческие риски

— Плохое управление проектом, недостаточное планирование, слабый контроль выполнения задач.

## 3. Риски, связанные с персоналом

— Недостаток квалифицированных специалистов, конфликт в команде.

## 4. Риски, связанные с требованиями

— Изменение требований, неопределенность требований, неполные или неточные требования.

## 5. Риски, связанные с качеством

— Недостаточное тестирование, низкое качество продукта.

## 6. Внешние риски

— Изменения на рынке, законодательные изменения, природные катаклизмы.

## 7. Риски безопасности

## 8. Риски, связанные с бюджетом и сроками

**Наиболее частыми являются 1, 2, 4, 8 пункты.** Соответственно они и являются главными рисками проекта.

Эффективное управление рисками требует систематического подхода и регулярного пересмотра рисков на всех этапах жизненного цикла проекта. Фактически, на этапе планирования рисков происходит управление проектом. Для каждого риска, из списка критичных, необходимо придумать стратегию, которая обезопасит проект от этих рисков. Всего стратегий используется три:

**Transfer.** Переносим ответственность за последствия риска на третью сторону (заказчика, компанию партнера, страховую компанию и так далее). Применять эту стратегию есть смысл, если самим невозможно повлиять на риск и есть на кого эту ответственность переложить.

**Accept.** Принимаем ответственность за последствия риска на себя, но ничего не делаем, оставляем все как есть. Применять этот подход есть смысл только когда с риском ничего нельзя поделать, а делать трансфер на третью сторону неоправданно дорого.

**Mitigate.** Борьба с риском, принимая ответственность на него на себя. Для борьбы с риском нужно иметь несколько планов. Основной для того, чтобы риск подавить, и отходной, на случай если риск все-таки случился и влияет на проект:

Основной план необходимо внедрять сразу, до того, как риск случился. Он должен понижать либо вероятность, либо последствия риска. Тут нужно использовать запись рисков в формате «причина-риск-эффект». Чтобы понизить вероятность риска, нужно бороться с его причиной. Чтобы побороть последствия, нужно защищать предмет его воздействия.

Отходной план внедряется в случае, если меры по борьбе с риском не принесли результатов, риск случился и стал проблемой.

## 45. Управление командой. Организация команды [3]

Командная работа играет важнейшую роль в разработке отличного программного обеспечения и что великие команды олицетворяют понятие «мы», а не «я». Нет ничего ценнее, чем работать вместе с заинтересованными участниками команды над созданием действительно стоящего продукта.

Несмотря на общность ценностей, формулы идеальной команды не существует. Одни команды внедряют чистый Agile, другие используют MSF. Согласно классической теории необходимо, чтобы участники находились в непосредственной близости друг с другом, но иногда реалии бизнеса требуют географического распределения команды. Большинство команд имеют все необходимые навыки, но иногда для выполнения конкретной работы требуется помощь узких специалистов.

Создав команду необходимо помнить, что команды похожи на людей: для их роста нужно время. По мере своего развития команда проходит через четыре основные стадии (на рисунке).



**1. На этапе формирования** команды приходится иметь дело с совокупностью отдельных личностей. При этом происходят следующие процессы между участниками команды. Участники знакомятся с членами команды, знакомятся с лидером команды. Далее, по ходу работы у них складываются первичные мнения о других участниках и о лидере. Каждый находит «свое» место в команде и начинает развивать свои навыки.

По причине защитного механизма в поведение у человека в этот момент в команде может наблюдаться как прогресс, так и его отсутствие. На данном этапе члены команды пытаются много общаться, но могут не полностью понимать друг друга. Предпочтительный стиль лидерства на этом этапе – директивный.

На данном этапе лидер должен:

- Помогать решать проблемы, которые могут возникнуть при решении различных задач.
- Следить, что соблюдалась рабочая обстановка.
- Налаживать коммуникации между участниками команды.
- Уметь объяснить цели и обязанности участников команды.
- Поощрять активное участие в работе всех членов команды.
- Устанавливать культуру общения.

**2. На этапе притирания** члены команды осознают, что для достижения целей им предстоит пройти длинный путь. У участников возникают некоторые настроения и сомнения. Они начинают не доверять кому-то, ссориться, соперничать между собой.

Лидер на данном этапе должен выполнять функцию направления участников команды и команды в целом. Он должен участвовать в жизни команды, разрешать все проблемы и конфликты мирным способом, компромиссом. Участники должны видеть в лидере такого человека, которому можно было бы доверять

**3. На этапе стабилизации** участники команды постепенно привыкают друг к другу, появляется доверие и уважение. Общение и взаимодействие протекают более эффективно. Возникает чувство сплоченности и командный дух. Работа становится на первое место, конфликты сходят на нет.

Стиль лидерства на данном этапе – поддерживающий (не полностью ориентированный на задачи).

**4. На этапе расцвета** видна настоящая эффективность команды с творческой рабочей атмосферой. Поддерживается внутрикомандный климат. Участники поддерживают друг друга. Члены команды используют сильные и слабые стороны друг друга и чувствуют вовлеченность в командную работу. На этом этапе стиль лидерства - консультативный, сопряженный с непрерывным процессом поиском улучшения взаимодействия.

На данном этапе лидер должен:

- Совместно с членами команды определять цели, которые были бы интересны всем сотрудникам.
- Способствовать повышению эффективности команды.
- Развивать навыки команды по оцениванию своей деятельности.
- Признавать личный вклад каждого члена команды.
- Развивать каждого отдельного участника команды.

## 46. Управление командой. Принципы работы руководителя

[2]

Формирование успешной команды является основной задачей для руководителя проекта и требует наличия определенного опыта и знаний. Основные профессиональные компетенции руководителя:

1. Видение целей и пути их достижения.
2. Анализ проблем и поиск новых возможностей
3. Нацеленность на конечный результат.
4. Способность анализировать и понимать состояния членов команды.
5. Открытость в общении.
6. Умение разрешать конфликты.
7. Умение создавать творческую атмосферу и положительный микроклимат.

8. Терпимость, умение принимать людей какие они есть, учитывать их собственное мнение и право на ошибку.
9. Мотивировать к правильному профессиональному поведению членов команды.
10. Стремление выявлять индивидуальные возможности для профессионального роста каждого.
11. Способность активно «обеспечивать», «доставать», «выбивать» и т. д.

Важное качество руководителя – это умение принимать людей такими, какие они есть. Принимать их недостатки, уметь разглядеть их достоинства и использовать эти достоинства для достижения общей цели.

Для того, чтобы член команды мог успешно решить поставленную вами задачу, необходимо и достаточно выполнение четырех условий:

- 1. Понимание целей работы.** Задача лидера обеспечить общее видение целей и стратегии их достижения.
- 2. Умение ее делать.** Задача лидера – научить, быть наставником и образцом для подражания
- 3. Возможность ее сделать.** Задача лидера обеспечить для этого все необходимые условия.
- 4. Желание ее сделать.** Обеспечить адекватную мотивацию участника на протяжении всего проекта, на мой взгляд, одна из самых сложных задач лидера.

## **47. Управление командой. Состав команды [2]**

Современные команды по разработке программного обеспечения наряду с разработчиками и тестировщиками включают менеджеров по продукту, дизайнеров, маркетологов и специалистов по операциям. Обычно команды сгруппированы по трем этапам: производство, продажа и эксплуатация.

Каждый этап жизненного цикла продукта поддерживается тремя командами и образует триаду. Каждая триада характеризуется гибкостью, поскольку по мере развития продукта команды последовательно работают над каждым этапом и узнают больше как о самом продукте, так и о рынке.

Наибольшую эффективность демонстрируют небольшие, легкие на подъем команды из 5–7 человек с разноплановыми и при этом пересекающимися наборами навыков. Между участниками команды с такой структурой могут сложиться тесные, доверительные взаимоотношения; применяя разные навыки, они быстрее справляются с работой.



Триада	Кто	Ключевая деятельность
Производство	Управление	Понимание рынка, типов целевых клиентов и принципов хорошего дизайна продукта
	Дизайн	Определение предлагаемых преимуществ, целей продукта и минимально жизнеспособного продукта
	Разработка	Разработка продукта с использованием надлежащих устойчивых инженерных практик
Продажи	Управление продуктами	Понимание конкурентной среды и развития рынка продукта
	Дизайн	Позиционирование, которое подчеркивает предлагаемые преимущества продукта для каждого сегмента клиентов
	Маркетинг	Создание материалов, поддерживающих запуск продукта: веб-страниц, электронных рассылок, блогов, видеозаписей и т. д.
Эксплуатация	Управление	Регулярный выпуск программного обеспечения для клиентов
	Разработка	Реагирование на проблемы клиентов
	Поддержка и эксплуатация	Передача отзывов клиентов в производственную триаду (разработка, управление продуктами, дизайн) в качестве входных данных для будущих разработок продукта

Люди, с которыми нам приходится работать, обычно относятся к одной из двух категорий: специалисты широкого профиля и специалисты узкого профиля.

**Специалист широкого профиля** обладает широкими познаниями и может работать в разных сферах деятельности. **Специалист узкого профиля** обладает глубокими уникальными познаниями в конкретной сфере деятельности. Во многих методиках заложена идея о том, что всем участникам команды следует стать специалистами широкого профиля ради взаимозаменяемости.

## 48. Формирование команды. Правильные люди [3]

--- Про формирование команд. Так можно начинать расписывать ответ ---

Самым ценным ресурсом программного проекта являются, конечно, люди. В команде предполагается комбинация двух стилей: работа в команде и лидерство. Хорошая команда должна демонстрировать сплав самых разнообразных качеств: *профессиональные навыки, опыт, сплоченность, дух товарищества*.

**Кристаллизация.** Команда, прошедшая кристаллизацию, это группа людей, столь сильно связанных, что целое становится больше суммы составляющих его частей. Производительность этой команды выше, чем производительность тех же людей, не перешедших порог кристаллизации. И, что столь же важно, удовольствие от работы

также выше, чем можно было бы ожидать, учитывая природу работы.

Кристаллизованная команда может замечательно себя чувствовать, работая над задачей, которую другие посчитали бы скучной. Причины такого явления не столь сложны: команды по природе своей склонны формироваться при наличии целей.

#### --- Формирование команды ---

Сперва нужно определять обязанности. Поэтому лидер определяет группы разработчиков в команде и назначает в них менеджера (управляющего группой). Менеджер в ответе за свою группу, и именно он контактирует с лидером. Раздаются обязанности каждому участнику и каждый *\*занят своим делом\**.

#### --- Сам вопрос «Правильные люди» ---

Прежде всего, руководитель должен организовать правильный подбор членов команды: они могут дополнять друг друга по навыкам и опыту и должны быть совместимы друг с другом психологически. Обычно кандидаты проходят собеседование, перед тем как их примут на работу. Ценят следующие качества: 1) *опыт работы во многих аппаратно-программных средах*; 2) *знание языков программирования*; 3) *образование и опыт работы по специальности*; 4) *коммуникабельность*; 5) *способность адаптироваться*; 6) *жизненная позиция*; 7) *личностные качества*.

Образование является комплексным показателем начальных знаний и навыков кандидата, а также его способности к обучению. Опыт же характеризует конечные знания и навыки специалиста. Коммуникабельность характеризует возможности общения с коллегами, руководителями и другими заинтересованными в проекте лицами. Способность к адаптации может пояснять «послужной список» - имеющийся рабочий стаж. О жизненной позиции судить трудно, но важно. Ведь она говорит о любви к профессии и стремлении развивать знания и навыки. Личностные качества оценить, пожалуй, труднее всего. Здесь и психологический портрет, и темперамент, и инициативность, и целеустремленность, и многое другое. Именно эти качества определяют совместимость кандидата с коллективом.

Конечно, руководители должны не на словах, а на деле учитывать типичные человеческие факторы сотрудников. Люди хотят иметь интересную работу, хотят иметь возможность проявить себя, хотят, чтобы их заметили и наградили, и хотят теплых дружеских отношений в коллективе. Здоровое самоуважение является предпосылкой этих желаний. Один из источников самоуважения - качественная работа. Следовательно, сотрудники должны точно знать, что означает качество. Например, сотрудники должны знать, как оценить трудозатраты на создание хорошего продукта, как доказать себе и другим, что работа сделана корректно, и как измерить качество сделанной работы.



## 49. Формирование команды. Лидерство и управление [3]

Важно также правильно выбрать лидера команды. Он отвечает за техническое руководство или за административное управление (возможно и совмещение этих обязанностей). Лидеры должны быть в курсе повседневной деятельности команды, обеспечивая ее эффективную работу и сотрудничество с руководством проекта. Они должны ладить со всеми членами коллектива, сглаживая напряженности и устраняя неприятности.

Лидеры во многом обеспечивают сплоченность команды, помогают отдельным сотрудникам преодолевать трудности. Они воспитывают чувство единения, ответственности за работу всей команды.

В сплоченной команде возможно доминирование стиля «программирования без персонализации». При программировании без персонализации все рабочие продукты (модели, код, документация) считаются собственностью всей команды, а не отдельного сотрудника, который занимался их созданием.

Преимущества разработки без персонализации:

- упрощение процедур проверки, критики недостатков, повышение их объективности;
- поощрение стиля непринужденного обсуждения рабочих заданий, достоинств и недостатков отдельных решений;
- активизация дружеских отношений, повышение уровня искренности;
- быстрый рост мастерства (благодаря работе бок о бок);
- улучшение качества, совершенствование результатов работы.

Несомненно, опытные руководители способствуют сплоченности команды, регулярно проводя в этих целях различные совещания, собрания, устраивая совместные празднования и другие неофициальные мероприятия.

## 50. Формирование команды. Эффективное взаимодействие [3]

Для команды программного проекта просто необходима развитая система взаимодействия, иными словами, общение и хорошие средства связи между сотрудниками. Сотрудники должны информировать друг друга о ходе работы, принимаемых решениях, а также об изменениях, которые внесены в предыдущие решения. Постоянное взаимодействие тоже способствует сплоченности и повышению качества работы, поскольку сотрудники совместно обсуждают решения, начинают лучше понимать мотивацию своих коллег.

На эффективность взаимодействия влияют следующие параметры:

**Размер/структура команды.** С ростом числа участников количество связей тоже растет. Например, между 3-мя участниками есть 3 связи, 4 участника имеют 6 связей, 50 человек = 1225 связей. Вот формула :  $n(n - 1)/2$ .

Для больших команд альтернативой является их разделение на группы. Каждая группа отвечает за определенную часть проекта и работает над ней. Обычно численность группы не превышает 8 человек. В таких группах проблемы взаимодействия исчезают. Для взаимодействия с другими группами в каждой группе выделяется один сотрудник. Такая структура сохраняет преимущества небольших команд, но позволяет большому количеству людей создавать большие программные продукты.

**Иерархия команды.** Сотрудники в команде с горизонтальной организацией легче общаются между собой, чем в командах с многоуровневой организацией и иерархией отношений (начальники/подчиненные). В последних командах взаимодействие происходит между уровнями, в иерархической последовательности. Сотрудники одного уровня могут вовсе не общаться между собой. Если в такой команде сотрудники разных групп общаются только через своих менеджеров, возможны запаздывания в разработке и проблемы недопонимания.

**Рабочее окружение.** Организация рабочего места оказывает существенное влияние на возможности общения. Психологи доказали, что человек всегда предпочитает работать в отдельном помещении, которое он может оформить по своему вкусу и в котором может сосредоточиться. Исследования подтвердили - в открытом помещении сотруднику сконцентрироваться труднее, следствием чего становится снижение рабочей активности.

## 51. Формирование команды. Мотивация [4]

Мотивация команды является ключевым фактором для успешного выполнения проекта. Вот некоторые стратегии для мотивации команды:

**1. Признание и награды.** Лидер должен ценить вклад каждого разработчика. Он должен понимать какое место разработчик занимает в команде и как команда существовала бы без него. Лидер должен регулярно оценивать работу участника, и если она хороша, то поощрять успехи. И наоборот, если выполняемая работа плоха, то нужно помочь, ну или в крайнем случае уволить... В качестве награды за работу можно давать различные премии, бонусы и другие финансовые поощрения.

**2. Профессиональное развитие.** Предоставляйте возможности для повышения квалификации и профессионального роста. Предлагайте пути карьерного роста и продвижения внутри компании. Если сотрудник действительно хорош, то повышение может мотивировать, что даст толчок в производительности не только сотрудника, но и всей команды.

**3. Создание благоприятной рабочей среды. Гибкий график работы.** Лидер, если это возможно, может позволять сотрудникам выбирать удобное время работы. Данный способ позволяет сотрудникам распределять личные дела и работу. Очень важно, чтобы психическое и моральное состояние сотрудника было в норме, чтобы работа не

давила на него и не было излишней нагрузки. Также как вариант может подойти работа удаленно. Человеку, комфортнее сосредоточиться в домашней обстановке, не приходится тратить время на дорогу, соответственно и результаты работы могут улучшиться.

**4. Участие и вовлеченность.** Привлекайте сотрудников к обсуждению и принятию решений по проекту. Сотрудник должен чувствовать, что он нужен команде и должен осознавать свою роль в команде. Организуйте мероприятия для укрепления командного духа и улучшения межличностных отношений. Создайте атмосферу доверия и открытой коммуникации внутри команды.

**5. Забота о благополучии сотрудников.** Хороший лидер видит баланс между работой и личной жизнью сотрудников, что предотвращает выгорание. Отпуск и отгулы – отличный способ дать сотрудникам перерыв, в течение которого они смогут заняться своим здоровьем, семьей или просто отдохнуть.

## 52. Ключевые участники и заинтересованные стороны [4]

Есть человек «*владелец компании*». Он это тот, кому принадлежит компания. Он может как сам управлять компанией, так и может назначить руководителя компании.

**Руководитель** (т. е. лидер) является главным человеком, который управляет проектами компании. Помимо руководителя есть и другие немаловажные должности. Состав разработчиков зависит от большого числа разнообразных факторов: стиля менеджмента, принятого в организации, предметной области и размера проекта, профессиональных возможностей сотрудников организации и т. д. Перечислим типовые роли:

1. **Аналитик** отвечает за развитие и интерпретацию требований заказчика; должен быть экспертом в предметной области, но работать в тесном контакте с остальными сотрудниками.
2. **Архитектор** отвечает за проектирование и развитие архитектуры продукта, является одним из наиболее квалифицированных специалистов, имеющих опыт принятия стратегических решений. Кроме опыта проектирования, архитектор должен уметь программировать, поскольку его решения воплощаются в программном коде.
3. **Конструктор компонентов** – главный создатель компонентов (строительных кирпичиков, из которых komponуется продукт).
4. **Специалист по применению** программирует компоненты и отвечает за их сборку
5. **Специалист по повторному использованию** внедряет в продукт готовые компоненты из сторонних коммерческих библиотек.

6. **Специалист по интеграции** отвечает за сборку совместимых версий компонентов и проверку правильности их совместной работы, поддерживает выпуск версий продукта.
7. **Специалист по документации** документирует все реализованные решения, готовит документацию для пользователя.
8. **Системный программист** отвечает за создание и адаптацию программных утилит, облегчающих разработку в проекте.
9. **Системный администратор** управляет физическими компьютерными ресурсами в проекте.

Разумеется, не каждый проект требует исполнения всех этих ролей. В небольших проектах сотрудники могут играть сразу несколько ролей.

## Заинтересованные стороны

**Заказчики.** Заказчики продукта в любом случае заинтересованы, так как они будут пользоваться программой и преподносить его пользователям. Именно они определяют требования к продукту

**Пользователи.** Пользователи используют продукт. Их отношение к программе можно определить по оставляемым отзывам. Поэтому в командах обычно бывает отдел, занимающийся ответами на отзывы и передающий пожелания пользователей разработчикам.

**Руководство компании.** Руководство заинтересовано в проекте, потому что оно хочет преподнести готовый продукт в лучшем цвете. Если руководство не заинтересовано в выпускаемой продукции, то велика вероятность провальности проекта.

**Спонсоры.** Спонсорские компании могут заинтересоваться вашим проектом, если они видят, что могут получить выгоду от спонсирования.

**Инвесторы.** Инвесторы могут инвестировать в компанию, если сочтут что доходы вашей компании будут увеличиваться. Если ваши доходы увеличиваются, то скорее всего и количество инвесторов увеличится.

---

Камран

**53. Scrum. Основные понятия [4]**

**54. Scrum. Основные участники [4]**

**55. Scrum. Собrania [4]**

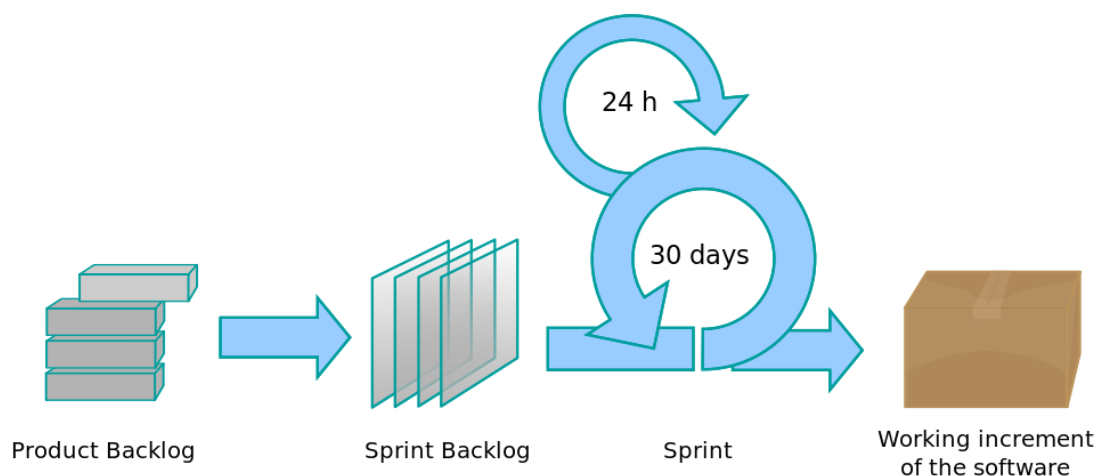
В настоящее время, Scrum является одной из наиболее популярных «методологий» разработки ПО. Согласно определению, Scrum — это каркас разработки, с

использованием которого люди могут решать появляющиеся проблемы, при этом продуктивно и производя продукты высочайшей значимости.

Это говорит о том, что в Scrum невозможно найти ответы на все вопросы и указания к действию во всех ситуациях (к примеру, в официальном описании Scrum лишь указана необходимость оценки времени, необходимой на выполнение работы, но не уточняется вид оценки. Т.е. это может быть и planning poker и другой способ оценки).

Авторами Scrum заявлены следующие особенности:

- Легкий (англ. Lightweight)
  - Понятный, доступный
  - Сложный в освоении
- (практически взаимоисключающие параграфы)



## Роли в Scrum

В классическом Scrum существует 3 базовых роли:

- Product owner*
- Scrum master*
- Команда разработки (Development team)*

*Product owner* (PO) является связующим звеном между командой разработки и заказчиком. Задача PO — максимальное увеличение ценности разрабатываемого продукта и работы команды.

Одним из основных инструментов PO является Product Backlog. Product Backlog содержит необходимые для выполнения рабочие задачи (такие как Story, Bug, Task и др.), отсортированные в порядке приоритета (срочности).

*Scrum master* (SM) является «служашим лидером» (англ. servant-leader). Задача Scrum Master — помочь команде максимизировать ее эффективность посредством устранения препятствий, помощи, обучении и мотивации команде, помощи PO

*Команда разработки* (Development team, DT) состоит из специалистов, производящих непосредственную работу над производимым продуктом. Согласно The Scrum Guide (документу, являющимся официальным описанием Scrum от его авторов), DT должны обладать следующими качествами и характеристиками:

- Быть самоорганизующейся. Никто (включая SM и PO) не может указывать команде, как им преобразовать Product Backlog в работающий продукт
- Быть многофункциональной, обладать всеми необходимыми навыками для выпуска работающего продукта
- За выполняемую работу отвечает вся команда, а не индивидуальные члены команды

Рекомендуемый размер команды — 7 (плюс-минус 2) человека. Согласно идеологам Scrum, команды большего размера требуют слишком больших ресурсов на коммуникации, в то время как команды меньшего размера повышают риски (за счет возможного отсутствия требуемых навыков) и уменьшают размер работы, который команда может выполнить в единицу времени.

## **Процесс Scrum**

Основой Scrum является Sprint, в течении которого выполняется работа над продуктом. По окончании Sprint должна быть получена новая рабочая версия продукта. Sprint всегда ограничен по времени (1-4 недели) и имеет одинаковую продолжительность на протяжении всей жизни продукта.

Перед началом каждого Sprint производится Sprint Planning, на котором производится оценка содержимого Product Backlog и формирование Sprint Backlog, который содержит задачи (Story, Bugs, Tasks), которые должны быть выполнены в текущем спринте. Каждый спринт должен иметь цель, которая является мотивирующим фактором и достигается с помощью выполнения задач из Sprint Backlog.

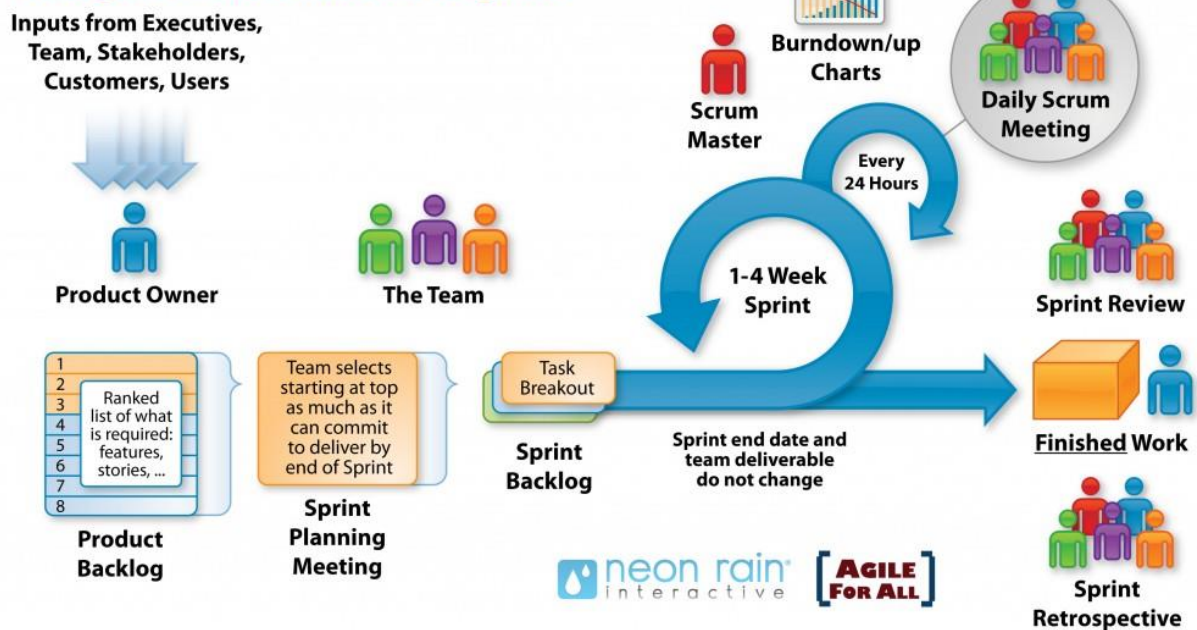
Каждый день производится Daily Scrum, на котором каждый член команды отвечает на вопросы «что я сделал вчера?», «что я планирую сделать сегодня?», «какие препятствия на своей работе я встретил?». Задача Daily Scrum — определение статуса и прогресса работы над Sprint, раннее обнаружение возникших препятствий, выработка решений по изменению стратегии, необходимых для достижения целей Sprint'a.

По окончании Sprint'a производятся Sprint Review и Sprint Retrospective, задача которых оценить эффективность (производительность) команды в прошедшем Sprint'e, спрогнозировать ожидаемую эффективность (производительность) в следующем спринте, выявлении имеющихся проблем, оценки вероятности завершения всех необходимых работ по продукту и другое.

Схематическое изображение процесса приведено на следующем рисунке:



## The Agile: Scrum Framework at a glance



### Важные, часто забываемые особенности

Часто можно услышать, что Scrum не работает, или работает хуже, чем ожидалось. Необходимо заметить, что чаще всего так происходит по одной из следующих причин:

#### 1. Scrum применяется неверно или неполностью.

Согласно авторам Scrum, эмпирический опыт является главным источником достоверной информации. Необходимость полного и точного выполнения Scrum указана в The Scrum Guide и обусловлена нетипичной организацией процесса, отсутствием формального лидера и руководителя.

#### 2. Недооценена важность работы по обеспечению мотивации команды.

Одним из основных принципов Scrum являются самоорганизующиеся, многофункциональные команды. Согласно исследованиям социологов, численность самомотивированных сотрудников, способных на самоорганизацию не превышает 15% от работоспособного населения [2]. Таким образом, лишь небольшая часть сотрудников способно эффективно работать в Scrum без существенных изменений в ролях Scrum master и Product Owner, что противоречит идеологии Scrum, и потенциально приводит к неверному или неполному использованию Scrum.

#### 3. Scrum применяется для продукта, требования к которому противоречат идеологии Scrum.

Scrum относится к семейству Agile, так Scrum приветствует изменения в требованиях в любой момент (Product backlog может быть изменен в любой момент). Это затрудняет использование Scrum в fixed-cost/fixed-time проектах. Идеология Scrum утверждает, что заранее невозможно предусмотреть все изменения, таким образом нет смысла заранее планировать весь проект, ограничившись только just-in-time планированием, т. е. Планировать только ту работу, которая должна быть выполнена в текущем Sprint. [3] Существуют и иные ограничения.

### Достоинства и недостатки

Scrum обладает достаточно привлекательными достоинствами. Scrum ориентирован на клиента, адаптивен. Scrum дает клиенту возможность делать изменения в требованиях в любой момент

времени (но не гарантирует того, что эти изменения будут выполнены). Возможность изменения требований привлекательна для многих заказчиков ПО.

Scrum достаточно прост в изучении, позволяет экономить время, за счет исключения не критичных активностей. Scrum позволяет получить потенциально рабочий продукт в конце каждого Sprint'a. Scrum делает упор на самоорганизующуюся, многофункциональную команду, способную решить необходимые задачи с минимальной координацией. Это особенно привлекательно для малых компаний и стартапов, так как избавляет от необходимости найма или обучения специализированного персонала руководителей.

Конечно, у Scrum есть и важные недостатки. Ввиду простоты и минималистичности, Scrum задает небольшое количество довольно жестких правил. Однако это вступает в конфликт с идеей клиентоориентированности в принципе, т. к. клиенту не важны внутренние правила команды разработки, особенно если они ограничивают клиента. К примеру, в случае необходимости, по решению клиента Sprint backlog может быть изменен, не смотря на явное противоречие с правилами Scrum.

Проблема является большей, чем кажется. Т.к. Scrum относится к семейству Agile, в Scrum не принято, к примеру, создание плана коммуникаций и реагирования на риски. [3] Таким образом, делая сложным или невозможным формальное (юридическое или административное) противодействие нарушениям правил Scrum.

Другой слабой особенностью Scrum является упор на самоорганизующуюся, многофункциональную команду. При кажущемся снижении затрат на координацию команды, это приводит к повышению затрат на отбор персонала, его мотивацию, обучение. При определенных условиях рынка труда, формирование полноценной, эффективной Scrum команды может быть невозможным.

## 56. Планирование управления качеством [4]

### Из ChatGPT

Планирование управления качеством (Quality Management Planning) включает в себя определение критериев качества, методов и процессов для обеспечения того, чтобы программный продукт соответствовал установленным требованиям. Этот процесс необходимо начинать с самого начала проекта и продолжать на всех его этапах. Рассмотрим ключевые компоненты плана управления качеством.

#### 1. Определение стандартов и требований

- **Идентификация стандартов:** Определите стандарты, которых должен придерживаться проект. Это могут быть отраслевые стандарты (например, ISO, IEEE), корпоративные стандарты или стандарты, специфичные для проекта.
- **Требования к качеству:** Установите чёткие требования к качеству продукта, включая функциональные и нефункциональные требования, такие как производительность, безопасность, удобство использования и т.д.

#### 2. Планирование процессов обеспечения качества

- **Процессы контроля качества:** Определите процессы, которые будут использоваться для мониторинга и оценки качества продукта. Это могут быть тестирование, код ревью, аудит процессов, оценка соответствия стандартам и т.д.



- **Методы тестирования:** Разработайте стратегию тестирования, включая виды тестирования (модульное, интеграционное, системное, приемочное), инструменты и частоту проведения тестов.

### 3. Объективная проверка соответствия

- **Проверка программных продуктов:** Регулярно проводите проверку кода, документации и других артефактов на соответствие установленным стандартам.
- **Анализ технологических операций:** Проводите аудит процессов разработки и тестирования, чтобы убедиться, что они соответствуют лучшим практикам и процедурам.

### 4. Управление отклонениями по качеству

- **Идентификация отклонений:** Регулярное проведение тестов и ревизий для обнаружения отклонений от стандартов.
- **Анализ причин отклонений:** Используйте методики анализа, такие как "5 Почему" или диаграмма Исикавы, для определения корневых причин отклонений.
- **Корректирующие меры:** Разрабатывайте и внедряйте меры для устранения выявленных проблем.
- **Контроль исполнения и эффективности:** Постоянный мониторинг внедренных мер для обеспечения их эффективности и предотвращения повторного возникновения проблем.

### 5. Представление информации высшему руководству

- **Независимая оценка:** Представляйте высшему руководству независимую информацию о значительных несоответствиях, которые не удалось устранить на уровне проекта.
- **Повышение прозрачности:** Обеспечьте прозрачность процессов управления качеством и информируйте руководство о прогрессе и возникающих проблемах.

### 6. Обучение и развитие команды

- **Тренинги и семинары:** Организуйте обучение для команды по вопросам обеспечения качества, методам тестирования, использованию инструментов и т.д.
- **Обмен знаниями:** Поощряйте обмен знаниями и лучшими практиками внутри команды для повышения общего уровня компетенции.

### 7. Документирование и отчетность

- **Документация планов и процессов:** Ведите документацию всех процессов и планов по управлению качеством, чтобы обеспечить прозрачность и возможность их воспроизведения.
- **Отчетность по качеству:** Регулярно создавайте отчеты о состоянии качества продукта и эффективности применяемых мер для предоставления их заинтересованным сторонам.

## Заключение

Планирование управления качеством является неотъемлемой частью успешного выполнения программного проекта. Оно включает определение стандартов и требований, планирование процессов обеспечения качества, управление отклонениями и представление информации высшему руководству. Включение всех этих аспектов в план проекта помогает достигать высокого уровня качества и успешного завершения проекта.

## 57. Планирование управления конфигурациям [2]

### Из книги

Конфигурационное управление один из важных процессов производства программного обеспечения. Об этой области знаний написана не одна книга. Мы будем говорить только о том, что эта работа должна быть спланирована. План проекта должен включать в себя работы по обеспечению единого хранилища всей проектной документации и разрабатываемого программного кода, обеспечению сохранности и восстановлению проектной информации после сбоя. Работы по настройке рабочих станций и серверов, используемых участниками проектной команды, тоже должны войти в план. Кроме этого в плане должны содержаться работы, необходимые для организации сборки промежуточных выпусков системы, а также ее конечного варианта. Эти работы, как правило, выполняет один человек – инженер по конфигурациям. Если проект небольшой, то эта роль может быть дополнительной для одного из программистов. Я как-то видел, что эту роль выполнял менеджер проекта. «Размазывать» эту работу на всех участников проекта, во-первых, неэффективно. Установка и конфигурирование среды разработки, например, баз данных и серверов приложений, требует определенных компетенций и знаний особенностей конкретных версий продуктов. Если эти навыки придется осваивать всем разработчикам, то на это

уйдет слишком много рабочего времени. Во-вторых, «размазывание» работ по управлению конфигурациями может привести к коллективной безответственности, когда никто не знает, от чего не собирается проект и как откатиться к консистентной версии. Управление конфигурациями может многократно усложниться, если проектной команде параллельно с разработкой новой функциональности продукта приходится поддерживать несколько релизов этого продукта, которые были установлены ранее у разных клиентов. Все эти работы должны быть учтены в плане проекта.

### Из ChatGPT

Управление конфигурациями (Configuration Management, CM) является важным процессом в производстве программного обеспечения, направленным на обеспечение целостности и контроля изменений в проекте. Этот процесс включает в себя множество задач, которые должны быть тщательно спланированы и выполнены.

#### Ключевые компоненты плана управления конфигурациями:

- Единое хранилище проектной документации и программного кода:**
  - Создание и поддержание репозитория, в котором будет храниться весь исходный код, документация и другие артефакты проекта.
  - Обеспечение доступности хранилища для всех членов команды.
  - Использование систем управления версиями (VCS) таких как Git, SVN и других для отслеживания изменений в коде и документации.
- Сохранность и восстановление проектной информации:**
  - Регулярное создание резервных копий (backup) всей проектной информации.
  - Определение процедур восстановления данных после сбоя, включая тестирование этих процедур для обеспечения их надежности.
  - Хранение резервных копий в надежных и защищенных местах, возможно, на удаленных серверах или в облачных хранилищах.
- Настройка рабочих станций и серверов:**
  - Установка и конфигурирование среды разработки, включая операционные системы, базы данных, серверы приложений и другие необходимые компоненты.
  - Создание и поддержание документации по установке и настройке среды разработки, чтобы упростить процесс для новых членов команды.
  - Обеспечение автоматизации этих процессов с помощью инструментов, таких как Ansible, Chef, Puppet или других.
- Организация сборки промежуточных и конечных выпусков системы:**
  - Определение процессов и инструментов для автоматической сборки (build) и тестирования (continuous integration).

- Настройка систем непрерывной интеграции (CI) и непрерывного развертывания (CD), таких как Jenkins, Travis CI, GitLab CI/CD или других.
- Определение критериев для промежуточных и конечных выпусков, включая проведение тестирования и верификации перед выпуском.

#### 5. Роль инженера по конфигурациям:

- Назначение ответственного лица (инженера по конфигурациям) для выполнения всех задач, связанных с управлением конфигурациями.
- В небольших проектах, где выделение отдельного инженера может быть неоправданным, эту роль может выполнять один из программистов или менеджер проекта. Однако важно, чтобы обязанности были четко определены и понятны.

#### 6. Управление параллельными релизами и поддержка нескольких версий продукта:

- Создание веток в системе управления версиями для разработки новой функциональности и поддержки предыдущих релизов.
- Определение стратегий слияния (merge) и устранения конфликтов при работе с различными ветками.
- Обеспечение возможности быстрого переключения между версиями продукта и их развертывания у клиентов.

## Заключение

Планирование управления конфигурациями играет критически важную роль в успешной реализации программных проектов. Правильное управление конфигурациями обеспечивает стабильность и контроль над изменениями в проекте, помогает предотвратить потерю данных и обеспечивает возможность быстрого восстановления после сбоев. Включение всех вышеописанных аспектов в план проекта позволяет минимизировать риски и повысить эффективность работы проектной команды.

## 58. Планирование управления содержанием [2]

### Из книги

Одна из распространенных «болезней» программных проектов называется «ползучий фичеризм». Это, когда к изначально спроектированной будке для любимой собаки сначала пристраивают сарайчик для хранения садового инвентаря, а потом и домик в несколько этажей для ее хозяина. И все это пытаются построить на одном и том же фундаменте и из тех же самых материалов. Эта болезнь стала причиной летального исхода многих проектов разработки программного обеспечения. Поэтому, сразу, как только удалось стабилизировать и согласовать иерархическую структуру работ, необходимо разработать план управления содержанием проекта. Для этого следует:

1. Определить источники запросов на изменение.
2. Установить порядок анализа, оценки и утверждения или отклонения изменения содержания.
3. Определить порядок документирования изменений содержания.
4. Определить порядок информирования об изменении содержания.

Первая задача, которую необходимо решить при анализе запроса на изменения - выявить объекты изменений: требования, архитектура, структуры данных, исходные коды, сценарии тестирования, пользовательская документация, проч. Затем требуется спроектировать и детально описать изменения во всех выявленных объектах. И наконец, следует оценить затраты на внесение изменений, тестирование изменений и регрессионное тестирование продукта и их влияние на сроки проекта. Эта работа, которая потребует затрат рабочего времени и порой значительных разных специалистов: аналитиков, проектировщиков, разработчиков, тестировщиков и менеджера проекта. Поэтому эта работа должна обязательно быть учтена в плане.

Из ChatGPT(пришлось использовать так как в книге ничего нет)

## Планирование управления содержанием проекта

"Ползучий фичеризм" или "feature creep" - распространенная проблема в программных проектах, когда к изначально запланированному содержанию добавляются новые требования и функции, что часто приводит к серьезным проблемам. Чтобы избежать этого, необходимо разработать четкий и структурированный план управления содержанием проекта.

### Шаги для разработки плана управления содержанием проекта:

#### 1. Определение источников запросов на изменение

- Идентифицировать все возможные источники запросов на изменение содержания проекта. Это могут быть заказчики, пользователи, члены проектной команды, внешние консультанты и т.д.
- Установить процедуры для формализации запросов на изменение, включая использование стандартных форм и шаблонов для подачи запросов.

#### 2. Установление порядка анализа, оценки и утверждения или отклонения изменения содержания

- Создать комитет по управлению изменениями (Change Control Board, CCB), который будет отвечать за рассмотрение всех запросов на изменения.
- Разработать критерии для оценки изменений, включая их влияние на требования, архитектуру, данные, исходный код, тестовые сценарии и документацию.
- Определить процедуру принятия решений по изменениям, включая этапы анализа, обсуждения и голосования.

#### 3. Определение порядка документирования изменений содержания

- Разработать систему документирования всех изменений содержания. Это включает в себя ведение журнала изменений, который должен содержать информацию о каждом запросе на изменение, его статусе и принятом решении.
- Обеспечить доступность документации для всех заинтересованных сторон проекта.

#### 4. Определение порядка информирования об изменении содержания

- Установить процедуры для информирования всех заинтересованных сторон о принятых изменениях. Это может включать регулярные обновления статуса проекта, рассылки по электронной почте, обновления на внутренних порталах и т.д.
- Обеспечить, чтобы вся информация была актуальной и точной, и чтобы все участники проекта были осведомлены о текущем состоянии содержания.

## Анализ и проектирование изменений

#### 1. Выявление объектов изменений

- При анализе запроса на изменение необходимо определить, какие объекты проекта будут затронуты. Это могут быть:
  - Требования
  - Архитектура
  - Структуры данных
  - Исходный код
  - Сценарии тестирования
  - Пользовательская документация
  - Другие компоненты проекта

#### 2. Проектирование и описание изменений

- Спроектировать и детально описать необходимые изменения для всех выявленных объектов. Включить в описание спецификации изменений, диаграммы, схемы и другие документы, которые помогут реализовать изменения.
- Обеспечить, чтобы все изменения были совместимы с текущей архитектурой и структурой проекта.

#### 3. Оценка затрат и влияния на сроки проекта

- Оценить затраты на внедрение изменений, включая затраты на проектирование, разработку, тестирование и документирование изменений.
- Оценить влияние изменений на сроки выполнения проекта, включая возможные задержки и изменения в графике.

## **Включение работы по управлению изменениями в план проекта**

### **1. Ресурсы**

- Определить и выделить необходимые ресурсы для выполнения работы по управлению изменениями, включая аналитиков, проектировщиков, разработчиков, тестировщиков и менеджера проекта.

### **2. График**

- Включить все задачи по управлению изменениями в общий график проекта, учитывая их влияние на общие сроки проекта.

### **3. Контроль**

- Установить процедуры для регулярного контроля и отчета о выполнении задач по управлению изменениями.

## **Заключение**

Эффективное управление содержанием проекта требует четко структурированного плана, который включает определение источников запросов на изменения, процедуры анализа и утверждения изменений, документирование и информирование всех заинтересованных сторон. Важно учитывать влияние изменений на весь проект и включать эти работы в общий план и график проекта, чтобы избежать "ползучего фичеризма" и обеспечить успешное завершение проекта.

# **59. Планирование качеством проекта**

## **Из книги**

Обеспечение качества еще одна из базовых областей знаний в программной инженерии. Относительно того, что такое качество ПО и как его эффективно обеспечивать, можно рассуждать очень и очень долго. В нашем курсе мы ограничимся утверждением о том, что обеспечение качества – важная работа, которая должна быть спланирована заранее и выполняться по ходу всего программного проекта, а не только во время приемосдаточных испытаний. При планировании этой работы необходимо понимать, что продукт проекта не должен обладать наивысшим возможным качеством, которое недостижимо за конечное время. Необходимое качество продукта определяется требованиями к нему. Основная задача обеспечения качества – это не поиск ошибок в готовом продукте (выходной контроль) а их предупреждение в процессе производства. Для примера, гладкость обработки детали на токарном станке только случайно может оказаться соответствующей требуемому качеству в 1 микрон, если шпиндель, в котором крепится деталь, плохо центрован.

План управления качеством должен включать в себя следующие работы:

1. Объективную проверку соответствия программных продуктов и технологических операций применяемым стандартам, процедурам и требованиям.
2. Определение отклонений по качеству, выявление их причин, применение мер по их устранению, а также контроль исполнения принятых мер и их эффективности.
3. Представление высшему руководству независимой информации о несоответствиях, не устраняемых на уровне проекта.

Помимо перечисленных разделов план проекта должен включать:

1. План управления рисками.
2. Оценку трудоемкости и сроков работ.

Планирование качества проекта – это важный и многогранный процесс, который направлен на обеспечение того, что программный продукт соответствует установленным требованиям и стандартам. Он должен начинаться с самого начала проекта и продолжаться на всех его этапах. Рассмотрим ключевые компоненты плана управления качеством:

## 1. Объективная проверка соответствия

Объективная проверка включает несколько аспектов:

- **Программные продукты:** Проверка кода, документации и других артефактов на соответствие установленным стандартам.
- **Технологические операции:** Анализ процессов разработки и тестирования, чтобы убедиться, что они выполняются в соответствии с лучшими практиками и процедурами.

## 2. Управление отклонениями по качеству

Необходимые действия при управлении отклонениями:

- **Идентификация отклонений:** Регулярное проведение тестов и ревизий для обнаружения отклонений от стандартов.
- **Анализ причин:** Использование методик анализа, таких как "5 Почему" или диаграмма Исикавы, для определения корневых причин отклонений.
- **Корректирующие меры:** Разработка и внедрение мер для устранения выявленных проблем.
- **Контроль исполнения и эффективности:** Непрерывный мониторинг внедренных мер, чтобы убедиться в их эффективности и предотвратить повторное возникновение проблем.

## 3. Представление информации высшему руководству

Высшее руководство должно быть информировано о значительных несоответствиях, которые не удалось устранить на уровне проекта. Это обеспечивает:

- **Независимую оценку:** Руководство получает независимую информацию, что помогает в принятии объективных решений.
- **Повышение прозрачности:** Это способствует улучшению процессов и предотвращению повторения ошибок.

## Дополнительные компоненты плана проекта

Помимо плана управления качеством, проект должен включать:

### 1. План управления рисками

Этот план включает:

- **Идентификация рисков:** Выявление потенциальных рисков на различных этапах проекта.
- **Анализ рисков:** Оценка вероятности и воздействия каждого риска.
- **Разработка мер по снижению рисков:** Планирование действий по уменьшению вероятности или воздействия рисков.
- **Мониторинг рисков:** Постоянное отслеживание и пересмотр рисков в ходе выполнения проекта.

### 2. Оценка трудоемкости и сроков работ

Для точного планирования необходимо:

- **Оценка ресурсов:** Определение необходимых человеческих и материальных ресурсов.
- **Оценка времени:** Расчет времени, необходимого для выполнения каждой задачи.
- **График выполнения:** Создание временного графика, который включает все этапы и задачи проекта.

## Заключение

Планирование качества проекта является критически важным для обеспечения того, что программный продукт будет соответствовать всем установленным требованиям и стандартам. Включение управления качеством, управления рисками и оценки трудоемкости и сроков в план проекта помогает достигать высокого уровня качества и успешного завершения проекта.

## 60. Планирование организационной структуры [2]

### Из книги

Организационная структура – это согласованное и утвержденное распределение ролей, обязанностей и целей деятельности ключевых участников проекта. Она в обязательном порядке должна включать в себя систему рабочих взаимоотношений между рабочими группами проекта, систему отчетности, оценки хода выполнения проекта и систему принятия решений. Следует помнить, что организационная структура проекта – «живой» организм. Она начинает складываться на стадии планирования и должна меняться по ходу проекта. Нестабильность организационной структуры – частая смена исполнителей – может стать серьезной проблемой в управлении проектом, поскольку, существует цена замены, которая определяется временем вхождения нового участника в контекст проекта

### Из ChatGPT

## Планирование организационной структуры проекта

Организационная структура проекта представляет собой согласованное и утвержденное распределение ролей, обязанностей и целей деятельности ключевых участников проекта. Эта структура включает систему рабочих взаимоотношений между рабочими группами проекта, систему отчетности, оценки хода выполнения проекта и систему принятия решений. Организационная структура проекта является динамичной и может изменяться в ходе проекта, чтобы соответствовать изменяющимся требованиям и условиям.

## Шаги для разработки организационной структуры проекта

### 1. Определение ключевых ролей и обязанностей

- **Руководитель проекта:** Ответственен за общее руководство проектом, принятие ключевых решений, управление ресурсами и взаимодействие с заинтересованными сторонами.
- **Команда управления проектом:** Включает менеджеров по отдельным направлениям (технический менеджер, менеджер по качеству, менеджер по рискам и т.д.), которые поддерживают руководителя проекта в управлении специализированными аспектами проекта.
- **Команды разработки:** Группы разработчиков, тестировщиков, архитекторов, аналитиков и других специалистов, которые выполняют конкретные задачи проекта.
- **Заинтересованные стороны:** Заказчики, пользователи, спонсоры и другие внешние участники, которые имеют интерес в успешном завершении проекта.

### 2. Определение системы рабочих взаимоотношений

- **Взаимодействие между группами:** Определить, как различные рабочие группы будут взаимодействовать друг с другом. Это включает регулярные встречи, совещания, рабочие группы и совместные семинары.
- **Коммуникационные каналы:** Установить четкие коммуникационные каналы для обмена информацией между командами и участниками проекта. Это может включать использование систем управления проектами, электронной почты, мессенджеров и других инструментов.

### 3. Система отчетности и оценки хода выполнения проекта

- **Регулярные отчеты:** Установить график регулярных отчетов о ходе выполнения проекта. Отчеты могут быть еженедельными, ежемесячными или по завершении ключевых этапов проекта.
- **Показатели успеха:** Определить ключевые показатели успеха (KPI), которые будут использоваться для оценки прогресса проекта. Это могут быть сроки выполнения, качество продукции, бюджет и другие важные метрики.
- **Аудиты и ревизии:** Периодически проводить внутренние и внешние аудиты для оценки состояния проекта и соответствия его целям.

### 4. Система принятия решений

- **Процедуры принятия решений:** Разработать процедуры для принятия решений, включая механизмы голосования, утверждения и документирования решений.
- **Комитеты и группы:** Создать специализированные комитеты или группы для принятия решений по определенным аспектам проекта, например, комитет по управлению изменениями или комитет по рискам.

### Управление изменениями в организационной структуре

- **Гибкость:** Обеспечить, чтобы организационная структура оставалась гибкой и могла адаптироваться к изменениям в проекте. Это включает в себя возможность пересмотра ролей и обязанностей, а также изменения состава команд.
- **Процедуры внесения изменений:** Разработать процедуры для внесения изменений в организационную структуру. Это должно включать оценку необходимости изменений, обсуждение предложений и формальное утверждение изменений.
- **Документирование изменений:** Вести документацию всех изменений в организационной структуре, чтобы сохранить историю изменений и обеспечить прозрачность.

### Управление заменой участников проекта

- **План замены:** Разработать план на случай замены ключевых участников проекта. Это включает в себя процесс найма или назначения новых сотрудников, их обучение и ввод в контекст проекта.
- **Поддержка новых участников:** Обеспечить поддержку новых участников в их адаптации к проекту. Это может включать наставничество, обучение и доступ к необходимой документации и ресурсам.
- **Минимизация воздействия на проект:** Стремиться минимизировать влияние замены участников на ход выполнения проекта, планируя замены заранее и обеспечивая плавный переход.

### Заключение

Эффективное планирование организационной структуры проекта является ключевым фактором успеха. Это требует четкого определения ролей и обязанностей, установления систем взаимодействия, отчетности и принятия решений, а также управления изменениями в структуре.



Стабильность и гибкость организационной структуры помогают обеспечить успешное выполнение проекта в соответствии с его целями и требованиями.