Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра инженерной физики

УПРАВЛЕНИЕ IT-ПРОЕКТАМИ

Курс лекций

Витебск
ВГУ имени П.М. Машерова
2021

УДК 005.8:004(075.8)
ББК 65.291.217с51я73
У67

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 1 от 27.10.2021.

Составитель: заведующий кафедрой инженерной физики ВГУ имени П.М. Машерова, кандидат физико-математических наук **А.И. Никитин**

Р е ц е н з е н т ы :
заведующий кафедрой информационных систем
и автоматизации производства УО «ВГТУ», кандидат технических наук,
доцент *В.Е. Казаков;*
доцент кафедры информационных технологий и управления бизнесом
ВГУ имени П.М. Машерова, кандидат биологических наук,
доцент *А.А. Чиркина*

Управление IT-проектами : курс лекций / сост. А.И. Никитин. – Витебск : ВГУ имени П.М. Машерова, 2021. – 66 с.
У67 ISBN 978-985-517-857-7.

Предлагаемый курс лекций содержит описание основных подходов, используемых в управлении проектами в IT-отрасли. Приводятся различные методологии и подходы к управлению проектами, описываются особенности их реализаций, а также решения, которые находят применение в каждом конкретном случае. Описаны этапы управления проектами и риски, которые могут возникнуть при их реализации.

Предназначается для студентов специальности «Управление информационными ресурсами» (дисциплины «Проектный менеджмент в IT-сфере», «Управление IT-проектами»).

УДК 005.8:004(075.8)
ББК 65.291.217с51я73

ISBN 978-985-517-857-7

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данный курс лекций содержит основные подходы и методики управления проектами по разработке программного обеспечения. В каждой лекции изложен материал, сопровождающийся примерами использования различных методик, а также описаны риски, которые могут возникнуть непосредственно в процессе реализации управления. В конце каждой лекции приведен ряд контрольных вопросов, которые помогают студентам закрепить полученные знания, а также дают понять, в каком направлении стоит расширять свои навыки по данному направлению.

Материал лекций основан на практиках крупных IT-компаний, которые имеют достаточно большой опыт в управление разработкой программного обеспечения. Представлены различные методологии, используемые в таких компаниях, как Microsoft, IBM, Amazon, Atlassian и др.

Особое внимание уделяется методологиям, основанных на «гибком управлении». Подчеркивается тот факт, что существует множество методик, в основе которых лежат ценности и принципы гибкого управления. Более подробно рассмотрен фреймворк Scrum как одна из самых популярных методик для управления проектами по разработке программного обеспечения.

Лекции будут полезны студентам, которые собираются профессионально заниматься разработкой программного обеспечения. Приходя в промышленную разработку программного обеспечения, выпускник долго обучается тому, что и как надо на работе делать. И еще больше времени у него уходит на понимание того, почему надо делать именно так, а не иначе.

Конспект лекций также направлен на начинающих руководителей, так как поможет освоить анализ рисков, планирование и контролирование проектных работ, и, что самое главное, научит понимать людей, эффективно взаимодействовать с ними, разрешать конфликты и обеспечивать адекватную мотивацию продуктивной работы.

Хочется отметить, что курс полезен так для начинающих управленцев проектов и в других областях. Он поможет им понять, что в разработке программного обеспечения есть свои особенности, по сравнению с другими производствами, которые необходимо знать и понимать для того, чтобы более осмысленно общаться со своими подчиненными.

Материал соответствует отдельным темам рабочих программ курсов «Проектный менеджмент в IT-сфере», «Управление IT-проектами» (специальность «Управление информационными ресурсами»).

Лекция 1
ИСТОРИЯ РАЗВИТИЯ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1. История развития и основные термины

Программная инженерия есть применение определенного систематического измеримого подхода при разработке, эксплуатации и поддержке программного обеспечения [3].

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 результате был выпущен стандарт ISO/IEC 12207. В 2004 году в отрасли был создан основополагающий труд «Руководство к своду знаний по программной инженерии» (SWEBOK) [1], в котором были собраны основные теоретические и практические знания, накопленные в этой отрасли.

В дальнейшем нам понадобится ряд определений, которые будут активно использоваться на протяжении курса лекций.

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, попрежнему, 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, когда сначала цели отображаются на требования к системе, требования – на высокоуровневую архитектуру и спецификации компонентов, спецификации - на дизайн компонентов, дизайн - на исходный код. Далее исходный код при помощи компиляторов и сборщиков отображается на код развертывания, код развертывания – на вызовы функций программного обеспечения окружения (операционные системы, промежуточное программное обеспечение, базы данных), которое может

располагаться на множестве компьютеров, объединенных в сеть, и только после этого – в машинные команды и данные.

Профессиональное программирование (синоним производство программ) – 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, но об этом позже.

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.

Процесс разработки программного обеспечения – совокупность процессов, обеспечивающих создание и развитие программного обеспечения.

Самый распространенный процесс разработки программного обеспечения, который используется как ни странно достаточно распространен в любой отрасли, можно назвать «как получится». Это не означает, что процесса как такового нет. Он есть и, как правило, обеспечивает разработку программного обеспечения при приемлемых затратах и качестве, но этот процесс не документирован, держится на людях и передается из поколения

в поколение. Целенаправленная работа по оценке эффективности и улучшению процесса не ведется.

Модель процесса разработки программного обеспечения – формализованное представление процесса разработки программного обеспечения. Часто при описании процессов вместо слова модель употребляется термин методология, что приводит к неоправданному расширению данного понятия.

1.2. SWEBOK

В конце 90-х годов прошлого века знания и опыт, которые были накоплены в индустрии программного обеспечения за предшествующие 30-35 лет, а также более чем 15-летних попыток применения различных моделей разработки, все это, наконец, оформилось в то, что принято называть дисциплиной программной инженерии – Software Engineering. В какой-то мере, такое формирование дисциплины на основе широко распространенного практического опыта напоминает те процессы, которые происходили в управлении проектами. Возникали и развивались профессиональные ассоциации, специализированные институты, комитеты по стандартизации и другие образования, которые, в конце концов, пришли к общему мнению о необходимости сведения профессиональных знаний по соответствующим областям и стандартизации соответствующих программ обучения.

С 1993 года IEEE и ACM координируют свои работы в рамках специального совместного комитета – Software Engineering Coordinating Committee. Проект SWEBOK был инициирован этим комитетом в 1998 году. Оцененный предположительный объем содержания SWEBOK и другие факторы привели к тому, что было рекомендовано проводить работы по реализации проекта не только силами добровольцев из рядов экспертов индустрии и представителей крупнейших потребителей и производителей программного обеспечения, но и на основе принципа “полной занятости”. Базовый комплекс работ, в соответствии со специальным контрактом, был передан в Software Engineering Management Research Laboratory Университета Квебек в Монреале. Среди компаний, поддержавших этот уникальный проект были Boeing, MITRE, Raytheon, SAP. В результате проекта, осуществленного при финансовой поддержке этих и других компаний и организаций, а также с учетом его значимости для индустрии, SWEBOK Advisory Committee принял решение сделать SWEBOK 2004 trial edition общедоступной. В перспективе, в зависимости от финансирования, SWAC считал необходимым законченную версию SWEBOK (изначально планировалось, что она будет готова в 2008 году) сделать также свободно доступной на Web-сайте проекта. Сегодняшняя публичность результатов проекта стала возможна, в первую очередь, именно благодаря поддержке

SWEBOK Industrial Advisory Board – структуры, объединяющей представителей компаний, поддержавших проект.

Проект SWEBOK планировался в виде трех фаз: *Strawman, Stoneman* и *Ironman*. К 2004 году была выпущена версия руководства по «своду знаний» третьей фазы – Ironman, то есть максимально приближенная к окончательному варианту и одобренная IEEE в феврале 2005 года к публикации в качестве пробной версии. Основная цель SWEBOK 2005 – улучшить представление, целостность и полезность материала руководства на основе сбора и анализа откликов на данную версию с тем, чтобы выпустить финальную редакцию документа в 2008 году. Однако версии 2008 не появилось, хотя ряд дополнений на начало 2010 года и находится уже в виде драфтов, что не исключает расширения SWEBOK в ближайшей перспективе и, в то же время, не принижает значимости уже выпущенной версии 2004.

По ряду обоснованных причин, SWEBOK является достаточно консервативным. После 6 лет непосредственных работ над документом, SWEBOK включал лишь 10 областей знаний. При этом, что справедливо и для PMBOK, добавление новых областей знаний в SWEBOK достаточно прозрачно. Все, что для этого необходимо, зрелость (или, по крайней мере, явный и быстрый процесс достижения зрелости) и доступность соответствующей области знаний, если это не приведет к серьезному усложнению SWEBOK.

Важно понимать, что программная инженерия является развивающейся дисциплиной. Более того, данная дисциплина не касается вопросов конкретизации применения тех или иных языков программирования, архитектурных решений или, тем более, рекомендаций, касающихся более или менее распространенных, или развивающихся с той или иной степенью активности и заметности технологий. Руководство к своду знаний, каковым является SWEBOK, включает базовое определение и описание областей знаний и, безусловно, является *недостаточным* для охвата всех вопросов, относящихся к вопросам создания программного обеспечения, но, в то же время *необходимым* для их понимания.

Необходимо отметить, что одной из важнейших целей SWEBOK является именно определение и систематизация тех аспектов деятельности, которые составляют суть профессии инженера-программиста.

Описание областей знаний в SWEBOK построено по иерархическому принципу, как результат структурной декомпозиции. Такое иерархическое построение обычно насчитывает два–три уровня детализации, принятых для идентификации тех или иных общепризнанных аспектов программной инженерии. При этом, структура декомпозиции областей знаний детализирована только до того уровня, который необходим для понимания природы соответствующих тем и возможности нахождения источников компетен-

ции и других справочных данных и материалов. В принципе, считается, что как таковой «свод знаний» по программной инженерии представлен не в обсуждаемом руководстве (SWEBOK), а в первоисточниках [1].

SWEBOK описывает 10 областей знаний (Рис. 1):

1. Software requirements – программные требования.
2. Software design – дизайн (архитектура).
3. Software construction – конструирование программного обеспечения.
4. Software testing – тестирование.
5. Software maintenance – эксплуатация (поддержка) программного обеспечения.
6. Software configuration management – конфигурационное управление.
7. Software engineering management – управление в программной инженерии.
8. Software engineering process – процессы программной инженерии.
9. Software engineering tools and methods – инструменты и методы
10. Software quality – качество программного обеспечения.

SWEBOK также включает обзор смежных дисциплин, связь с которыми является важной для программной инженерии:

1. Computer engineering – разработка компьютеров
2. Computer science – информатика;
3. Management – общий менеджмент;
4. Mathematics – математика;
5. Project management – управление проектами;
6. Quality management – управление качеством;
7. Systems engineering – системное проектирование.

Стоит отметить, что принятые разграничения между областями знаний, их компонентами и другими элементами достаточно произвольны. При этом, в отличие от PMBOK, области знаний SWEBOK не включают «входы» и «выходы». В определенной степени такая декомпозиция связаны с тем, что SWEBOK не ассоциирован с той или иной моделью (например, жизненного цикла) или методом. Хотя на первый взгляд первые пять областей знаний в SWEBOK представлены в традиционной последовательной модели, это не более чем следование принятой последовательности освещения соответствующих тем. Остальные области и структура декомпозиции областей представлены в алфавитном порядке.

Все это необходимо знать и уметь применять, для того чтобы разрабатывать программное обеспечение. Как видим, управление проектами, о котором будет вестись разговор далее, лишь одна из 17 областей знаний программной инженерии, и то вспомогательная. Однако основной причиной большинства провалов программных проектов является именно применение неадекватных методов управления разработкой.

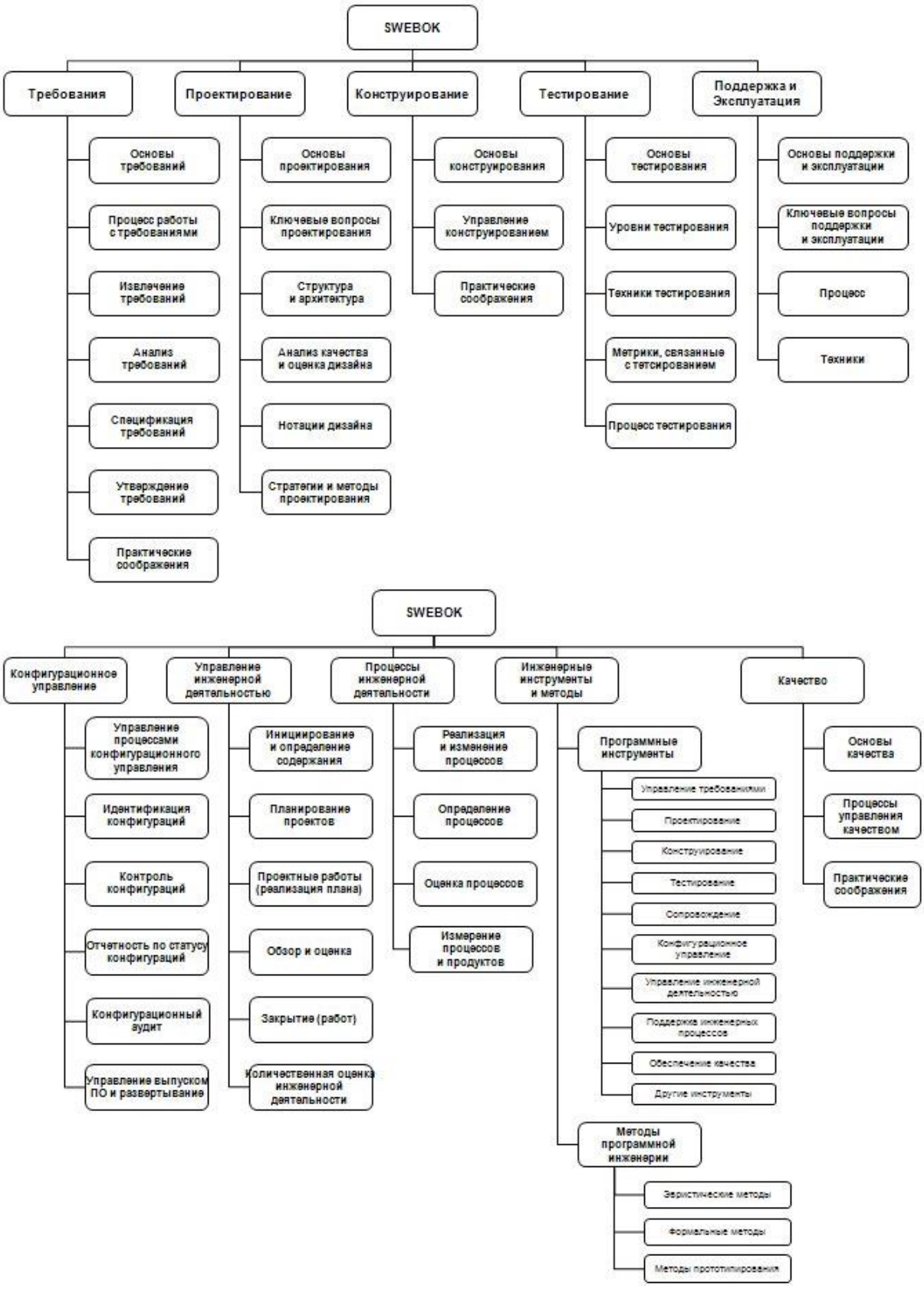

Рис. 1. **– Области знаний SWEBOK**

1.3. Отличие разработки программного обеспечения от других отраслей

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.

Кто в этом виноват? Никто. Как никто не виноват в том, что тучи закрывают солнце, что на улице идет снег, что дует ветер. Поскольку самого-то «хаоса» не было, и нет, а есть лишь Богом данная (для атеистов – объективная) реальность, которая заключена в особой специфике производства программ, по сравнению с любой другой производственной деятельностью, потому что то, что производят программисты – нематериально, это коллективные ментальные модели, записанные на языке программирования. И с этой спецификой придется считаться, если, конечно, не хотим «дуть против ветра».

«Программист, подобно поэту, работает почти непосредственно с чистой мыслью. Он строит свои замки в воздухе и из воздуха, творя силой воображения. Трудно найти другой материал, используемый в творчестве, который столь же гибок, прост для шлифовки или переработки и доступен для воплощения грандиозных замыслов», - писал Ф. Брукс в 1975 году.

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. Просто в силу уникальности отрасли опыт профессионалов, накопленный в материальном производстве и изложенный в стандарте PMI PMBOK[2] – своде знаний по управлению проектами, мало способствует успеху в управлении программным проектом. Управлять разработкой программного обеспечения чаще всего необходимо по-другому.

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 искушенному 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, чтобы удерживать многочисленные взаимосвязи, существующие на всех уровнях программистских абстракций, а также взаимосвязи между этими уровнями. 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 маниакальной усидчивостью, сосредоточенностью и упорством для перебора всех возможных вариантов поведения своих абстракций и доскональной проработки всех деталей. Проработка должна быть абсолютно точной и не должна содержать ни одной ошибки, неправильного, лишнего или отсутствующего символа исходного кода (а это порой миллионы строк). Инструменты программирования: синтаксические анализаторы, компиляторы и прочее, - лишь незначительно помогают в этой работе.

И еще одна аналогия программных проектов с кинематографом. Наличие даже самых звездных актеров не обеспечивает успех фильма. Только талантливый режиссер способен организовать и вдохновить акте-

ров на создание шедевра, открыть новые звезды. А талантливых режиссеров, как и талантливых менеджеров программных проектов, не так уж и много, как хотелось бы на самом деле.

1.4. Контрольные вопросы

1. Что такое проект?
2. Профессионального программирование отличается от любительского?
3. Какие области знаний описаны в SWEBOK?
4. Чем SWEBOK отличается от PMBOK?
5. Какие особенности имеет управление разработкой в IT-отрасли от других?
6. В чем особенности профессии разработчика программного обеспечения от других профессий?

Лекция 2
ПОДХОДЫ К РАЗРАБОТКЕ УПРАВЛЕНИЯ ПРОГРАММНЫМИ ПРОЕКТАМИ

За десятки лет развития программной инженерии накопилось большое количество моделей разработки программного обеспечения.

2.1. «Как получится»

Данный подход полностью основан на том, что планирование практически отсутствует. То есть формально оно есть, но где-то там, в головах у технических лидеров команды. Такой подход к разработке программного обеспечения полностью зависит от навыков и опыта вашей команды. Время и бюджет, как правило не контролируется, а итоговый результат практически не предсказуем. И, если для небольших проектов такой подход еще и имеет место быть и вполне может приносить даже положительный результат, то для более-менее крупных проектов такой подход не минуемо ведет к провалу.

2.2. «Водопад» или каскадная модель

Самой старой и известной моделью построения многоуровневого процесса разработки является каскадная (или попросту водопадная) модель (Рис. 2): в ней каждый этап разработки, соответствующий стадии жизненного цикла программного обеспечения, продолжает предыдущий.

Рис. 2. – **Каскадная модель разработки программного обеспечения**

Следуя каскадной модели, разработчик переходит от одной стадии к другой строго последовательно. Сначала полностью завершается этап «определение требований», в результате чего получается список требований к программному обеспечению. После того как требования полностью определены, происходит переход к проектированию, в ходе которого создаются документы, подробно описывающие для программистов способ и план реализации указанных требований. После того, как проектирование полностью выполнено, программистами выполняется реализация полученного проекта. На следующей стадии процесса происходит интеграция отдельных компонентов, разрабатываемых различными командами программистов. После того, как реализация и интеграция завершены, производится тестирование и отладка продукта; на этой стадии устраняются все недочёты, появившиеся на предыдущих стадиях разработки. После этого программный продукт внедряется и обеспечивается его поддержка – внесение новой функциональности и устранение ошибок.

Тем самым, каскадная модель подразумевает, что переход от одной фазы разработки к другой происходит только после полного и успешного завершения предыдущей фазы, и что переходов назад либо вперёд или перекрытия фаз – не происходит.

Тем не менее, существуют модифицированные каскадные модели (включая модель Ройса), имеющие небольшие или даже значительные вариации описанного процесса.

2.3. «Гибкое управление»

Подход основан на разделении проектов на более мелкие задачи и этапы. Такое дробление позволяет командам, применяющим гибкую методологию, учитывать отзывы участников проекта, переоценивать результаты работы и применять итеративный подход на каждом этапе процесса (Рис. 3). Один из наиболее популярных подходов к применению гибкой методологии заключается в разделении проекта на короткие этапы разработки, которые называются спринтами. Это обеспечивает команде возможность быстро выполнять работу и регулярно анализировать результаты с руководством и участниками проекта при планировании спринтов и проведении ежедневных собраний. По результатам анализа команда и участники проекта могут либо продолжить двигаться в существующем направлении, либо пересмотреть планы очередных спринтов. По сравнению с традиционными подходами к управлению проектами, в гибкой методологии основными приоритетами являются оперативность, гибкость, командная работа и потребности участников проекта.

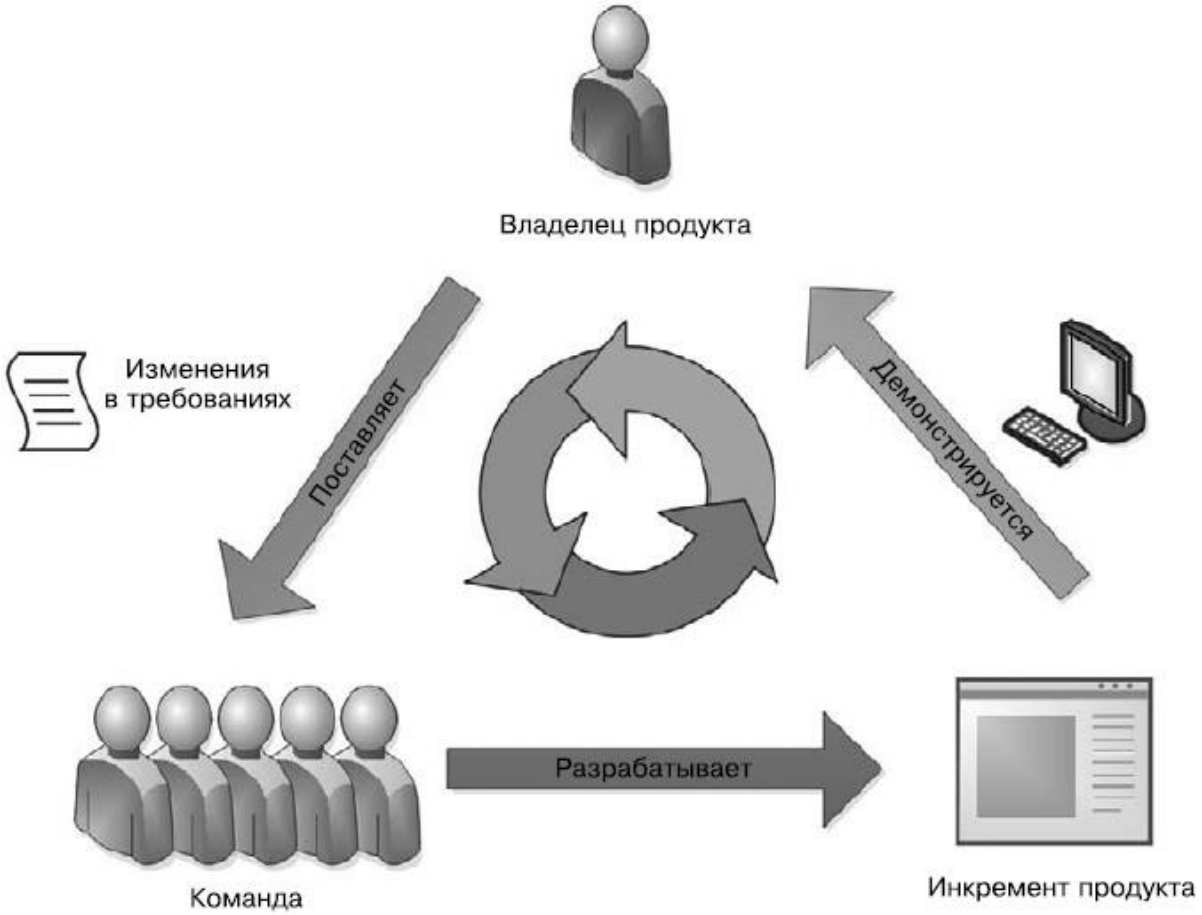

Рис. 3. – **Гибкое управление**

Идея гибкой методологии была предложена в начале 2000-х годов группой разработчиков программного обеспечения, которые сформулировали четыре основных принципа этой методики:

1. Люди и взаимодействие важнее процессов и инструментов.

2. Работающее программное обеспечение важнее полной документации.

3. Сотрудничество с клиентом важнее обсуждения условий контракта.

4. Реагирование на изменения важнее следования первоначальному плану.

Эти принципы, закрепленные в Манифесте гибкой разработки программного обеспечения (Agile Manifesto), выпущенном в 2001 году, определили методологию гибкого управления проектами и трансформировали индустрию разработки программного обеспечения.

До этого момента наиболее целесообразным подходом к управлению проектами по разработке программного обеспечения считалась каскадная модель, но к 2000-м годам она стала тяжеловесной. Принципиальным является то, что данная модель требовала огромных объемов документации и существенного планирования до начала проектных работ. После фактического начала работы выполнялись в строгом соответствии с планом отдельными и зачастую разрозненными командами, что затрудняло адаптацию проекта в случае возникновения проблем или необходимости изменений. По сравнению с этой моделью команды, применяющие гибкую методологию в разработке, могли приступать к реализации проекта быстрее, адаптироваться к возникающим проблемам и напрямую взаимодействовать с заказчиками и участниками проектов при планировании.

Классические методы управления перестают работать в случаях, когда структура и свойства управляемого объекта не известна или изменяется во времени. Эти подходы так же не эффективны, если текущие свойства объекта не позволяют ему двигаться с необходимыми характеристиками. Например, летательный аппарат не может развить требуемое ускорение или разрушается при недопустимой перегрузке. Аналогично, если рабочая группа проекта не может обеспечить требуемую эффективность и поэтому постоянно работает в режиме аврала, то это приводит не к росту производительности, а к уходу профессионалов из проекта.

Для того чтобы понять структуру и свойства объекта и воздействовать на него с целью их приведения к желаемому состоянию, в проекте должен быть дополнительный контур обратной связи – контур адаптации.

Известно, что производительность разных программистов может отличаться в десятки раз. Утверждаю, что производительность одного и того же программиста может так же отличаться в десятки раз. Заставьте лучшего в мире бегуна бегать в мешке, и он покажет в 10 раз худший результат. Заставьте лучшего программиста заниматься «сизифовым трудом»: пло-

дить документацию (которую, как правило, никто не читает) в угоду «Методологии», – и его производительность снизится в 10 раз.

Поэтому, помимо чисто управленческих задач руководитель, если он стремится получить наивысшую производительность рабочей группы, должен направлять постоянные усилия на изучение и изменение объекта управления: людей и их взаимодействия.

2.4. Контрольные вопросы

1. Какие подходы к разработке программного обеспечения существуют?
2. При каких условиях отсутствие планирования принесет лучший результат в отличии от подхода, имеющего планирование?
3. Почему «гибкое управление» в области IT заняла ведущую роль для управления проектами?
4. Какие еще подходы существуют, кроме тех, чтобы были перечислены в материале лекций?

Лекция 3
МОДЕЛИ ПРОЦЕССА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

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.

3.1. ГОСТ

Как и для любой деятельности, которая регламентируется какими-либо стандартами, так и для описания процессов разработки программного обеспечения в Республике Беларусь существуют государственные стандарты:

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

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

Данные стандарты описывают последовательный подход к разработке программного обеспечения. Разработка в соответствии с этими стандартами проводится по этапам, каждый из которых предполагает выполнение

строго определенных работ, и завершается выпуском достаточно большого числа весьма формализованных и обширных документов. Таким образом, 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.

3.2. SW-CMM

В ноябре 1986 года американский институт Software Engineering Institute (SEI) совместно с Mitre Corporation начали разработку обзора зрелости процессов разработки программного обеспечения, который был предназначен для помощи в улучшении их внутренних процессов.

Разработка такого обзора была вызвана запросом американского федерального правительства на предоставление метода оценки субподрядчиков для разработки программного обеспечения. Реальная же проблема состояла в неспособности управлять большими проектами. Во многих компаниях проекты выполнялись со значительным опозданием и с превышением запланированного бюджета. Необходимо было найти решение данной проблемы.

В сентябре 1987 года SEI выпустил краткий обзор процессов разработки программного обеспечения с описанием их уровней зрелости, а также опросник, предназначавшийся для выявления областей в компании, в которых были необходимы улучшения. Однако, большинство компаний рассматривало данный опросник в качестве готовой модели, в следствие чего через 4 года вопросник был преобразован в реальную модель, Capability Maturity Model for Software (CMM). Первая версия CMM, вышедшая в 1991 году, в 1992 году 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.

Постоянное улучшение процессов базируется на постепенном взращивании культуры компании, а не на проведении революционных инноваций. 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 18 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. Во всех проектах используется стандартный для организации процесс разработки и поддержки программного обеспечения, адаптированный под конкретный проект.

4. *Управляемый* – собираются детальные количественные данные по функционированию процессов разработки и качеству конечного продукта. Анализируется значение и динамика этих данных.

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

Документация с полным описанием CMM занимает около 500 страниц и определяет набор из 312 требований, которым должна соответствовать организация, если она планирует аттестоваться по этому стандарту на 5-ый уровень зрелости.

3.3. 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. Конечная итерация предполагает, что набор целей итерации должен в точности совпадать с набором целей, указанных заказчиком продукта, то есть все требования должны быть выполнены.

Рис. 4. – **Цикл разработки программного обеспечения в RUP**

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. Здесь начинается анализ прецедентов и построение пользовательских историй – формальное отображение требований пользователя в UML. Результатом являются документы уровня менеджмента.

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. Все изменения привязываются в RUP непосредственно к моделям, а средства автоматизации и довольно гибкий язык моделирования позволяют управлять данным процессом более или менее безболезненно в плане затрат времени и ресурсов. Здесь имеется в виду тот факт, что результатом разработки является не модель, а исполняемый код, поэтому заказчик обычно не очень любит платить за моделирование, так как модели не являются продуктом, который ему нужен.

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. Стоит специально отметить, что регрессионное тестирование в данном случае должно содержать все актуальные тесты от предыдущей итерации и приемосдаточные тесты от предыдущей фазы;

6. Внедрение – предполагает установку продукта на полигоне заказчика, подготовку персонала, запуск системы плюс приемо-сдаточные испытания, подготовка стандартов упаковки и распространения продукта, 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. Технические риски оцениваются и «расставляются» согласно приоритетам на ранних стадиях цикла разработки, а затем пересматриваются с течением времени и с развитием проекта в течение последующих итераций. Новые цели появляются в зависимости от приоритетов данных рисков. Релизы версий распределяются таким образом, что наиболее приоритетные риски устраняются первыми.

3.4. MSF

В 1994 году, стремясь достичь максимальной отдачи от IT-проектов, компания Microsoft выпустила в свет пакет руководств по эффективному проектированию, разработке, внедрению и сопровождению решений, построенных на основе своих технологий. 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) [7].

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. Модель процессов в MSF 3.0 была дополнена ещё одним инновационным аспектом: она покрывает весь жизненный цикл создания решения, начиная с его отправной точки и заканчивая непосредственно внедрением. Такой подход помогает проектным группам сфокусировать своё внимание на бизнес-отдаче решения, поскольку эта отдача становится реальной лишь после завершения внедрения и начала использования продукта.

Процесс MSF ориентирован на «вехи» (milestones) – ключевые точки проекта, характеризующие достижение в его рамках какого-либо существенного (промежуточного либо конечного) результата. Этот результат может быть оценен и проанализирован, что подразумевает ответы на вопросы: «Пришла ли проектная группа к однозначному пониманию целей и рамок проекта?», «В достаточной ли степени готов план действий?», «Соответствует ли продукт утверждённой спецификации?», «Удовлетворяет ли решение нужды заказчика?» и т. д.

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.

Итеративный подход к процессу разработки требует использования гибкого способа ведения документации. «Живые» документы должны изменяться по мере эволюции проекта вместе с изменениями требований к конечному продукту. 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.

3.5. 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.
11. В сочетании с процессом персонального процесса разработки программного обеспечения, процесс командной разработки программного обеспечения обеспечивает определенную структуру операционного процесса, которая предназначена для помощи командам менеджеров и инженеров в организации проектов и производстве программного обеспечения для продуктов, которые варьируются по размеру от небольших проектов с несколькими тысячами строк кода до очень крупных проектов с более чем полумиллионом строк кода. 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, руководства и руководства. Цикл разработки программного обеспечения TSP начинается с процесса планирования, называемого запуском, под руководством тренера, который прошел специальную подготовку. Запуск предназначен для начала процесса построения команды, и в течение этого времени команды и менеджеры

устанавливают цели, определяют роли команды, оценивают риски, усилия, распределяют задачи и создают план команды. На этапе выполнения разработчики отслеживают запланированные и фактические усилия, график и регулярно (как правило, еженедельно) отказываются от проведения совещаний для создания отчетов о состоянии и пересмотра планов. Цикл разработки завершается с помощью Post Mortem для оценки производительности, пересмотра параметров планирования и сбора извлеченных уроков для улучшения процесса.

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. Роль руководителя группы отличается от роли тренера тем, что руководители группы отвечают за управление продуктами и результатами проекта, в то время как тренер отвечает за развитие индивидуальных и командных показателей.

Последовательное применение модели PSP/TSP позволяет сделать нормой в организации пятый уровень CMM.

3.6. 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. Хотя обычно за расстановку приоритетов между поставляемыми функциями отвечает руководитель проекта или владелец продукта, то, как будет выполняться работа, решает команда. Она самостоятельно выбирает, какие части работы выполнить и как разделить обязанности между участниками.

Agile не сводится к ряду собраний и конкретных приемов разработки. Agile – это группа методологий (Рисунок 5), 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 [5].

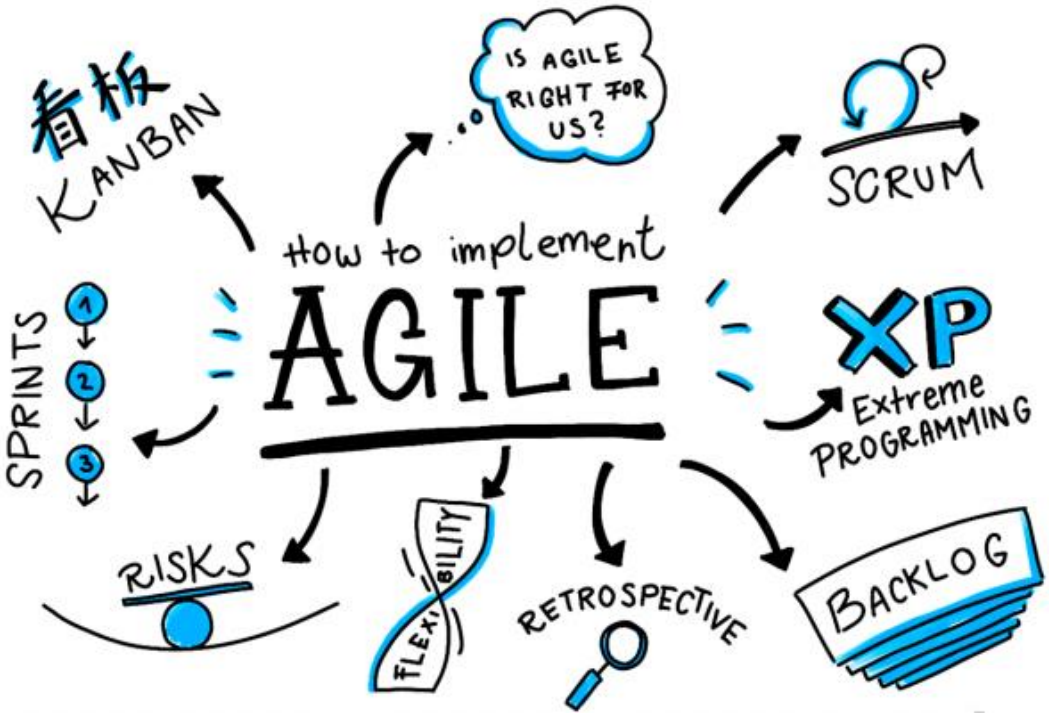

Рис. 5. – **Методология Agile**

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.

Рассмотрим, зачем нужны эти ценности Agile.

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. Правда, в современном бизнесе общение часто вынуждено переходить в онлайн.

Но тогда это должна быть видеосвязь с интерактивными онлайн-досками, а не только письма и чаты.

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.

Чтобы бизнес-ценность продукта быстро росла, заказчик с разработчиком должны плотно общаться по ходу работы. В этом случае все возникающие изменения и проблемы оперативно обрабатываются обеими сторонами.

А чтобы такое сотрудничество исполнителя и заказчика стало возможным, нужно выстраивать их доверие друг к другу.

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.

Что касается готовности к изменениям со стороны представителей заказчика (клиента), то в такой ситуации они могут пожертвовать чем-то запланированным (но менее ценным) ради новых возможностей. Готовность заказчика оперативно жертвовать какой-то частью запланированного также нужна в ситуации, когда исполнители столкнулись с непредвиденными проблемами в ходе разработки.

Помимо ценностей, в Agile-манифесте есть также **12 принципов**, которые уточняют и дополняют ценности:

1. Наивысшим приоритетом для нас является удовлетворение потребностей заказчика, благодаря регулярной и ранней поставке ценного программного обеспечения.

2. Изменение требований приветствуется, даже на поздних стадиях разработки. Agile-процессы позволяют использовать изменения для обеспечения заказчику конкурентного преимущества.

3. Работающий продукт следует выпускать как можно чаще, с периодичностью от пары недель до пары месяцев.

4. На протяжении всего проекта разработчики и представители бизнеса должны ежедневно работать вместе.

5. Над проектом должны работать мотивированные профессионалы. Чтобы работа была сделана, создайте условия, обеспечьте поддержку и полностью доверьтесь им.

6. Непосредственное общение является наиболее практичным и эффективным способом обмена информацией как с самой командой, так и внутри команды.

7. Работающий продукт – основной показатель прогресса.

8. Инвесторы, разработчики и пользователи должны иметь возможность поддерживать постоянный ритм бесконечно. Agile помогает наладить такой устойчивый процесс разработки.

9. Постоянное внимание к техническому совершенству и качеству проектирования повышает гибкость проекта.

10. Простота – искусство минимизации лишней работы – крайне необходима.

11. Самые лучшие требования, архитектурные и технические решения рождаются у самоорганизующихся команд.

12. Команда должна систематически анализировать возможные способы улучшения эффективности и соответственно корректировать стиль своей работы.

Таким образом, Agile – это не методология разработки, а 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.

3.7. Выбор модели процесса разработки

Тяжелые и легкие модели производственного процесса имеют свои достоинства и свои недостатки (Таблица 1).

Таблица 1. – **Достоинства и недостатки моделей процесса разработки**

Сложность модели	Достоинства	Недостатки
Сложная	Процессы рассчитаны на среднюю квалификацию исполнителей. Большая специализация исполнителей. Ниже требования к стабильности команды. Отсутствуют ограничения по объему и сложности выполняемых проектов.	Требуют существенной управленческой надстройки. Более длительные стадии анализа и проектирования. Более формализованные коммуникации.
Простая	Меньше непроизводительных расходов, связанных с управлением проектом, рисками, изменениями, конфигурациями. Упрощенные стадии анализа и проектирования, основной упор на разработку функциональности, совмещение ролей. Неформальные коммуникации.	Эффективность сильно зависит от индивидуальных способностей, требуют более квалифицированной, универсальной и стабильной команды. Эффективность сильно зависит от индивидуальных способностей, требуют более квалифицированной, универсальной и стабильной команды.

Те, кто пытается следовать описанным в книгах моделям, не анализируя их применимость в конкретной ситуации, показания и противопоказания, уподобляются последователям культа «Карго» – религии самолетопоклонников. В Меланезии верят, что западные товары созданы духами предков и предназначены для меланезийского народа. Считается, что белые люди нечестным путём получили контроль над этими предметами. В наиболее известных культах карго из кокосовых пальм и соломы строятся точные копии взлётно-посадочных полос, аэропортов и радиовышек. Члены культа строят их, веря в то, что эти постройки привлекут транспортные самолёты (которые считаются посланниками духов), заполненные грузом (карго). Верующие регулярно проводят строевые учения («муштру») и некое подобие военных маршей, используя ветки вместо винтовок и рисуя на теле ордена. Все это для того чтобы снова с неба спустились самолеты и этих предметов стало больше.

Алистер Коуберн, один из авторов «Манифеста гибкой разработки программного обеспечения» проанализировал очень разные программные проекты, которые выполнялись по разным моделям от совершенно облегченных и «гибких» до тяжелых (CMM-5) за последние 20 лет. Он не обна-

ружил корреляции между успехом или провалом проектов и моделями процесса разработки, которые применялись в проектах. Отсюда он сделал вывод о том, что эффективность разработки программного обеспечения не зависит от модели процесса, а также о том, что:

– у каждого проекта должна быть своя модель процесса разработки.

– у каждой модели - свое время.

Это означает, что 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, в соответствие с «Законом 4-х П» (Рис. 6). 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, например, сайта «dosug.by». И, наконец, по-разному следует организовывать процесс разработки в команде вчерашних студентов и в команде состоявшихся профессионалов.

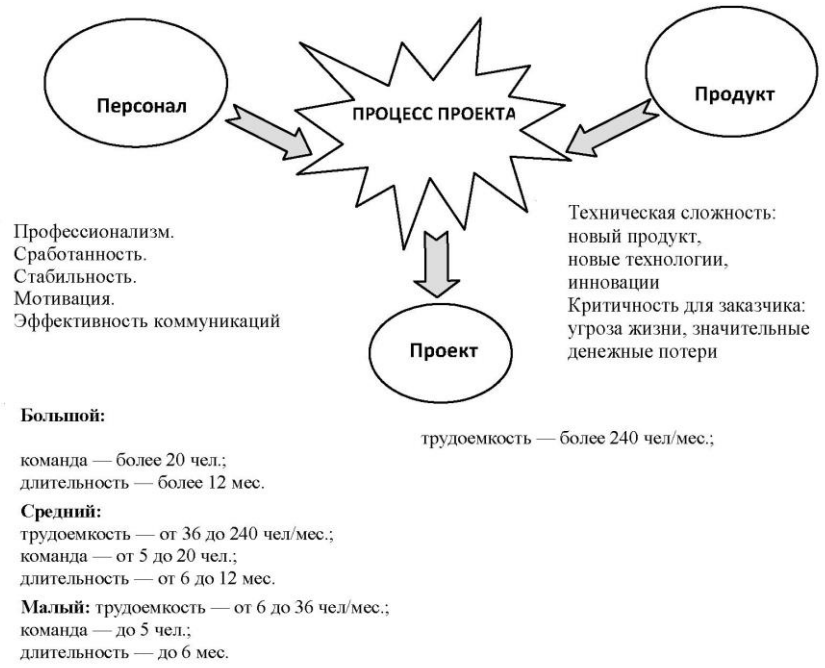

Рис. 6. – **Закон четырех П**

Команда, которая начинала проект, не остается неизменной, она проходит определенные стадии формирования и, как правило, количественно растет по мере развития проекта. Поэтому процесс должен постоянно адаптироваться к этим изменениям. Главный принцип: не люди должны строиться под выбранную модель процесса, а модель процесса должна подстраиваться под конкретную команду, чтобы обеспечить ее наивысшую эффективность.

3.8. Контрольные вопросы

1. Какие модели процесса разработки существует?
2. Где могу применяться стандарты ГОСТ для разработки программного обеспечения и нужно ли их использовать для разработки коммерческих продуктов в частных компаниях?
3. Стандарт SW-CMM описывает уровни зрелости процессов разработки программного обеспечения. Приведите примеры известных ресурсов и приложений, которые по вашему мнению соответствуют каждому уровню.
4. Какие UML-диаграммы используются для моделирования процессов управления разработкой программного обеспечения в RUP?
5. Каким образом можно совершенствовать навыки индивидуального разработчика, которыми он должен обладать согласно PCP?
6. Как реализуются профессиональные копметенции разработчика в рамках командой разработки?
7. В чем суть ценностей Agile? Какие принципы прописаны в Agile-манифесте?
8. Какие методологии стоит использовать для маленьких проектов, а какие для больших? В чем принципиальное отличие в подборе модели в таких случаях?

Лекция 4
SCRUM.
ПЛАТФОРМА ДЛЯ УПРАВЛЕНИЯ ПРОЕКТАМИ

4.1. История создания

Первоисточниками методики Scrum послужили производственная система компании Toyota и цикл OODA (OODA loop, или петля OODA, или петля Бойда) концепции применения боевой авиации, включающий в себя четыре составляющих: observe («наблюдать»), orient («ориентироваться»), decide («решать»), act («действовать»).

Подход впервые описали Хиротака Такэути и Икудзиро Нонака в статье «The New Product Development Game» (Гарвардский Деловой Обзор, январь-февраль 1986). Они отметили, что проекты, над которыми работают небольшие команды из специалистов различного профиля, обычно систематически производят лучшие результаты, и объяснили это как «подход регби». Кен Швабер в начале 1990-х использовал подход, который привел Scrum в его компанию. Впервые метод Scrum был представлен на общее обозрение задокументированным, чётко сформированным и описанным совместно Швабером и Джефом Сазерлендом на OOPSLA’96 в Остине.

К. Швабер и Д. Сазерленд на протяжении следующих лет работали вместе, чтобы обработать и описать весь свой опыт и лучшие практические образцы для индустрии в одно целое, в ту методику, что известна сегодня как SCRUM. Кен Швабер объединил усилия с Майком Бидлом в 2001 году, чтобы детально описать метод в книге «Agile Software Development with SCRUM». В 2002 году Швабер вместе с другими основал Альянс Scrum и создал серию сертифицированных аккредитаций Scrum.

Швабер покинул Scrum Alliance в конце 2009 года и основал Scrum.org, который курирует параллельную серию аккредитаций Professional Scrum. С 2009 года публичный документ под названием The Scrum Guide официально определяет Scrum. Он был пересмотрен более 5 раз. В 2018 году Швабер и сообщество Scrum.org вместе с лидерами сообщества Kanban опубликовали Руководство по Kanban для групп Scrum.

4.2. Основные понятия

Понятия Scrum и Agile часто путают, потому что Scrum строится вокруг идеи о постоянном совершенствовании, которое является главным принципом Agile. И все же Scrum – это методика работы, а Agile – это образ мышления. Перейти на Agile не так-то просто; вся команда должна стремиться изменить свой подход к созданию ценности для клиентов. Но можно просто начать использовать методику, такую как Scrum. Это направит мышление в нужное русло и поможет практиковать принципы Agile в повседневном общении и работе.

Методика Scrum по своей сути является эвристической. В ее основе лежит постоянное обучение и адаптация к изменчивым факторам. Согласно Scrum, команда не знает всего в начале проекта, но будет развиваться, извлекая уроки из опыта. В структуре Scrum заложена та свобода, с которой команды приспосабливаются к меняющимся условиям и требованиям пользователей. Рабочий процесс предусматривает изменение приоритетов и короткие циклы релиза, что способствует постоянному обучению и совершенствованию команды.

Структурированность не мешает методологии Scrum быть гибкой. Ее можно адаптировать к нуждам организации. Существует множество теорий о том, как следует применять Scrum, чтобы достичь успеха. Однако, независимо от того, какая методология в конечном итоге будет использоваться, ее главными принципами должны быть ясность коммуникации, прозрачность и стремление к постоянному совершенствованию.

Сначала определим три основных понятия Scrum: бэклог продукта, бэклог спринта и инкремент с критериями готовности. Эти три постоянные присутствуют в каждой команде Scrum.

Бэклог продукта – это главный список задач, которые необходимо выполнить. Его ведет владелец либо менеджер продукта. Это постоянно

меняющийся перечень функциональных возможностей, требований, улучшений и исправлений, из которого составляются задачи для бэклога спринта. В общем и целом, это список задач команды. Владелец продукта постоянно обращается к бэклогу продукта, меняет в нем приоритеты и поддерживает его актуальность, потому что может появиться новая информация или могут произойти изменения на рынке, из-за чего пропадет смысл выполнять имеющиеся задачи или появятся новые способы решения проблем.

Бэклог спринта – это список рабочих задач, пользовательских историй или исправлений багов, отобранных командой разработчиков для реализации в текущем цикле спринта. Перед каждым спринтом проводится собрание по планированию спринта (его мы обсудим далее в статье), на котором команда выбирает, какие задачи из бэклога продукта нужно выполнить в рамках спринта. Бэклог спринта может не быть фиксированным и может меняться по ходу спринта. Однако ничто не должно мешать достижению основной цели спринта – того, чего команда хочет добиться за текущий спринт.

Инкремент (или цель спринта) – это готовый к использованию конечный продукт по итогам спринта. В компаниях принято представлять инкремент на демонстрации в конце спринта, на которой команда показывает, что она сделала за спринт. Слово «инкремент» не так уж широко встречается в повседневной жизни. Его часто определяют, как принятые в команде критерии готовности продукта, контрольную точку, цель спринта или даже полную версию или поставленный эпик. Все зависит от того, какими критериями готовности руководствуется команда и как выбираются цели спринта. Например, некоторые команды предпочитают выпускать что-нибудь для своих клиентов в конце каждого спринта. Для них слово «готово» означает «поставлено». Однако для других команд это может быть непрактично. Представьте, что ведется разработка над серверным продуктом, который можно поставлять клиентам лишь раз в три месяца. Можно разбивать работу на двухнедельные спринты, но продукт будет «готов», когда завершится работа над частью большей версии, которую планируется поставить целиком. Однако не будем забывать, что чем больше времени уходит на выпуск программного обеспечения, тем меньше шансов у этого программного обеспечения снискать успех.

Очевидно, что допускаются различные варианты, даже когда речь идет об различных понятиях, которым можно придавать ту или иную форму. Это показывает, почему важно оставаться открытыми к совершенствованию, в частности к совершенствованию способа ведения понятий. Возможно, из-за принятых критериев готовности команда может испытывать чрезмерное давление и необходимо пересмотреть эти критерии.

4.3. Собрания

Чуть более известны такие составляющие методики Scrum, как последовательные мероприятия, церемонии или собрания, которые регулярно проводят команды Scrum. Именно в подходе к собраниям заметнее всего проявляются различия между командами. Некоторым командам в тягость проводить однообразные собрания; в других рабочие встречи обязательны.

Перечислим основные собрания, в которых может принять участие команда Scrum.

Организация бэклога. За это мероприятие, также известное как ведение бэклога, несет ответственность владелец продукта. В число его основных обязанностей входят приведение продукта в соответствие с его концепцией и постоянное отслеживание настроений на рынке и потребностей клиента. Для этого владелец продукта и ведет список, изменяя в нем приоритеты и поддерживая его в актуальном виде на основании информации от пользователей и команды разработчиков, чтобы в любое время можно было приступить к работе над внесенными в него задачами.

Планирование спринта. На этом собрании команда разработчиков под руководством Scrum-мастера планирует работу (объем спринта), которую необходимо выполнить в течение текущего спринта. На нем выбирается цель спринта. Затем в спринт добавляются конкретные пользовательские истории из бэклога продукта. Эти истории всегда соотносятся с целью. При этом команда Scrum согласовывает такие истории, которые можно будет реализовать на практике в ходе спринта. В конце собрания по планированию каждый член команды Scrum должен четко представлять, какие задачи можно выполнить за спринт и как поставить инкремент.

Спринт. Спринт – это фактический промежуток времени, в течение которого команда Scrum совместно работает над созданием готового инкремента. Как правило, спринт длится две недели, хотя некоторым командам проще спланировать объем спринта на одну неделю или поставить инкремент, обладающий достаточной ценностью, за месяц. Дейв Уэст из Scrum.org рекомендует планировать спринт тем короче, чем сложнее работа и чем больше в ней неизвестных. Но последнее слово всегда за командой. Не стесняйтесь менять продолжительность спринта, если покажется, что она не подходит. В течение этого периода владелец продукта и команда разработчиков могут пересмотреть объем спринта, если это необходимо. Это и есть ключ к пониманию эмпирической сути Scrum.

Все мероприятия, от планирования до ретроспективы, проводятся в течение спринта. После того как временной промежуток для спринта определен, он должен оставаться неизменным, пока ведется разработка. Так команда будет извлекать ценные уроки из прошлого опыта и применять выводы к будущим спринтам.

Ежедневное Scrum-совещание, или стендап. Это очень короткое ежедневное собрание, которое для удобства проводится в одно и то же время (обычно утром) и в одном и том же месте. Многие команды стараются уложиться в 15 минут, однако это лишь рекомендация. Такое собрание также называется «ежедневным стендапом», что подчеркивает его краткость. Ежедневное Scrum-совещание проводится, чтобы каждый участник команды был в курсе происходящего, не отклонялся от цели и получал план работы на ближайшие 24 часа.

Чаще всего в рамках стендапа каждому участнику команды предлагается ответить на следующие три вопроса, связанные с достижением цели спринта:

– «Что мне удалось сделать вчера?»,

– «Что я планирую сделать сегодня?»,

– «Может ли мне что-то помешать?».

Впрочем, такое собрание может превратиться в зачитывание людьми записей из ежедневника. Стендап нужен, чтобы вся болтовня оставалась в рамках ежедневного собрания, а в остальное время команда могла сосредоточиться на работе.

Обзор итогов спринта. В конце спринта команда собирается для просмотра демонстрации инкремента (или для его изучения) в неформальной обстановке. Команда разработчиков представляет заинтересованным сторонам и коллегам завершенные рабочие задачи из бэклога, чтобы собрать отзывы. Владелец продукта решает, стоит ли выпускать инкремент, хотя в большинстве случаев команда получает зеленый свет.

На собрании по обзору итогов владелец продукта также изменяет бэклог продукта на основании результатов последнего завершенного спринта. Этот процесс может перейти в планирование следующего спринта. Если спринт длится один месяц, отводите под собрание для обзора итогов не более четырех часов.

Ретроспектива спринта. Ретроспектива проводится, чтобы команда зафиксировала и обсудила все успехи и неудачи спринта, проекта, участников и их взаимоотношений, инструментов или даже определенных собраний. Цель ретроспективы – создать условия, чтобы команда могла уделить внимание всему, что удалось и что нужно улучшить в следующий раз, и не зацикливалась на неудачах.

4.4. Основные участники

Состав scrum-команды предполагает три отдельные роли: владелец продукта, scrum-мастер и команда разработчиков. Поскольку scrum-команды сочетают в себе множество функций, в команду разработчиков также входят тестировщики, дизайнеры, UX-специалисты и инженеры по операциям.

Владелец продукта Scrum. Владельцы ратуют за свой продукт. Их задача – понимать требования бизнеса, клиента и рынка. На основе этого понимания они расставляют приоритеты между рабочими задачами, которые техническая команда будет выполнять в соответствующем порядке. Об эффективных владельцах продукта можно сказать следующее:

1. Они составляют бэклог продукта и управляют им.

2. Они тесно сотрудничают с руководством компании и командой, сообщая каждому участнику значение рабочих задач в бэклоге продукта.

3. Они дают команде понятные указания, чтобы ее участники знали, какие возможности поставить следующими.

4. Они решают, когда поставить продукт, стремясь делать это как можно чаще.

Роль владельца продукта не всегда совмещена с ролью менеджера продукта. Владельцы стремятся создать все условия, чтобы команда разработчиков создавала максимальную ценность для бизнеса. Важно, чтобы владельцем продукта был один человек. Вряд ли команда разработчиков захочет получать разные указания от разных владельцев одновременно.

Scrum-мастер. Scrum-мастера следят за применением принципов Scrum в своих командах. Они обучают команды, владельцев продуктов и остальную компанию тонкостям Scrum-процесса и стараются оптимизировать применение этой практики.

Успех Scrum-мастера зависит от того, насколько хорошо он разбирается в работе, которую выполняет команда, и может помочь команде повысить прозрачность работы и оптимизировать процесс поставки продукта. Это главный координатор, который составляет перечень требуемых ресурсов (кадровых и материально-технических) для собраний по планированию спринта и обзору его итогов, стендапа и ретроспективы спринта.

Команда разработчиков Scrum. На команды Scrum ложится вся основная работа. Они специалисты по принципам сбалансированной разработки. Самые успешные команды сплочены, находятся в одном месте и обычно состоят из 5–7 участников. Чтобы определить размер команды, можно обратиться к известному «правилу двух пицц», которое сформулировал глава Amazon Джефф Безос (в команде должно быть столько участников, чтобы им хватало двух пицц).

Каждый участник команды обладает своими компетенциями. Участники обучают друг друга выполнению разных задач, чтобы ни один из них не стал препятствием на пути к цели. Успешные Scrum-команды способны к самоорганизации, и их подход к проектам пронизан командным духом. Все участники команды помогают друг другу, чтобы успешно завершить спринт.

Scrum-команды составляют план на каждый спринт. Они прогнозируют объем работы, который способны выполнить за итерацию, используя в качестве ориентира показатели своей скорости в прошлых спринтах. Бла-

годаря фиксированной продолжительности итерации команда разработчиков может проанализировать правильность оценки сложности и процесса поставки продукта, что, в свою очередь, значительно повышает точность ее прогнозов со временем.

4.5. Эффективность работы команды

Существует эффективный способ проведения оценки трудоёмкости выполнения задач спринта в некоторых единицах трудоёмкости – человеко-часах и тому подобных. Оценка задач выполняется разработчиками группы проекта вместе со scrum-мастером и владельцем продукта. Правильным методом оценки задач является покер планирования. Показано, что такая оценка трудоёмкости значительно точнее оценок, проводимых другими лицами.

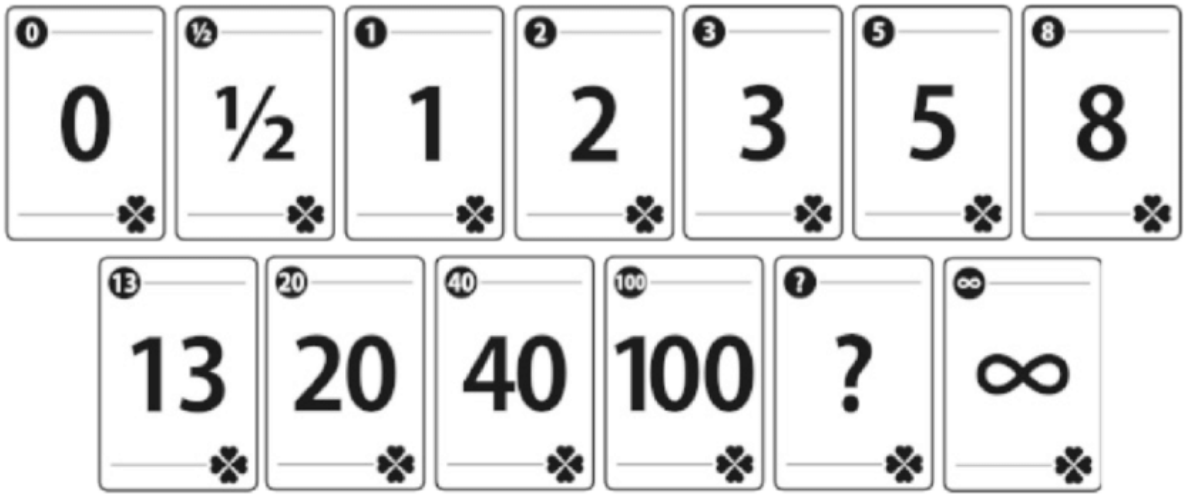

Рис. 7. – **Карточки для покера планирования**

Каждый разработчик должен дать свою независимую от других оценку трудоёмкости задачи, при этом должны использоваться числа из ряда Фибоначчи (1, 2, 3, 5, 8, 13, 20, 40, 100). Вместо чисел 21, 34, 55 используются числа 20, 40, 100. Оценки могут записываться на листках бумаги, либо для этого могут использоваться специальные карточки (Рисунок 7) и должны открываться одновременно. Такая организация проведения оценки позволяет избежать эффекта привязки.

Эффект привязки возникает, когда команда открыто обсуждает оценки. Команда обычно имеет в своём составе как сдержанных, так и импульсивных участников, могут быть участники, у которых есть определённые планы; разработчики, вероятно, захотят как можно больше времени заниматься работой над проектом, а владелец продукта или заказчик, вероятно, захочет, чтобы работа была закончена как можно скорее.

Если все выбранные разработчиками значения попадают в интервал не более чем из трёх последовательных чисел Фибоначчи, то в качестве конечной оценки задачи группой используется усреднённая оценка всех разработчиков группы. Если выбранные оценки выходят за пределы такого интервала, то разработчики с наибольшим и наименьшим значением должны объяснить основания своего выбора, после чего раунды оценки повторяются до тех пор, пока множество оценок не уложится в интервал из трёх последовательных чисел Фибоначчи. Как показывает практика, в случае, если для формирования конечной оценки трудоёмкости задачи применяется вариант с усреднением оценок лежащих в интервале большем, чем три последовательных числа Фибоначчи, то результат оказывается значительно менее точным.

Хотя изначально задачи, и, соответственно, истории и проект в целом, оцениваются в какой-то определённой единице трудоёмкости, эти оценки в дальнейшем используются как относительные величины трудоёмкости в качестве очков для определения скорости работы команды Scrum.

Производительность вычисляется в конце спринта как сумма очков (story points) по всем полностью завершенным элементам бэклога спринта. Очки по частично завершенным или незавершенным историям не должны участвовать в расчете производительности команды.

Прогноз производительности должен отслеживаться в течение спринта на основании диаграммы сгорания задач спринта (Рисунок 8). Конечно, в идеале итоговая производительность спринта должна совпасть с числом очков по всем задачам, запланированным на спринт, но по факту она может отличаться как в меньшую, так и в большую сторону.

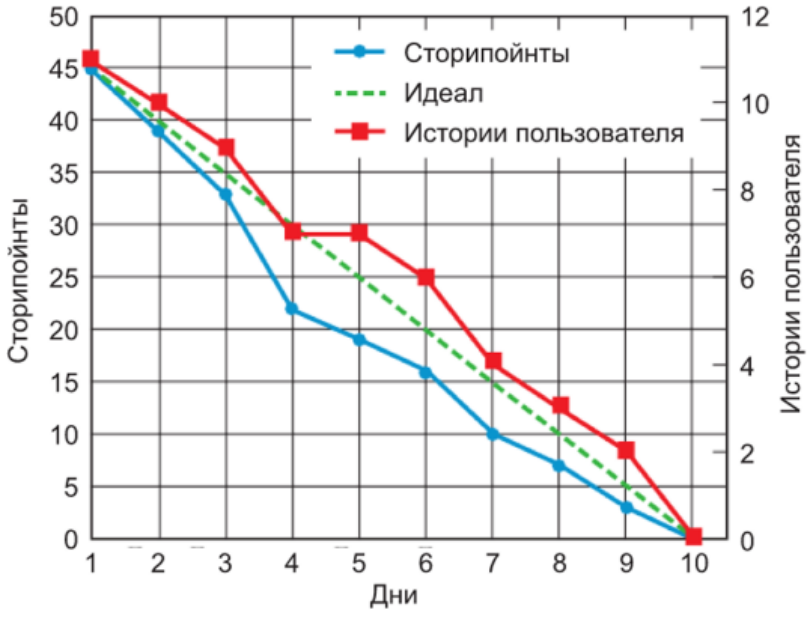

Рис. 8. – **Диаграмма сгорания задач**

Производительность – это ключевой механизм получения обратной связи для команды. Она позволяет оценить, как внедрённые процессные изменения повлияли на эффективность ее работы. И хотя производительность команды от спринта к спринту может меняться, в среднем у хорошо функционирующих scrum-команд она стабильно возрастает примерно на 10% за спринт.

Этот показатель помогает достаточно точно прогнозировать, сколько историй команда может делать за один спринт. Для расчета прогноза необходимо взять среднее значение производительности за последние три спринта. Это означает, что для корректного расчета производительности команде необходимо работать в том же составе, как минимум, три спринта, что бывает очень сложно объяснить нетерпеливым заказчикам.

4.6. Контрольные вопросы

1. Что такое Scrum? Чем Scrum отличается от Agile?
2. Что такое спринт? Из каких этапов он состоит?
3. Каким образом составляется бэклог проекта? Как образом составляется бэклог спринта?
4. Что такое пользовательские истории? Что должно быть в них описано?
5. Что может стать причиной остановки спринта? Когда это стоит делать?
6. Как оценивается эффективность работы каждого сотрудника?
7. Какие выходы существуют из ситуаций, когда команда не справляется с поставленными задачами?
8. Как распределяются задачи на спринте?
9. Что такое экстремальное программирование? Как оно реализуется в Scrum?
10. Какие еще методики кроме Scrum основаны на подходе «гибкого управления»? Чем они отличаются от Scrum?
11. Можно ли подход Kanban использовать в рамках спринта?

Лекция 5
ИНИЦИАЦИЯ ПРОЕКТА

5.1. Основные определения

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: выполняются людьми, ограничены доступностью ресурсов, планируются, 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.

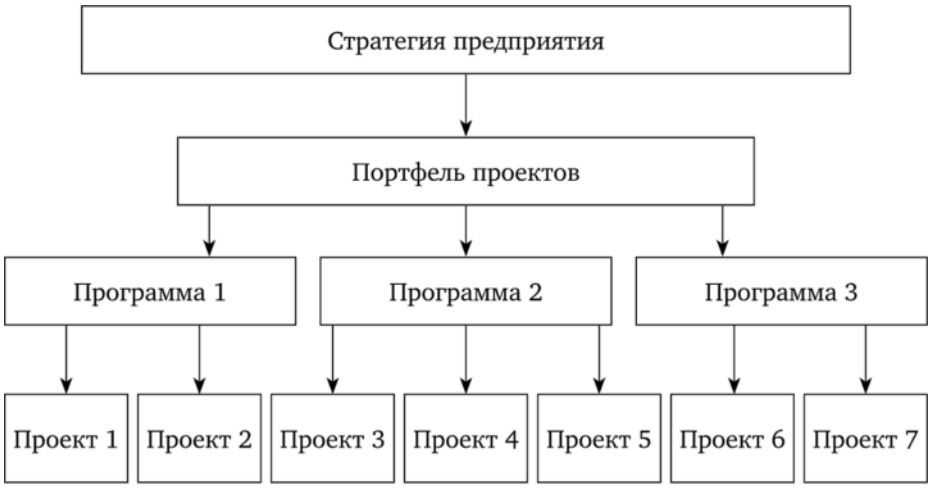

Рис. 9. – **Структура проектной деятельности**

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.

5.2. Критерии успешности проекта

Задача проекта – достижение конкретной бизнес-цели, при соблюдении ограничений «железного треугольника» (Рисунок 10). 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 соответствие 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 гибельного для проекта, чем равнодушие или уныние его участников.

Рис. 10. «**Железный треугольник» проекта**

Эффективность – это отношение полученного результата к произведенным затратам. Нельзя рассматривать эффективность, исходя только из результативности: чем больше ты производишь, чем больше делаешь, тем выше твоя эффективность. С таким подходом можно «зарезать на ужин курицу, несущую золотые яйца». Затраты не следует путать с инвестициями. Оплата аренды, электроэнергии, коммунальные платежи – затраты. Создание и закрепление эффективной команды – это стратегическое приобретение компании. Обучение участников проекта – инвестиции. Вложение в людей – это увеличение числителя в формуле эффективности. Уход из компании всех профессионалов после проекта, выполненного по принципу «любой ценой», – затраты, причем очень тяжело восполняемые. Нарастающая конкуренция указывает на совершенно четкий тренд в мировой экономике – персонал – это форма инвестиций, активов, которые нужно уметь наращивать, управлять и сохранять. Сегодня люди – это капитал.

Современное предприятие обязано относиться к своим работникам так же, как к своим лучшим клиентам. Главный капитал современной компании – это знания. Большая часть этих знаний неотъемлема от их носителя – человека. Те предприятия, которые этого не поняли, не выживут потому, что не смогут быть эффективными. Сегодня эффективное предприятие – это сервис. Предприятие, с одной стороны, предоставляет услуги и продукты своим клиентам, а с другой, – рабочие места для профессионального персонала. Принципы «Одно предприятие на всю жизнь», «Рабо-

тай продуктивно, а предприятие о тебе позаботится» – уходят в прошлое. На рынке рабочей силы в IT – правила устанавливают профессионалы.

5.3. Приоритеты проекта

Приоритет любого проекта должен определяться на основе оценки трех его характеристик:

1. Финансовая ценность;
2. Стратегическая ценность;
3. Уровень рисков.

Шкала **оценки финансовой ценности проекта** может выглядеть следующим образом:

– *Высокая.* Ожидаемая окупаемость до 1 года. Ожидаемые доходы от проекта не менее чем в 1.5 раз превышают расходы. Все допущения при проведении этих оценок четко обоснованы.

– *Выше среднего.* Ожидаемая окупаемость проекта от 1 года до 3 лет. Ожидаемые доходы от проекта не менее чем в 1.3 раза превышают расходы. Большинство допущений при проведении этих оценок имеют под собой определенные основания.

– *Средняя.* Проект позволяет улучшить эффективность производства в Компании и потенциально может снизить расходы компании не менее чем на 30%. Проект может иметь информационную ценность или помочь лучше контролировать бизнес.

– *Низкая.* Проект немного снижает расходы компании не менее чем на 10% и дает некоторые улучшения производительности производства.

Одной финансовой ценности для определения приоритета проекта недостаточно. Например, ни одна компания разработчик программного обеспечения не возьмется за автоматизацию нелегального оборота наркотиков, если это не соответствует стратегии ее бизнеса. Поэтому, важным показателем приоритета проекта является его соответствие стратегическим целям компании.

Шкала **оценки стратегической ценности проекта** может иметь следующий вид:

– *Высокая.* Обеспечивает стратегическое преимущество, дает устойчивое увеличение рынка или позволяет выйти на новый рынок. Решает значительные проблемы, общие для большинства важных клиентов. Повторение конкурентами затруднено или потребует от 1 до 2 лет.

– *Выше среднего.* Создает временные конкурентные преимущества. Выполнение обязательств перед многими важными клиентами. Конкурентное преимущество может быть удержано в течение 1 года.

– *Средняя.* Поддерживается доверие рынка к компании. Повышает мнение клиентов о качестве предоставляемых услуг или способствует вы-

полнению обязательств перед несколькими клиентами. Конкуренты уже имеют или способны повторить новые возможности в пределах года.

– *Низкая.* Стратегическое воздействие отсутствует или незначительно. Влияние на клиентов несущественно. Конкуренты могут легко повторить результаты проекта.

Третьим обязательным показателем приоритета проекта должна быть оценка уровня его риска. Ни один проект, который имеет даже самую высокую оценку финансовой выгодности, не будет запущен в производство, если достижение этой сверх выгоды имеет минимальные шансы.

Примерная шкала **оценки уровня рисков проекта** может иметь следующий вид:

– *Низкий.* Цели проекта и требования хорошо поняты и документированы. Масштаб и рамки проекта заданы четко. Ресурсы требуемой квалификации доступны в полном объеме. Разрабатываемые системы не потребуют новой технологической платформы.

– *Средний.* Цели проекта определены более-менее четко. Хорошее понимание требований к системе. Масштаб и рамки проекта заданы достаточно хорошо. Ресурсы требуемой квалификации доступны в основном. Системы создаются на новой, но стабильной технологической платформе.

– *Выше среднего.* Цели проекта недостаточно четки. Задачи системы или бизнес-приложения поняты недостаточно полно. Понимание масштаба и рамок проекта недостаточно. Ресурсы требуемой квалификации сильно ограничены. Системы создаются на новой технологической платформе, сомнения в рыночной стабильности платформы.

– *Высокий.* Цели проекта нечетки. Основные функциональные компоненты системы не определены. Масштаб и рамки проекта непонятны. Ресурсы требуемой квалификации практически отсутствуют. Системы создаются на новой технологической платформе, в отношении которой крайне мало ясности. Технологии имеют неподтвержденную стабильность.

Если компания уделяет мало внимания управлению приоритетами своих проектов, то это приводит к переизбытку реализуемых проектов, перегруженности исполнителей, постоянным авралам и сверхурочным работам и, как следствие, к низкой эффективности производственной деятельности. При старте нового проекта с высоким приоритетом, компания должна остановить или закрыть менее значимые проекты, чтобы обеспечить новый проект необходимыми ресурсами, а не пытаться сделать все и сразу за счет интенсификации работ, как правило, это не получается.

5.4. Концепция проекта

Концепция проекта разрабатывается на основе анализа потребностей бизнеса. Главная функция документа – подтверждение и согласование еди-

ного 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.

Концепцию проекта следует излагать по содержанию в соответствии с требуемыми разделами документа.

Цели и результаты проекта. 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 значимыми (направленными на достижение стратегических целей компании), конкретными (специфичными для данного проекта), измеримыми (иметь проверяемые количественные оценки), реальными (достижимыми). 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.

Допущения и ограничения. 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.

Ключевые участники и заинтересованные стороны. На этапе инициации проекта необходимо выявить и описать всех его участников: заинтересованные стороны, лица и организации, например, заказчики, спонсоры, исполняющая организация, которые активно участвуют в проекте или чьи интересы могут быть затронуты при исполнении или завершении проекта. Участники могут влиять на проект и его результаты поставки. К ключевым участникам программного проекта, относятся:

1. Спонсор проекта – лицо или группа лиц, предоставляющая финансовые ресурсы для проекта в любом виде.

2. Заказчик проекта – лицо или организация, которые будут использовать продукт, услугу или результат проекта. Следует учитывать, что заказчик и спонсор проекта не всегда совпадают.

3. Пользователи результатов проекта.

4. Куратор проекта – представитель исполнителя, уполномоченный принимать решение о выделении ресурсов и изменениях в проекте.

5. Руководитель проекта – представитель исполнителя, 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 требуется определить и оценить ресурсы, необходимые для его выполнения:

1. Людские ресурсы и требования к квалификации персонала.

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

3. Бюджет проекта. План расходов и, при необходимости, предполагаемых доходов проекта с разбивкой по статьям и фазам/этапам проекта.

4. Специфика программного проекта заключается в том, что людские ресурсы вносят основной вклад в его стоимость. Все остальные затраты, как правило, незначительны, по сравнению с этими расходами. На фазе инициации проекта хорошей считается оценка трудозатрат с точностью от –50% до +100%.

Помимо непосредственно программирования в проекте, разработки программной системы (ПС), есть много других процессов, которые требуют ресурсов соответствующей квалификации, а само программирование составляет лишь четверть всех затрат.

Прежде чем определять численность и состав проектной команды, необходимо сделать оценку трудоемкости разработки программных средств [чел/час].

Сроки проекта. Практически ни один проект невозможно завершить быстрее, чем за 3/4 расчетного оптимального графика вне зависимости от количества занятых в нем специалистов. Кроме сроков завершения проекта необходимо еще определить его этапы – контрольные точки (вехи), в которых будет происходить переоценка проекта на основе реально достигнутых показателей.

Контрольная точка – важный момент или событие в расписании проекта, отмечающее достижение заданного результата и/или начало/завершение определенного объема работы. Каждая контрольная точка характеризуется датой и объективными критериями ее достижения.

Для программного проекта контрольные точки должны соответствовать выпуску каждой промежуточной версии ПС, в которой будет реализована и протестирована определенная часть конечной функциональности программного продукта. В зависимости от сложности и масштаба проекта продолжительность одной итерации может составлять от 2 до 4 недель.

Риски проекта. Риск – неопределенное событие или условие, наступление которого отрицательно или положительно сказывается на целях проекта. Как правило, в случае возникновения негативного риска почти всегда стоимость проекта увеличивается и происходит задержка в выполнении мероприятий, предусмотренных расписанием проекта.

На этапе инициации, когда нет необходимых данных для проведения детального анализа риска, часто приходится ограничиваться качественной оценкой общего уровня рисков: низкий, средний, высокий.

Критерии приемки проекта. Критерии приемки должны определять числовые значения характеристик системы, которые должны быть продемонстрированы по результатам приемо-сдаточных испытаний или опытной эксплуатации и однозначно свидетельствовать о достижении целей проекта.

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

1. Для кого предназначены результаты проекта. Описание текущей ситуации. Какие у потенциального заказчика существуют проблемы.

2. Каким образом результаты проекта решают эти проблемы.

3. Насколько значимо для клиента решение данных проблем (оценка экономического эффекта).

4. Какие преимущества в итоге из этого может извлечь компания-исполнитель проекта.

5.5. Контрольные вопросы

1. Какие стандарты существуют для управления проектами?

2. В чем суть «железного треугольника» проекта?

3. Как реализуются приоритеты проекта в рамках небольшого проекта?

4. Из каких разделов состоит концепция проекта? Как правильно выбрать название проекта? Как оценить риски проекта?

5. Каковы критерии успешности проекта? Что необходимо для этого делать?

Лекция 6
ПЛАНИРОВАНИЕ ПРОЕКТА

6.1. Содержание проекта

«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 боле простые подзадачи, которые, в свою очередь, могут быть разделены на еще боле 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 у частников 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.

6.2. Планирование управления содержанием проекта

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.

6.3. Планирование организационной структуры проекта

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.

6.4. Планирование управления конфигурациями проекта

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 восстановление 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.

6.5. Планирование качеством проекта

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.

6.6. Планирование расписания проекта

После определения трудоемкости работ необходимо определить график их выполнения и общие сроки реализации проекта – составить расписание работ по проекту. Базовое расписание - утвержденный план-график с указанными временными фазами проекта, контрольными точками и элементами иерархической структуры работ.

Базовое расписание может быть наиболее наглядно представлено диаграммой Ганта (Рис. 11). В этой диаграмме плановые операции или элементы иерархической структуры работ перечислены с левой стороны,

даты отображаются сверху, а длительность операций показана горизонтальными полосками от даты начала до даты завершения.

Базовое расписание это, как правило, элемент контракта с заказчиком. Контрольные точки (вехи) должны служить точками анализа состояния проекта и принятия решения «GO/NOT GO», поэтому они должны зримо демонстрировать статус проекта. Контрольная точка «Проектирование завершено» - плохо. Наиболее эффективный подход – метод последовательных поставок: контрольная точка «Завершено тестирование требований 1, 3, 5, 7»

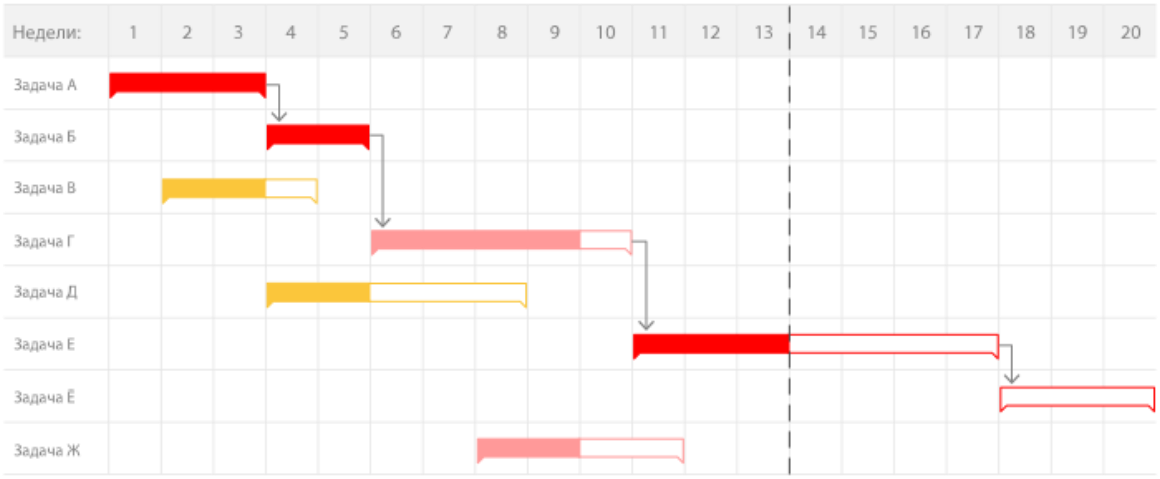

Рис. 11. – **Диаграмма Ганта**

Если работы не связаны между собой, то любую из них мы можем начинать и завершать, когда нам удобно. Все работы можно делать параллельно и в этом случае минимальная длительность проекта равна длительности самой долгой работы. Однако, на практике между работами существуют зависимости, которые могут быть «жесткими», например, анализ - проектирование – кодирование – тестирование и документирование конкретной функции; или «нежесткими», которые могут пересматриваться или смягчаться. Например, последовательное выполнение задач конкретным исполнителем (можно перепланировать на другого исполнителя) или разработка базового программного обеспечения, которая должна предшествовать разработке прикладного программного обеспечения. В этом случае можно создавать «заглушки» эмулирующие работу базового программного обеспечения. Таким образом, диаграмма Ганта для расписания проекта выглядит как гамак, составленный из множества цепочек взаимосвязанных работ с единой точкой начала и завершения.

Критический путь проекта – самая длинная цепочка работ в проекте. Увеличение длительности любой работы в этой цепочки приводит к увеличению длительности всего проекта.

В проекте всегда существует хотя бы один критический путь, но их может быть несколько. Критический путь может меняться во время исполнения проекта. При исполнении проекта руководитель должен обращать внимание на исполнение задач на критическом пути в первую очередь и следить за появлением других критических путей. Практическая рекомендация: на критическом пути должны стоять работы с нежесткими связями, которые всегда можно перепланировать, если возникает угроза срыва сроков.

6.7. Контрольные вопросы

1. Из каких этапов состоит управление проектами?
2. Приведите примеры «ползучего фичеризма», которые основаны на планировании вашего рабочего процесса в течении дня?
3. Что входит в состав организационной структуры проекта?
4. Что подразумевается под качеством программного продукта? Как оценивать качество управления процессом разработки?
5. Что такое диаграмма Гранта? Как она используется при оценки критического пути в рамках проекта?

Лекция 7
УПРАВЛЕНИЕ КОМАНДОЙ

7.1. Организация команды

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 формировании 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. Предпочтительный стиль лидерства на этом этапе – директивный.

На данном этапе лидер должен:

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

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 данном этапе лидер должен:

1. Помогать членам команды установить нормы общения
2. Способствовать эффективному взаимодействию участников команды.
3. Участвовать в обсуждении решений команды.
4. Поощрение участников команды, с той целью, чтобы они делились идеями.
5. Предотвращать конфликты.
6. Внушать уверенность и спокойствие.

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 (не полностью ориентированный на задачи). Лидер обязан максимально сконцентрироваться на поддержке и оценке усилий каждого участника, совершенствовании рабочей атмосферы, коммуникациях, сотрудничестве и дальнейшем улучшении отношений в группе.

На данном этапе лидер должен:

1. Открыто выражать свое мнение о ситуациях и вещах, которые беспокоят сотрудников.
2. Ставить задачи, решаемые коллективно.
3. Делегировать свои полномочия участникам команды.

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.

На данном этапе лидер должен:

1. Совместно с членами команды определять цели, которые были бы интересны всем сотрудникам.

2. Способствовать повышение эффективность команды.

3. Развивать навыки команды по оцениванию свой деятельности.

4. Признавать личный вклад каждого члена команды.

5. Развивать каждого отдельного участника команды.

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.

7.2. Состав команды

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. Ниже приведена таблица, в которой описаны группы специалистов, которые участвуют в разработке проекта (табл. 2).

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 [5].

Независимо от того, в какой триаде работает команда, всегда можно ускорить ее работу и позволить получать больше удовольствия от процесса.

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. Иногда, правда, коллективных способностей команды для реализации проекта оказывается недостаточно.

Таблица 2. – **Структура команд проекта по группам**

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. Тем не менее команде стоит заручиться поддержкой специалиста узкого профиля в следующих случаях:

– сотрудник с определенными компетенциями не нужен команде на постоянной основе;

– в компании работает ограниченное количество сотрудников с определенными компетенциями, которые взаимодействуют с несколькими командами;

– для работы в конкретной сфере, к которой у обычной команды нет доступа, нужно специальное разрешение.

7.3. Принципы работы руководителя

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 достижение 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.

7.4. Контрольные вопросы

1. Каким образом происходит процесс формирования команды?
2. Сколько времени необходимо, чтобы сформировать сплоченную команду?
3. Какими компетенциями должен обладать руководитель?
4. Как вести себя лидеру команды, если в один из ее членов явно отстает от других?
5. Каким образом стоит распределять задачи между участниками команды?

Лекция 8
УПРАВЛЕНИЕ РИСКАМИ ПРОЕКТА

8.1. Основные понятия

Как бы не организованно и правильно разрабатывался проект, но 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 обусловливает 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 предотвращение и снижению последствий на проект.

2. Непрогнозируемые. Риски, которые невозможно идентифицировать и, следовательно, спланировать ответные действия.

Непрогнозируемые риски – это непредвиденные событий. 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 прави-

ло, 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

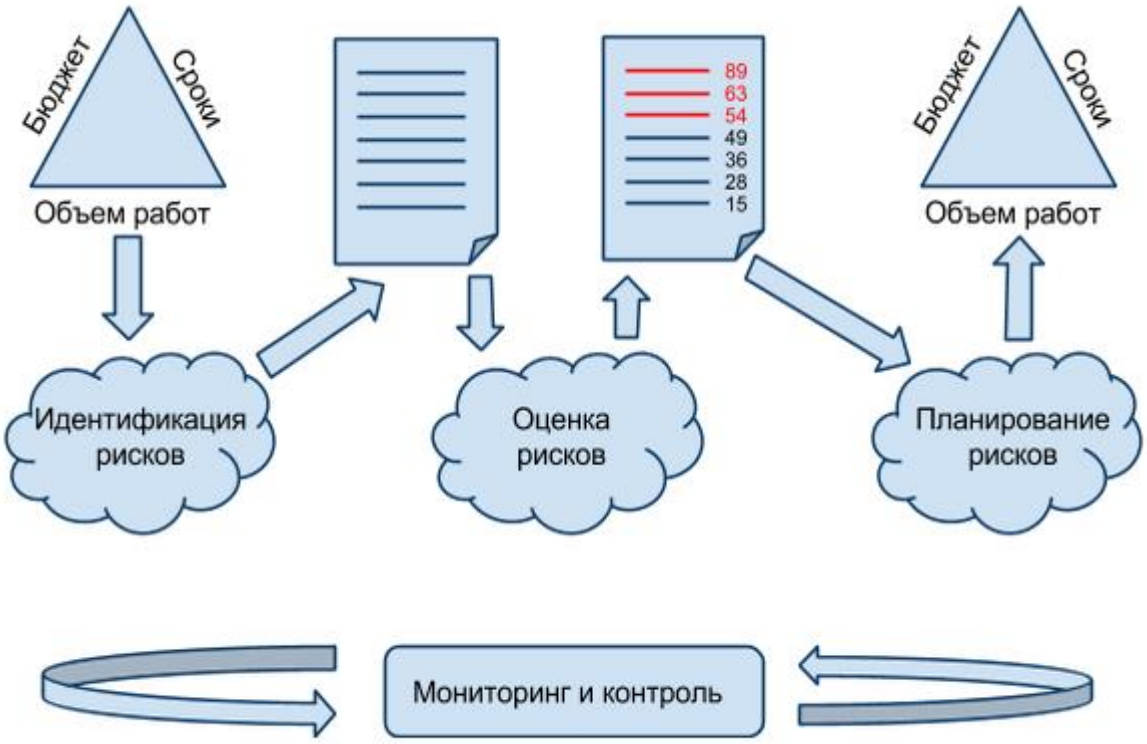

Рис. 13. – **Жизненный цикл рисков проекта**

Рассмотри более подробно каждый этап.

8.2. Идентификация рисков

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).

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 определенной 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 дается определение. Цель – составить первичный перечень возможных рисков для последующего отбора и анализа.

3. Метод Дельфи. Данный подход похож на мозговой штурм, за исключение 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. Остается составить список названных рисков и раздать его участникам для внесения изменений и дополнений.

Наиболее распространенными рисками проекта по разработке программного обеспечения являются:

1. Дефицит квалифицированных специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация функциональности, несоответствующая проекту.
4. Разработка плохого пользовательского интерфейса.

5. Перфекционизм, ненужная оптимизация.
6. Постоянный поток правок в проекте.
7. Нехватка информации.
8. Недостатки в работах, выполняемых внешними ресурсами.
9. Слабая производительность итогового проекта.
10. Слишком большая разница в квалификации различных участников проекта.

8.3. Анализ рисков

Очевидно, что бороться со всеми рисками дорого и неэффективно, поэтому цель анализа рисков – выявить наиболее важные из них.

Анализ рисков можно разделить на 2 этапа: качественный анализ и количественный анализ.

1. Качественный анализ включает в себя:
2. Определение вероятности реализации рисков.
3. Определение тяжести последствий реализации рисков.
4. Определения ранга риска по матрице «вероятность – последствия».
5. Определение близости наступления риска.
6. Оценка качества использованной информации.

Для оценки рисков необходима точная и адекватная информация. Использование неточной информации ведет к ошибкам в оценке. Неверная оценка риска также является риском.

Критерии оценки качества используемой при анализе информации выглядят следующим образом:

1. Степень понимания риска.
2. Доступность и полнота информации о риске.
3. Надежность, целостность и достоверность источников данных.

Результаты качественного анализа используются в ходе последующего количественного анализа рисков и планирования реагирования на риски.

Для количественного анализа рисков могут быть использованы следующие методы:

1. Анализ чувствительности.
2. Анализ дерева решений.
3. Моделирование и имитация.

Анализ чувствительности помогает определить, какие риски обладают наибольшим потенциальным влиянием на проект. В процессе анализа устанавливается, в какой степени неопределенность каждого элемента проекта отражается на исследуемой цели проекта, если остальные неопределенные элементы принимают базовые значения. Результаты представляются, как правило, в виде диаграммы «торнадо» (Рис. 14).

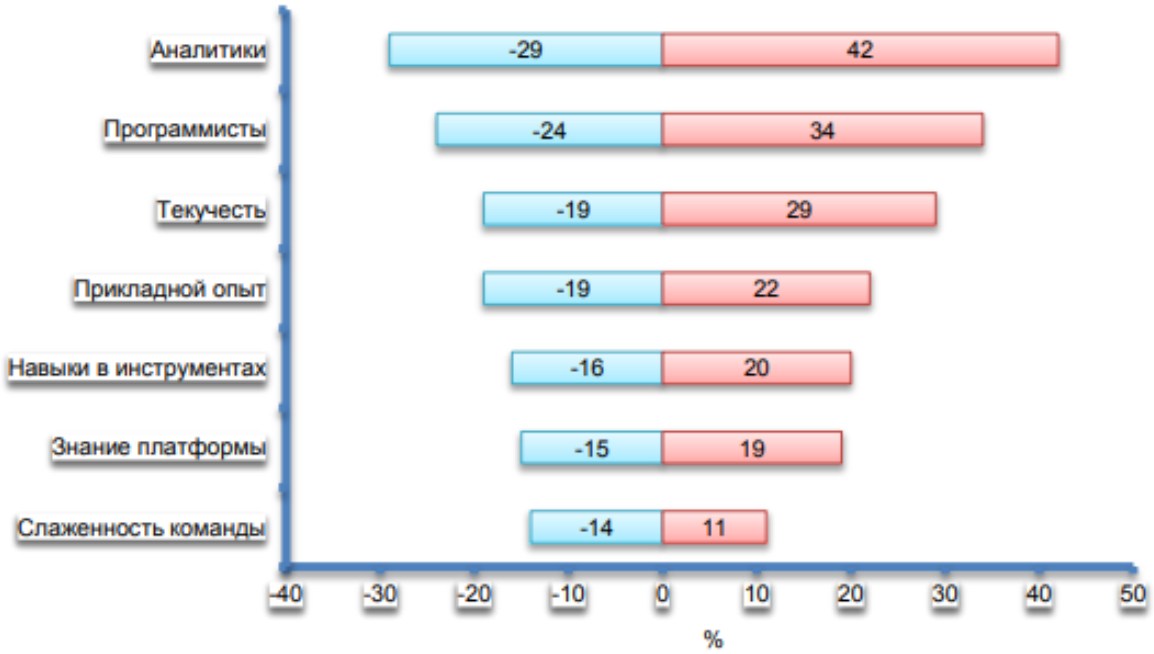

Рис. 14. – **Влияние факторов риска на трудозатраты**

8.4. Планирование рисков

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.

8.5. Мониторинг и контроль рисков

Мониторинг и контроль рисков – это процесс идентификации, анализа и планирования реагирования на новые риски, отслеживания ранее идентифицированных рисков, а также проверки и исполнения операций реагирования на риски и оценка эффективности этих операций.

В процессе мониторинга и контроля рисков используются различные методики, например, анализ трендов и отклонений, для выполнения которых необходимы количественные данные об исполнении, собранные в процессе выполнения проекта.

Мониторинг и контроль рисков включает в себя следующие задачи:

1. Пересмотр рисков.
2. Аудит рисков.
3. Анализ отклонений и трендов.

Пересмотр рисков должен проводиться регулярно, согласно расписанию. Управление рисками проекта должно быть одним из пунктов повестки дня всех совещаний команды проекта. Идентификация новых рисков, и пересмотр известных рисков происходит с использованием процессов, описанных ранее.

Аудит рисков предполагает изучение и предоставление в документальном виде результатов оценки эффективности мероприятий по реагированию на риски, относящихся к идентифицированным рискам, изучение основных причин их возникновения, а также оценку эффективности процесса управления рисками.

Тренды в процессе выполнения проекта подлежат проверке с использованием данных о выполнении. Для мониторинга выполнения всего проекта могут использоваться анализ освоенного объема и другие методы анализа отклонений проекта и трендов. На основании выходов этих анализов можно прогнозировать потенциальные отклонения проекта на момент его завершения по показателям стоимости и расписания. Отклонения от базового плана могут указывать на последствия, вызванные как угрозами, так и благоприятными возможностями.

8.5. Контрольные вопросы

1. Что такое риски? Каким образом их можно определить?
2. Как определяется жизненный цикл рисков проекта? Из каких этапов он состоит?
3. Какие методы существуют для определения рисков?
4. Какие риски наиболее распространены среди разработчиков программного обеспечения?
5. Какие существует методы количественного анализа рисков?
6. Каким образом можно предотвратить повторное появление рисков?

ЛИТЕРАТУРА

1. Guide to the Software Engineering Body of Knowledge, SWEBOK v 3.0. – IEEE Computer Society Professional Practices Committee, 2014. – 335 p.

2. Руководство к своду знаний по управлению проектами (Руководство PMBOK®) + Agile: практическое руководство (6-е издание). – Newtown Square, PA: Project Management Institute, 2017. – 1170 с.

3. Архипенков С. Лекции по управлению программными проектами. – М.: Самиздат, 2009. – 128 с.

4. Архипенков С. Руководство командой разработчиков программного обеспечения. Прикладные мысли. – М.: Самиздат, 2008. – 79 с.

5. Что такое Agile? [Электронный ресурс]: Прагматичное руководство Atlassian по agile-разработке. URL: https://www.atlassian.com/ru/agile (дата обращения: 20.08.2021).

6. Обзор Agile. Что это: методология, метод или философия? [Электронный ресурс]: Аджайл и Скрам. URL: https://scrumtrek.ru/blog/agile-scrum/ (дата обращения: 20.08.2021).

7. Microsoft Solutions Framework Дисциплина управления рисками MSF вер. 3.1. – White paper, 2002. – 41 с.

8. Брукс, Ф. Мифический человеко-месяц, или Как создаются программные комплексы / Ф. Брукс; пер. с англ. – СПб.: Символ-Плюс, 1999. – 304 с.

Учебное издание

УПРАВЛЕНИЕ IT-ПРОЕКТАМИ

Курс лекций

Составитель
НИКИТИН Александр Игоревич

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Л.И. Ячменёва*

Подписано в печать 2021. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 3,83. Уч.-изд. л. 3,94. Тираж экз. Заказ .

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.