С.Архипенков

Лекции по управлению программными проектами

2009

Москва

С. Архипенков

Лекции по управлению программными проектами

Москва
2009

Содержание

Отзыв на книгу

Когда я прочитал эту книжку, я расстроился. Дело в том, что существует огромное количество публикаций по управлению проектами, но все они носят либо формальный характер (просто описание стандарта PM BOK), либо слишком эклектичны и описывают только отдельные приемы управления. Зная это, я собирался сам написать практически полезную книгу по этому предмету. Теперь не буду, С. Архипенков меня опередил.

В книжке С. Архипенкова в достаточно интересной форме описаны основные элементы управления программными проектами. Автор, конечно, скромничает, говоря, что он охватил только 20% объема материала, я думаю, что гораздо больше. С массой интересных примеров описывается понятие программной инженерии, её сходства и отличия по сравнению с другими областями управления, далее в увлекательной форме описываются понятия проекта, жизненного цикла, планирования, управления рисками, вопросы формирования коллектива и реального управления. Некоторые свежие примеры (аналогия с киноиндустрией, культ «Карго» и многие другие) я обязательно буду использовать в своих лекциях. Каждая глава завершается выводами, где в краткой форме формулируются основные идеи раздела управления проектами. Мои студенты будут в восторге от этой книги, прежде всего благодаря выводам. Нынешнее поколение студентов очень экономно и не любит читать толстых книг.

Особенно мне понравилось внятное изложение принципов количественного управления и соответствующих экономических оценок. Каждый раз, когда я читаю лекцию на эту тему, я сталкиваюсь с непониманием студентов, как это можно сравнивать числа разной размерности – плановый объем PV и освоенный объем EV, где EV выражается в рублях, но на самом деле является некоторым эквивалентом реально выполненных работ. С. Архипенкову удалось это объяснить буквально на пальцах так, что даже я понял.

По въевшейся профессорской привычке не удержусь от нескольких замечаний. Прежде всего, меня резануло слишком частое упоминание программирования как ремесла, лишенного строгой научной базы. Разумеется, это не так. Сегодняшнее программирование как небо от земли отличается от того предмета, с которого я начинал 40 лет назад. Есть еще очень много открытых тем для исследований, но тем и характеризуется каждая живая наука.

На странице 44 в сноске есть язвительное замечание, что «в 2000 году ещё оставались те, кто верил, что из модели UML можно будет генерировать исходный код». Мы уже много лет создаем информационные системы, программное обеспечение для телефонных станций и других систем реального времени практически без единой строки кода, написанного вручную. Да, действительно, пришлось многие вещи додумать, защитить несколько диссертаций, но практический опыт промышленного применения показывает, что цитируемая идея вовсе не была эфемерной.

Если бы эту книгу писал я, то, конечно, не уделил бы столько внимания несколько шарлатанской идее определения объема будущего ПО на основе функциональных точек. Этот метод, и то с натяжкой, работает только для типового ПО. С. Архипенков всего второй человек среди моих знакомых, который верит в этот метод. Первым был профессор Александр Самочадин из Санкт-Петербургского Политеха, да и тот недавно в частной беседе признался, что разочаровался в нем.

Как обычно, нашлось около десятка опечаток, но ни эти мелочи, ни предыдущие, более серьезные, на мой взгляд, замечания не изменили моего мнения о книжке, которую я «проглотил» за два приема. Это действительно хорошая книга, написанная живым, приятным для чтения языком, отражающая основные аспекты столь важной науки, как управление проектами.

Я с удовольствием буду рекомендовать книгу С. Архипенкова своим студентам и аспирантам, а если удастся, то и сотрудникам нашего холдинга AT Software.

Зав.кафедрой системного программирования СПбГУ

Председатель Совета директоров AT Software

Доктор физ-мат наук, профессор

А.Н. Терехов

Об авторе

Эксперт в управлении проектами, PMP PMI. В разработке ПО более 30 лет. Создавал имитационные модели сложных космических систем в Центре управления полетами (ЦНИИМаш). Руководил коммерческой разработкой ПО и проектами организационного развития в компаниях PriceWaterhouseCoopers, Luxoft, CBOSS. Выполнял проекты по заказу Европейского космического агентства, «Даймлер-Бенц Аэроспейс», корпорации «Боинг», ЦБ РФ, ОАО «Газпром». Автор книг, статей и учебных курсов по информационным технологиям и управлению программными проектами.

Вопросы и отзывы о лекциях можно отправить автору по адресу: sergey@arkhipenkov.ru.

Благодарности

Учителя это не те, кто учит, а те, у кого учатся. Мне повезло, для меня такими учителями стали Н.А.Анфимов, И.К.Бажинов, А.Т.Горяченков, Ю.А.Мозжорин, Н.Н.Нетылев, А.И.Шеховцов, а также многие другие научные работники и специалисты ЦНИИМаш. Еще больше было учителей, у которых я учился тому, как не надо руководить людьми. Всем спасибо!

Предисловие

Об управлении проектами за последние десять лет написано тысячи книг и прочитаны миллионы лекций. Какая может быть польза еще от одного курса лекций?

История этого курса берет начало в те времена, когда я руководил разработкой программного обеспечения (ПО) в Ассоциации CBOSS. Одной из задач, которую пришлось решать, было внедрение проектного управления в разработку. Когда в рамках этого внедрения первые руководители проектов прошли внешнее обучение, они высказали сомнение в эффективности такого подхода, поскольку «детальное планирование операции «Буря в пустыне» имеет очень мало общего с насущными задачами менеджера программного проекта».

С этим трудно было спорить, поскольку разработка ПО имеет свою специфику, которая плохо укладывается в классическое управление проектами. Поэтому было решено подготовить собственный курс по управлению проектами именно в разработке ПО. Так появилась первая версия данного «курса молодого бойца», который предназначался для начинающих руководителей проектов и их начальников. Не ставилась задача, заменить свод знаний PMBOK от PMI. Знать и понимать лучшие практики, изложенные в этом руководстве, должен каждый профессиональный менеджер проекта. В лекции вошли лишь 20% знаний о дисциплине управления проектами, но эти знания будут востребованы в 80% ситуаций, с которыми участникам проекта придется столкнуться в реальной работе.

С тех пор этот курс был прочитан десятки раз в ходе корпоративных и открытых тренингов для руководителей программных проектов и лидеров команд разработчиков из ведущих российских компаний производителей ПО. Структура и содержание курса постоянно улучшались, благодаря обратной связи, получаемой от слушателей, в ходе активных дискуссий и обсуждений с коллегами выполненных ими проектов. Наконец, появилось ощущение определенной стабилизации текущей версии. Поэтому был объявлен режим «features freeze» и принято решение курс документировать.

Ну, и кому от этого счастье?

В первую очередь, хочется надеяться, что лекции будут полезны студентам, которые собираются профессионально заниматься разработкой ПО. В вузах изучают языки и системы программирования, теории алгоритмов и реляционных баз данных, где-то даже ООП, паттерны проектирования и много что еще, но я не встречал курсов, на которых бы *учили работать*. Профессиональная разработка ПО имеет мало общего со «спихиванием лабораторок». Приходя в промышленную разработку программного обеспечения, молодой специалист долго методом проб и ошибок 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. Хотелось бы восполнить данный пробел.

Вторая категория читателей, которым адресована данная книга, - начинающие руководители программных проектов и лидеры команд разработчиков, которые вышли из программистов. Люди - не программные компоненты, поэтому хороший программист не всегда становится хорошим руководителем. Надеюсь, что данный курс поможет им не только анализировать риски, планировать и контролировать проектные работы, но и, что самое главное, научит понимать людей, эффективно взаимодействовать с ними, разрешать конфликты и обеспечивать адекватную мотивацию продуктивной работы.

И, наконец, третья категория читателей - это руководители, которые ранее успешно управляли проектами в других отраслях и получили назначение руководить проектами разработки ПО. Буду удовлетворен, если данный курс 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.

Лекция 1. Введение в программную инженерию

История и основные понятия

Программная инженерия есть применение определенного систематического измеримого подхода при разработке, эксплуатации и поддержке программного обеспечения [1].

Термин software (программное обеспечение, ПО) ввел в 1958 году всемирно известный статистик Джон Тьюкей (John Tukey). Термин software engineering (программная инженерия) впервые появился в названии конференции НАТО, состоявшейся в Германии в 1968 году и посвященной так называемому кризису программного обеспечения. С 1990-го 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 [2]. В 2004 году в отрасли был создан основополагающий труд «Руководство к своду знаний по программной инженерии» (SWEBOK) [3], в котором были собраны основные теоретические и практические знания, накопленные в этой отрасли.

Во избежание двусмысленностей, но, не претендуя на академичность, позволю себе ввести рабочие определения ряда терминов, которые я буду в дальнейшем активно использовать.

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

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

Это отображение может быть очень простым, например, перфорирование машинных команд и данных на перфокартах. А может быть многоступенчатым и очень сложным, 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 ПО окружения (ОС, промежуточное ПО, 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 [2] 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 ПО, выявленных при его эксплуатации (Рисунок 1). Жизненный цикл программного продукта завершается выводом продукта из эксплуатации и снятием его с поддержки и сопровождения.

Рисунок 1 Жизненный цикл программного продукта

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

Самый распространенный процесс разработки ПО, который пришлось наблюдать за годы работы в отрасли, можно назвать «как получится». Это не означает, что процесса как такового нет. Он есть и, как правило, обеспечивает разработку ПО при приемлемых затратах и качестве, но этот процесс не

документирован, является «знанием стаи», держится на людях и передается из поколения в поколение. Целенаправленная работа по оценке эффективности и улучшению процесса не ведется.

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

Согласно SWEBOK 2004, программная инженерия включает в себя 10 основных и 7 дополнительных областей знаний, на которых базируются процессы разработки ПО. К основным областям знаний относятся следующие области:

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.

Дополнительные области знаний включают в себя:

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.

Все это необходимо знать и уметь применять, для того чтобы разрабатывать ПО. Как видим, управление проектами, о котором мы будем говорить 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.

Отличия программной инженерии от других отраслей

Standish Group, проанализировав работу сотен американских корпораций и итоги выполнения нескольких десятков тысяч проектов, связанных с разработкой ПО, в своем докладе с красноречивым названием «Хаос» [4] пришла к следующим неутешительным выводам (Рисунок 2):

- Только 35 % проектов завершились в срок, не превысили запланированный бюджет и реализовали все требуемые функции и возможности.

- 46 % проектов завершились с опозданием, расходы превысили запланированный бюджет, требуемые функции не были реализованы в полном объеме. Среднее превышение сроков составило 120%, среднее превышение затрат 100%, обычно исключалось значительное число функций.

- 19 % проектов полностью провалились и были аннулированы до завершения.

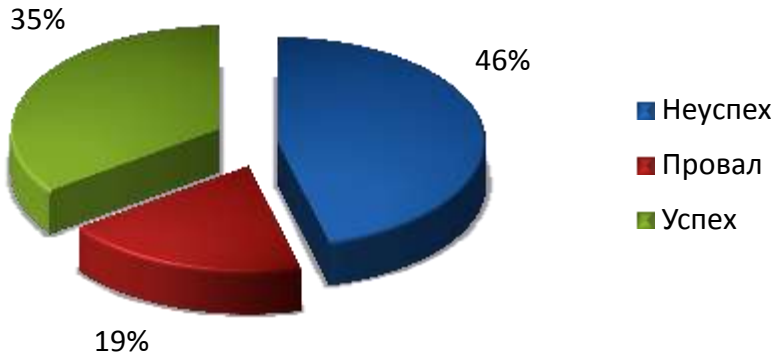

Рисунок 2. Результаты анализа успешность программных проектов за 2006 год

Сразу дам ответы на вечные российские вопросы «Кто виноват?» и «Что делать?».

Кто виноват? Никто. Как никто не виноват в том, что на небе тучи, что идет дождь, 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 мы обязаны считаться, если, конечно, не хотим «дуть против ветра».

Что делать? Управлять людьми. Успех, а равно и провал, проектов по производству ПО лежат в области психологии.

Гуру программирования Ф. Брукс в 1975 году писал [5]: «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».

То, что производят программисты нематериально – это коллективные мысли и идеи, выраженные на языке программирования. Я не говорю, что производство ПО суперсложная интеллектуальная деятельность. Отрасль еще только зарождается. Время вхождения в профессию сильно меньше, чем в других инженерных дисциплинах. Разрабатывать ПО точно не сложнее, чем делать ракеты. Просто в силу уникальности отрасли опыт профессионалов, накопленный в материальном производстве и изложенный в стандарте PMI PMBOK [6], мало способствует успеху в управлении программным проектом. Управлять разработкой ПО надо иначе.

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.

Творчество неразрывно связано с вдохновением, а это субстанция капризная и непредсказуемая (помните знаменитый сон Д. И. Менделеева, про Периодическую таблицу элементов его имени?). Знаю только, что без вдохновения в программировании не обойтись. И чем сложнее задача, тем труднее извлечь это вдохновение из подсознания. Иногда для этого требуются часы, а иногда недели.

Программирование это 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.

Существующее состояние программной инженерии напоминает большую поваренную книгу с многочисленными описаниями рецептов однажды успешно приготовленных блюд из ингредиентов, которых в будущем уже не будет. Завтра в новой системе будут другие вычислительные машины, технологии,

языки программирования, инструменты и окружающее ПО, новые проблемы взаимодействия с которыми обязательно придется решать.

Профессиональное творчество программиста принципиально отличается от творчества в науке и искусстве. Программистские задачи с каждым годом становятся все сложнее и объемнее, а сроки, за которые требуется решить эти задачи, наоборот, с каждым годом сокращаются. Поэтому современные программы создаются коллективами от нескольких до тысяч программистов, в то время как творческие деятели науки и искусства работают, как правило, в одиночку.

Есть еще нечто, что отличает труд профессионального программиста от ученого, художника, композитора и поэта. Предметом деятельности ученых являются упрощенные модели, в которых они могут абстрагироваться от большинства деталей реального мира, не существенных для их целей. Математик, доказывая новую теорему о тензорах, не заботится ни о чем, кроме системы постулатов, положенных в основание дифференциальной геометрии. Физик, описывая динамику жидкости в трубе, абстрагируется от того, как движутся и сталкиваются молекулы и от того, как движутся планеты вокруг Солнца. Деятели искусства тоже во многом оперируют абстракциями. Поэту, композитору, художнику достаточно лишь сделать намек, абрис объекта творчества, и на этом его работа закончена. Остальное пусть додумывает читатель, слушатель, зритель.

Программист тоже работает с абстракциями, но ему приходится держать в голове гораздо больше абстракций, чем любому ученому. Абстракции сопутствуют программисту на всех уровнях разработки программы от описания ее целей до исполняемого машинного кода. И этих уровней могут быть десятки. 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 проч., - лишь незначительно помогают в этой работе.

Еще одна особенность, которая присуща программистскому творчеству, это постоянное обновление информационных технологий, которые программисту необходимо знать и успешно применять в своей работе. Поэтому профессиональный программист должен, как сказал один из наших прежних вождей, «учиться, учиться и учиться». Программист должен удерживать в голове, постоянно пополнять и активно применять на практике гигабайты профессиональной информации. Это устройство компьютеров, компьютерных сетей и сетевые протоколы. Это операционные системы и языки программирования. Это программные интерфейсы промежуточного ПО и прикладных библиотек с особенностями и багами их реализации в конкретных продуктах. Это технологические стандарты, технологии разработки и

инструменты, которые их поддерживают. Это архитектуры программных систем, паттерны и антипаттерны проектирования и много-много другой информации.

Еще в начале 70-х замечательный ученый академик А.П.Ершов сказал [8]: «Программист должен обладать способностью первоклассного математика к абстракции и логическому мышлению в сочетании с эдисоновским талантом сооружать все, что угодно, из нуля и единиц. Он должен сочетать аккуратность бухгалтера с проницательностью разведчика, фантазию автора детективных романов с трезвой практичностью экономиста». Образно можно сказать, что программист должен сочетать в себе легкость и полет таланта Моцарта с усидчивостью и скрупулезностью Сальери.

Программирование - не искусство и не наука – это ремесло. Сегодня мы так же далеки от индустриальной разработки программ, как и 50 лет назад.

А поскольку это ремесло, то человек, научившийся писать программы на С ++, будет так же далек от профессионала, как ученик третьего класса средней школы, научившийся писать по-русски, от А. С. Пушкина или Ф. М. Достоевского. Путь к мастерству в ремесле лежит только через опыт. Нельзя научиться программированию, читая книги. Как нельзя по книгам научиться писать романы, картины, стихи, музыку. А еще программистам нужен постоянный труд самоусовершенствования и саморазвития. Поэтому далеко не все, кто пишет программы, становятся профессионалами.

Почему-то, если мы говорим о поэтах, художниках, композиторах, то разброс творческой производительности никого не удивляет. «Творческий полет», «творческий застой» - это про деятелей искусства. А когда говорим о неравномерности производительности программистов, то многие менеджеры начинают с этим спорить, и пытаются «пинать» программистов, как будто это заставит их думать быстрей. Не заставит. Но может заставить уволиться или заняться имитацией работы.

Правда, существуют еще инженерные дисциплины такие, как строительство, машиностроение, авиастроение и другие отрасли материального производства, в которых над созданием новых изделий трудятся сотни тысяч человек. Очень велик соблазн провести аналогию с этими отраслями и говорить об индустриальном подходе к разработке ПО. Не получается.

Упрощенно, путь от идеи до ее реализации в этих отраслях выглядит следующим образом: НИР-ОКР-завод. В верхней части этой пирамиды находятся отраслевые НИИ, которые производят идеи и занимаются проектированием новых изделий. На втором этаже пирамиды работают конструкторы в конструкторских бюро, в задачу которых входит реализация нового проекта в чертежах деталей и технологиях изготовления и сборки. На нижнем уровне находятся производственные мощности - заводы, на которых инженеры и рабочие воплощают «в железе» чертежи и технологии.

Если проводить аналогию, то программисты работают исключительно на вершине описанной пирамиды. Программирование – это проектирование и только проектирование. Роль конструкторского бюро для программного проекта выполняют компилятор и сборщик программ. А программистским аналогом завода, который переводит конструкторскую документацию в продукт, доступный потребителю, служит вычислительный комплекс, на котором развертывается и выполняется созданная программа.

А теперь давайте вспомним, сколько НИР так и остались на бумаге, не дойдя до ОКР, и сколько еще ОКР закончилось закрытием тематики. Я думаю, что процент инноваций, дошедших до производства от общего числа проектов,

выполненных в отраслевых НИИ, будет сравним с процентом успешных программных проектов. И давайте еще учтем, что ученые в НИИ опираются на достаточно хорошо изученные законы математики, физики и химии, а программирование, как мы отмечали выше, пока остается лишь ремесленным производством.

Для коллективного программистского творчества скорее уместна аналогия с созданием художественного кинофильма или театрального спектакля. Количество провальных проектов в этих областях ничуть не меньше, чем в программировании. Дай Бог, если хотя бы пятая часть кинофильмов не «ложится на полки» после первого показа. Об этом же пишет авторитет в управлении программными проектами У.Ройс [7]: «Менеджеры программных проектов смогут добиться большего, если будут применять методы управления, характерные для киноиндустрии».

И еще одна аналогия программных проектов с кинематографом. Наличие даже самых звездных актеров не обеспечивает успех фильма. Только талантливый режиссер способен организовать и вдохновить актеров на создание шедевра, открыть новые звезды. А талантливых режиссеров, как, впрочем, и талантливых менеджеров программных проектов, к сожалению, не так много, как хотелось бы.

Эволюция подходов к управлению программными проектами

За 50 лет развития программной инженерии накопилось большое количество моделей разработки ПО. Интересно провести аналогию между историей развития методов, применяемых в системах автоматического управления летательными аппаратами, и эволюцией подходов к управлению программными проектами.

«Как получится». Разомкнутая система управления. Полное доверие техническим лидерам. Представители бизнеса практически не участвует в проекте. Планирование, если оно и есть, то неформальное и словесное. Время и бюджет, как правило, не контролируются. Аналогия: баллистический полет без обратной связи. Можно, но недалеко и неточно.

«Водопад» или каскадная модель. Жесткое управление с обратной связью. Расчет опорной траектории (план проекта), измерение отклонений, коррекция и возврат на опорную траекторию. Лучше, но не эффективно.

«Гибкое управление». Расчет опорной траектории, измерение отклонений, расчет новой попадающей траектории и коррекция для выхода на нее. «Планы - ничто, планирование - все» (Эйзенхауэр, Дуайт Дэвид)

«Метод частых поставок». Самонаведение. Расчет опорной траектории, измерение отклонений, уточнение цели, расчет новой попадающей траектории и коррекция для выхода на нее.

Классические методы управления перестают работать в случаях, когда структура и свойства управляемого объекта нам не известны и/или изменяются со временем. Эти подходы так же не помогут, если текущие свойства объекта не позволяют ему двигаться с требуемыми 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.

Когда структура и свойства управляемого объекта нам не известны, необходимо использовать *адаптивное управление*, которое, дополнительно к прямым управляющим воздействиям, направлено на изучение и изменение свойств управляемого объекта. Продолжая аналогию с управлением летательными аппаратами - это расчет опорной траектории, измерение отклонений, уточнение цели, уточнение объекта управления, адаптация (необходимое изменение) объекта управления, расчет новой попадающей траектории и коррекция для выхода на нее.

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»: плодить документацию (которую, как правило, никто не читает) в угоду «Методологии» (именно с большой буквы ‘М’), - 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.

Модели процесса разработки ПО

Модели (или, как еще любят говорить, методологии) процессов разработки ПО принято классифицировать по «весу» - количеству формализованных процессов (большинство процессов или только основные) и детальности их регламентации. Чем больше процессов документировано, чем более детально они описаны, тем больше «вес» модели.

Наиболее распространенные современные модели процесса разработки ПО представлены на Рисунок 1.

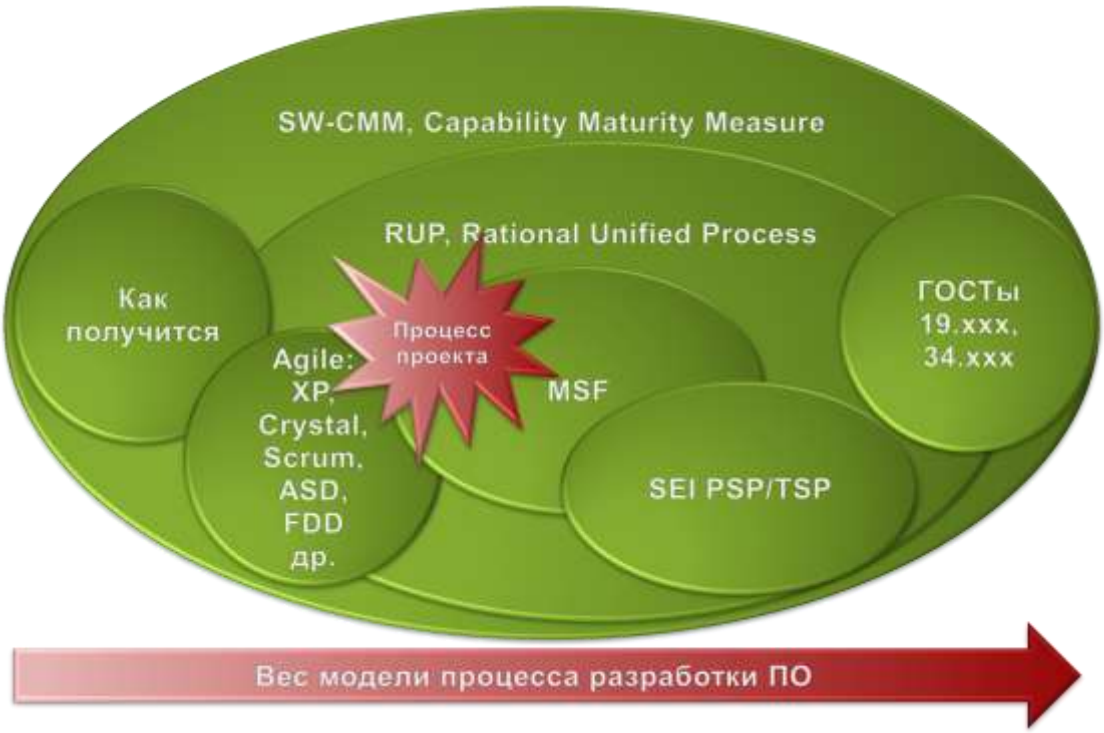

Рисунок 3 Различные модели процесса разработки ПО и их распределение по «весу»

ГОСТы

ГОСТ 19 «Единая система программной документации» и ГОСТ 34 «Стандарты на разработку и сопровождение автоматизированных систем» ориентированы на последовательный подход к разработке ПО. 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 формализованности разработки. На основе этих стандартов разрабатываются программные системы по госзаказам в России.

SW-CMM

В середине 80-х годов минувшего столетия Министерство обороны США крепко задумалось о том, как выбирать разработчиков ПО при реализации крупномасштабных программных проектов. По заказу военных Институт программной инженерии, входящий в состав Университета Карнеги-Меллона, разработал SW-CMM, Capability Maturity Model for Software [9] в качестве эталонной модели организации разработки программного обеспечения.

Данная модель определяет пять уровней зрелости процесса разработки ПО.

1. Начальный — процесс разработки носит хаотический характер. Определены лишь немногие из процессов, и успех проектов зависит от конкретных исполнителей.

2. Повторяемый — установлены основные процессы управления проектами: отслеживание затрат, сроков и функциональности. Упорядочены некоторые

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

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

RUP

Унифицированный процесс (Rational Unified Process, RUP) [10] был разработан Филиппом Крачтеном (Philippe Kruchten), Иваром Якобсоном (Ivar Jacobson) и другими сотрудниками компании "Rational Software" в качестве дополнения к языку моделирования UML.. Модель RUP описывает абстрактный общий процесс, на основе которого организация или проектная команда должна создать конкретный специализированный процесс, ориентированный на ее потребности. Именно эта черта RUP вызывает основную критику - поскольку он может быть чем угодно, его нельзя считать ничем определенным. В результате такого общего построения RUP можно использовать и как основу для самого что ни на есть традиционного водопадного стиля разработки, так и в качестве гибкого процесса.

MSF

Microsoft Solutions Framework (MSF) [11] - это гибкая и достаточно легковесная модель, построенная на основе итеративной разработки. Привлекательной особенностью MSF является большое внимание к созданию эффективной и небюрократизированной проектной команды. Для достижения этой цели MSF предлагает достаточно нестандартные подходы к организационной структуре, распределению ответственности и принципам взаимодействия внутри команды.

PSP/TSP

Одна из последних разработок Института программной инженерии Personal Software Process / Team Software Process [12,13]. Personal Software Process определяет требования к компетенциям разработчика. Согласно этой модели каждый программист должен уметь:

- учитывать время, затраченное на работу над проектом;
- учитывать найденные дефекты;
- классифицировать типы дефектов;

- оценивать размер задачи;
- осуществлять систематический подход к описанию результатов тестирования;
- планировать программные задачи;
- распределять их по времени и составлять график работы.
- выполнять индивидуальную проверку проекта и архитектуры;
- осуществлять индивидуальную проверку кода;
- выполнять регрессионное тестирование.

Team Software Process делает ставку на самоуправляемые команды численностью 3–20 разработчиков. Команды должны:

- установить собственные цели;
- составить свой процесс и планы;
- отслеживать работу;
- поддерживать мотивацию и максимальную производительность.

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

Agile

Основная идея всех гибких моделей заключается в том, что применяемый в разработке ПО процесс должен быть адаптивным. Они декларируют своей высшей ценностью ориентированность на людей и их взаимодействие, а не на процессы и средства. По сути, так называемые, гибкие методологии это не методологии, а набор практик, которые могут позволить (а могут и нет) добиваться эффективной разработки ПО, основываясь на итеративности, инкрементальности, самоуправляемости команды и адаптивности процесса.

Выбор модели процесса

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

Таблица 1. Плюсы и минусы тяжелых и легких моделей процессов разработки ПО

Вес модели	Плюсы	Минусы
Тяжелые	Процессы рассчитаны на среднюю квалификацию исполнителей. Большая специализация исполнителей. Ниже требования к стабильности команды. Отсутствуют ограничения по	Требуют существенной управленческой надстройки. Более длительные стадии анализа и проектирования. Более формализованные

	объему и сложности выполняемых проектов.	коммуникации.
Легкие	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. Объем и сложность выполняемых проектов ограничены.

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 (карго, англ. груз) 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 и надписи «USA». Все это для того чтобы снова с неба спустились самолеты и этих предметов стало больше.

Алистер Коуберн, один из авторов «Манифеста гибкой разработки ПО» [14] проанализировал очень разные программные проекты, которые выполнялись по разным моделям от совершенно облегченных и «гибких» до тяжелых (CMM-5) за последние 20 лет [15, 16]. Он не обнаружил корреляции между успехом или провалом проектов и моделями процесса разработки, которые применялись в проектах. Отсюда он сделал вывод о том, что эффективность разработки ПО 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 ПО, в каждом новом проекте процесс должен определяться каждый раз заново, в зависимости от проекта, продукта и персонала, в соответствие с «Законом 4-х П» (Рисунок 4). 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 ПО, например, система управления атомной электростанцией, то процесс разработки должен сильно отличаться от разработки, например, сайта «отдохни.ру». И, наконец, по-разному следует организовывать процесс разработки в команде вчерашних студентов и в команде состоявшихся профессионалов.

Рисунок 4. «Закон 4-х П». Процесс в проекте должен определяться в зависимости от проекта, продукта и персонала

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.

Что надо делать для успеха программного проекта

Стив Макконнелл в своей книге [17] приводит тест программного проекта на выживание. Этот чек-лист из 33-х пунктов, который я считаю необходимым процитировать с небольшими корректировками. Руководитель программного проекта должен его периодически использовать для внутреннего аудита своих процессов.

Чтобы программный проект стал успешным, необходимо:

1. Четко ставить цели.
2. Определять способ достижения целей.
3. Контролировать и управлять реализацией.
4. Анализировать угрозы и противодействовать им.
5. Создавать команду.

1. Ставим цели
   1.1. Концепция определяет ясные недвусмысленные цели.

1.2. Все члены команды считают концепцию реалистичной.

1.3. У проекта имеется обоснование экономической эффективности.

1.4. Разработан прототип пользовательского интерфейса.

1.5. Разработана спецификация целевых функций программного продукта.

1.6. С конечными пользователями продукта налажена двухсторонняя связь

2. Определяем способ достижения целей

2.1. Имеется детальный письменный план разработки продукта.

2.2. В список задач проекта включены «второстепенные» задачи (управление конфигурациями, конвертация данных, интеграция с другими системами).

2.3. После каждой фазы проекта обновляется расписание и бюджет.

2.4. Архитектура и проектные решения документированы.

2.5. Имеется план обеспечения качества, определяющий тестирование и рецензирование.

2.6. Определен план многоэтапной поставки продукта.

2.7. В плане учтены обучение, выходные, отпуска, больничные.

2.8. План проекта и расписание одобрен всеми участниками команды.

3. Контролируем и управляем реализацией

3.1. У проекта есть куратор. Это такой топ-менеджер исполняющей компании, который лично заинтересован в успехе данного проекта.

3.2. У проекта есть менеджер, причем только один!

3.3. В плане проекта определены «бинарные» контрольные точки.

3.4. Все заинтересованные стороны могут получить необходимую информацию о ходе проекта.

3.5. Между руководством и разработчиками установлены доверительные отношения.

3.6. Установлена процедура управления изменениями в проекте.

3.7. Определены лица, ответственные за решение о принятии изменений в проекте.

3.8. План, расписание и статусная информация по проекту доступна каждому участнику.

3.9. Код системы проходит автоматическое рецензирование.

3.10. Применяется система управления дефектами.

4. Анализируем угрозы

4.1. Имеется список рисков проекта. Осуществляется его регулярный анализ и обновление.

4.2. Руководитель проекта отслеживает возникновение новых рисков.

4.3. Для каждого подрядчика определено лицо, ответственное за работу с ним.

5. Работаем над созданием команды

5.1. Опыт команды достаточен для выполнения проекта.

5.2. У команды достаточная компетенция в прикладной области.

5.3. В проекте имеется технический лидер.

5.4. Численность персонала достаточна.

5.5. У команды имеется достаточная сплоченность.

5.6. Все участники привержены проекту.

Оценка и интерпретация теста

Оценка: сумма баллов, каждый пункт оценивается от 0 до 3:

- 0 – даже не слышали об этом;
- 1 – слышали, но пока не применяем;
- 2 – применяется частично;
- 3 – применяется в полной мере.

Поправочные коэффициенты:

- для малых проектов (до 5 человек) - 1.5;
- для средних (от 5 до 20 человек) – 1.25.

Результат:

- <40 – завершение проекта сомнительно.
- 40-59 – средний результат. В ходе проекта следует ожидать серьезные проблемы.
- 60-79 – хороший результат. Проект, скорее всего, будет успешным.
- 80-89 – отличный результат. Вероятность успеха высока.
- >90 – великолепный результат. 100% шансов на успех.

Этот чек-лист перечисляет, *что* надо делать для успеха программного проекта, но не дает ответ на вопрос *как* это следует делать. Именно об этом пойдет речь в остальных лекциях.

Выводы

То, что производят программисты нематериально – это коллективные мысли и идеи, выраженные на языке программирования. В силу уникальности отрасли опыт, накопленный в отраслях материального производства, мало способствует успеху в управлении программным проектом. Прямые аналогии с этими отраслями не работают. Управлять разработкой ПО надо иначе.

Не существует единственного правильного процесса разработки ПО. Эффективный производственный процесс должен основываться на итеративности, инкрементальности, самоуправляемости команды и адаптивности. Главный принцип: не люди должны строиться под выбранную модель процесса, а модель процесса должна подстраиваться под конкретную команду, чтобы обеспечить ее наивысшую производительность.

Чтобы программный проект стал успешным, необходимо:

1. Четко ставить цели.

2. Определять способ достижения целей.

3. Контролировать и управлять реализацией.

4. Анализировать угрозы и противодействовать им.

5. Создавать команду.

Дополнительная литература и источники

1. IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

2. EEE Std 1074-1995, IEEE Standard for Developing Software Life Cycle Processes.

3. «Руководство к своду знаний по программной инженерии». The Guide to the Software Engineering Body of Knowledge, SWEBOK, IEEE Computer Society Professional Practices Committee, 2004.

4. David Rubinstein, «Standish Group Report: There's Less Development Chaos Today». 2007 (http://www.sdtimes.com/content/article.aspx?ArticleID=30247)

5. Брукс Фредерик, «Мифический человеко-месяц, или Как создаются программные комплексы», Пер. с англ., СПб., Символ-Плюс, 1999.

6. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.

7. Уолкер Ройс, «Адаптивный стиль управления программными проектами». Открытые системы. 2006. № 1.

8. Ершов А. П., «О человеческом и эстетическом факторе в программировании». Информатика и образование. 1993. № 6.

9. Paulk, Mark C., and others, Capability Maturity Model for Software, Version 1.1 (CMU/SEI-93-TR-24). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1993.

10. Филипп Крачтен, «Введение в Rational Unified Process», Вильямс, 2002 г.

11. «MSF, Microsoft, Microsoft Solutions Framework», Отдел MSF, Microsoft, 2002.

12. M. Pomeroy-Huff, J. Mullaney, R. Cannon, M. Sebern, «The Personal Software Process (PSP) Body of Knowledge», version 1.0, SPECIAL REPORT CMU/SEI, 2005

13. Watts S. Humphrey, «The Team Software Process (TSP)», Technical Report CMU/SEI, 2000

14. Kent Beck, and others, «Manifesto for Agile Software Development», 2001 (http://www.agilemanifesto.org/)

15. А. Коуберн, «Люди как нелинейные и наиболее важные компоненты в создании программного обеспечения», Humans and Technology Technical Report, Oct.1999 (русский перевод К.Максимов, А.Максимова, http://www.maxkir.com/sd/people_as_nonlinearRUS.htm)

16. А. Коуберн, «Каждому проекту своя методология», Humans and Technology Technical Report, TR 99.04, Oct.1999 (русский перевод К.Максимов, А.Максимова, http://www.maxkir.com/sd/methyperproject_RUS.htm).

17. С. Макконнелл, «Остаться в живых. Руководство для менеджеров программных проектов», «Питер», 2006.

Лекция 2. Управление проектами. Определения и концепции

Проект - основа инноваций

Классическое управление проектами [1] выделяет два вида организации человеческой 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 (конвейер). Задача проекта – достижение конкретной бизнес-цели. Задача операционной деятельности – обеспечение нормального течения бизнеса.

Проект - это средство стратегического развития (Рисунок 5). 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.

Рисунок 5. Проект – средство стратегического развития

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.

Проекты и управление ими существовали всегда. В качестве самостоятельной области знаний управление проектами начало формироваться в начале XX 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. В России - СОВНЕТ.

Примерно 50 лет назад человечество начало жить в новой общественно-экономической формации, которая называется информационное или постиндустриальное общество. Мы живем в эпоху перемен, глобализации и интеллектуального капитала.

Эпоха перемен. Все в мире стало непрерывно и стремительно изменяться. Изобилие стало причиной острейшей конкуренции. Инновации - неотъемлемый атрибут нашего времени. «Если у вас медленный доступ в Интернет, вы можете навсегда отстать от развития информационных технологий». Практика должна постоянно перестраиваться применительно к новым и новым условиям. Пример.

Hewlett-Packard получает большую долю прибыли на товарах, которые год назад даже не существовали [2].

Глобализация. Всеобщая взаимозависимость и взаимосвязанность. Транснациональные компании. Бизнес идет туда, где дешевле рабочая сила. Интернет. Конкуренция без границ. Пример. Google. За ночь любой из нас в принципе может создать многомиллионную компанию у себя в гараже. С помощью Интернета вы можете выйти на рынок, на котором более 100 млн. потребителей.

Все решают таланты. Простая мобилизация средств и усилий уже не может обеспечить прогресс. Вспомним Ф. Брукса [3], «Если проект не укладывается в сроки, то добавление рабочей силы задержит его еще больше». Идею богатства теперь связывают не с деньгами, а с людьми, не с финансовым капиталом, а с «человеческим». Рынок труда превращается в рынок независимых специалистов и его участникам все больше известно о возможных вариантах выбора. Работники интеллектуального труда начинают самостоятельно определять себе цену.

Человечеству известны два вида деятельности. *Репродуктивная* деятельность (труд) является слепком, копией с деятельности другого человека либо копией своей собственной деятельности, освоенной в предшествующем опыте. Такая деятельность, как, например, труд токаря в любом механическом цеху, или рутинная повседневная деятельность менеджера-управленца на уровне раз и навсегда усвоенных технологий. *Продуктивная* деятельность (творчество) - деятельность, направленная на получение объективно нового или субъективно нового (для данного работника) результата.

Репродуктивная деятельность уходит в прошлое. В постиндустриальном обществе интеллект - основная производственная сила. Сегодня от 70 до 80% всего, что сегодня делается людьми, производится при помощи их интеллекта [4]. В любом товаре, сделанном в США, доля зарплаты составляет 70 процентов. Но это в среднем по всем товарам. Что касается разработки ПО, то почти все, что в этой отрасли производится, создается при помощи интеллекта.

Все меньший объем человеческой деятельности может быть организован в виде повторяющихся операций. Традиционный пример операционной деятельности – это работа бухгалтерии. Но жизнь так стремительно изменяется, что сегодня, по утверждению сведущих людей, подготовка и сдача годового финансового отчета каждый раз реализуется как самостоятельный проект.

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.

Критерии успешности проекта

Задача проекта – достижение конкретной бизнес-цели, при соблюдении ограничений «железного треугольника» (Рисунок 6). 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.

Рисунок 6. «Железный треугольник» ограничений проекта

Согласно текущей редакции стандарта PMBOK [1], 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 в 18:00 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.*

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», «Работай продуктивно, а предприятие о тебе позаботится» - уходят в прошлое. Посмотрите на рынок рабочей силы в ИТ - правила устанавливают профессионалы.

Проект и организационная структура компании

Организационная структура компании отражает ее внутреннее устройство, потоки управляющих воздействий, распределение труда и специфические особенности производства. Функциональная и проектная организации – противоположные полюса, а матричная организация – промежуточные состояния. Нет одной лучшей организационной структуры. Нет смысла противопоставлять функциональные структуры и проектные организации.

Синоним функциональной структуры - иерархическая структура (Рисунок 7).

Рисунок 7. Функциональная структура

Функциональная структура имеет следующие особенности:

- Сохраняется принцип единоначалия
- Понятные и стабильные условия работы
- Хорошо приспособлены для операционной деятельности.
- Специализация подразделений позволяет накапливать экспертизу.
- Затруднено принятие решений и коммуникации между исполнителями. Осуществляются только через руководство.
- Управление сконцентрировано и держится на компетенции высшего руководства

- Как правило, неэффективен контроль за ходом проекта (нет целостной картины)

Функциональная структура предполагает многоуровневую иерархию. Руководители функциональных подразделений это начальники управлений, начальники подчиненных им служб, отделов, лабораторий, секторов, групп. А еще у каждого начальника есть заместитель и, порой, не один. Примеры: министерства, ведомства, научные институты и предприятия советского периода.

На другом краю спектра организационных структур находится проектная структура (Рисунок 8).

Рисунок 8. Проектная структура

В чисто проектных организациях:

- Проект организуется как самостоятельное производственное подразделение.
- Персонал на проект набирается по временным контрактам.
- После завершения проекта персонал увольняется.
- Медленный старт.
- Опыт не аккумулируется.
- Команды не сохраняются.

Проектные организации не самые эффективные, но порой единственно возможные для выполнения проектов, которые физически удалённы от исполняющей организации, например, строительство нового нефтепровода.

В разработке ПО наиболее распространена матричная организация. Различают три вида матричной организационной структуры: слабая, сбалансированная и сильная (Рисунок 9 - Рисунок 11). Причем, в компаниях, которые занимаются продуктовой разработкой ПО, функциональные подразделения определяются в соответствие с линейкой продуктов. Например, отдел разработки CRM-систем, отдел разработки финансовых систем, отдел разработки дополнительных продуктов.

В компаниях, которые ориентированы в основном на заказную разработку ПО, функциональные подразделения чаще объединяются в соответствие с используемыми информационными технологиями. Например, отдел разработки баз данных, отдел разработки J2EE-приложений, отдел веб-разработок, отделы тестирования, документирования и т.д.

Рисунок 9. Слабая матрица

В слабой матрице роль и полномочия сотрудника, который координирует проект, сильно ограничены. Реальное руководство проектом осуществляет один из функциональных руководителей. Координатор проекта, его еще часто называют «трекер», помогает этому руководителю собирать информацию о статусе выполняемых проектных работ, учитывает затраты, составляет отчеты.

Рисунок 10. Сбалансированная матрица

Сбалансированная матрица характеризуется тем, что появляется менеджер проекта, который реально управляет выделенными на проект ресурсами. Он планирует работы, распределяет задачи среди исполнителей, контролирует сроки и результаты, несет полную ответственность за достижение целей проекта, при соблюдении ограничений.

В сбалансированных матрицах наиболее ярко проявляется проблема двойного подчинения. Руководитель функционального подразделения и менеджер проекта имеют примерно равное влияние на материальный и профессиональный рост разработчиков.

Рисунок 11. Сильная матрица

В сильной матрице признается, что проектное управление является самостоятельной областью компетенции, в которой необходимо накапливать экспертизу и использовать общие ресурсы. Поэтому в сильной матрице менеджеры проектов объединяются в самостоятельное функциональное подразделение - офис управления проектами (ОУП). ОУП разрабатывает корпоративные политики и стандарты в области проектного управления, планирует и осуществляет профессиональное развитие менеджеров.

Одной из особенностей матричных структур является то, что они становятся «плоскими», исчезает многоступенчатая иерархия. Предприятие, как правило, делится на функциональные отделы, в которых работают специалисты разных категорий, напрямую подчиняющиеся начальнику отдела. Начальники лабораторий, секторов, групп упраздняются за ненадобностью. В матричных структурах роль начальника функционального подразделения в производственном процессе заметно снижается, по сравнению с функциональными структурами. В его компетенции остаются вопросы стратегического развития функционального направления, планирование и развитие карьеры сотрудников, вопросы материально-технического обеспечения работ. Следует учитывать, что такое перераспределение полномочий и ответственности от функциональных руководителей к менеджерам проектов часто служит источником конфликтов в компаниях при их переходе от функциональной структуры к матричной.

Организация проектной команды

Каждый проект разработки ПО имеет свою организационную структуру, которая определяет распределение ответственности и полномочий среди участников проекта, а также обязанностей и отношений отчетности. Чем меньше проект, тем больше ролей приходится совмещать одному исполнителю.

Роли и ответственности участников типового проекта разработки ПО можно условно разделить на пять групп:

1. Анализ. Извлечение, документирование и сопровождение требований к продукту.

2. Управление. Определение и управление производственными процессами.

3. Производство. Проектирование и разработка ПО.

4. Тестирование. Тестирование ПО.

5. Обеспечение. Производство дополнительных продуктов и услуг.

Группа анализа включает в себя следующие роли:

- Бизнес-аналитик. Построение модели предметной области (онтологии).

- Бизнес-архитектор. Разрабатывает бизнес-концепцию системы. Определяет общее видение продукта, его интерфейсы, поведение и ограничения.

- Системный аналитик. Отвечает за перевод требований к продукту в функциональные требования к ПО.

- Специалист по требованиям. Документирование и сопровождение требований к продукту.

- Менеджер продукта (функциональный заказчик). Представляет в проекте интересы пользователей продукта.

Группа управления состоит из следующих ролей:

- Руководитель проекта. Отвечает за достижение целей проекта при заданных ограничениях (по срокам, бюджету и содержанию), осуществляет операционное управление проектом и выделенными ресурсами.

- Куратор проекта. Оценка планов и исполнения проекта. Выделение ресурсов.

- Системный архитектор. Разработка технической концепции системы. Принятие ключевых проектных решений относительно внутреннего устройства программной системы и её технических интерфейсов.

- Руководитель группы тестирования. Определение целей и стратегии тестирования, управление тестированием.

- Ответственный за управление изменениями, конфигурациями, за сборку и поставку программного продукта.

В производственную группу входят:

- Проектировщик. Проектирование компонентов и подсистем в соответствие с общей архитектурой, разработка архитектурно значимых модулей.

- Проектировщик базы данных.
- Проектировщик интерфейса пользователя.
- Разработчик. Проектирование, реализация и отладка отдельных модулей системы.

В большом проекте может быть несколько производственных групп, ответственных за отдельные подсистемы. Как правило, проектировщик выполняет роль лидера группы и управляет своим подпроектом или пакетом работ. Стоит не забывать, что руководитель проекта делегирует полномочия, но не ответственность.

Группа тестирования в проекте состоит из следующих ролей:

- Проектировщик тестов. Разработка тестовых сценариев.
- Разработчик автоматизированных тестов.
- Тестировщик. Тестирование продукта. Анализ и документирование результатов.

Участники группы обеспечения, как правило, не входят в команду проекта. Они выполняют работы в рамках своей процессной деятельности. К группе обеспечения можно отнести следующие проектные роли:

- Технический писатель.
- Переводчик.
- Дизайнер графического интерфейса.
- Разработчик учебных курсов, тренер.
- Участник рецензирования.
- Продажи и маркетинг.
- Системный администратор.
- Технолог.
- Специалист по инструментальным средствам.
- Другие.

В зависимости от масштаба проекта одну роль могут исполнять несколько человек. Например, разработчики, тестировщики, технические писатели. Некоторые роли всегда должен исполнять только один человек. Например, Руководитель проекта, Системный архитектор. Один человек может исполнять несколько ролей. Возможны следующие совмещения ролей:

- Руководитель проекта + системный аналитик (+ системный архитектор)
- Системный архитектор + разработчик
- Системный аналитик + проектировщик тестов (+ технический писатель)

- Системный аналитик + проектировщик интерфейса пользователя
- Ответственный за управление конфигурациями + ответственный за сборку и поставку (+ разработчик)

Крайне нежелательно совмещать следующие роли:

- Разработчик + руководитель проекта
- Разработчик + системный аналитик.
- Разработчик + проектировщик интерфейсов пользователя.
- Разработчик + тестировщик

Не раз приходилось наблюдать, как в критические периоды проекта его менеджер-разработчик с увлечением правит очередные баги, а проектная команда в полном составе стоит у него за спиной и наблюдает за этим процессом. Это плохой пример руководства проектом.

Программисты любят и умеют программировать. Пусть они этим и занимаются. Не стоит загружать программистов несвойственной для них работой. В каждом проекте разработки программного продукта много других работ: бизнес-анализ, проектирование эргономики, графический дизайн, разработка пользовательской документации. Эти работы с программированием не имеют ничего общего. Для них требуются совершенно другая квалификация и другой склад мышления.

При кустарном производстве программ эти задачи, как правило, поручаются программистам, которые это делать не умеют и не любят. Получается обычно плохо, да еще и дорого. В силу своей интроверсии, граничащей с аутизмом, программист просто не в состоянии увидеть свою программу чужими глазами – глазами пользователей. Никто уже не хочет работать с программами с технологической парадигмой навороченного пользовательского интерфейса - кустарным творением программистов - когда для того чтобы работать с системой, надо обязательно знать, как она устроена. Это типичное творение программиста, которому гораздо важнее видеть, как работает его программа, чем разбираться в том, что она делает для пользователя. Поэтому, необходимо привлекать в проектную команду бизнес-аналитиков, эргономистов, художников-дизайнеров, документалистов. Разделение труда и специализация - залог перехода от кустарного производства к более эффективному промышленному производству.

Из профессиональных программистов получаются отличные тестировщики. Лучшая команда тестирования, которую я встречал, была в Luxoft. Это были маститые программисты из одного академического НИИ с опытом 20-30 лет. Они не осваивали новые программистские технологии, но исключительно эффективно ломали то, что было сделано на их основе. Однако, совмещать одновременно роли программиста и тестировщика – плохая практика. Хороший программист убежден, что он пишет программы правильно и ему психологически тяжело допустить, что где-то в его коде может быть ошибка. А ошибки есть всегда!

Организационная структура проекта обязательно должна включать в себя эффективную систему отчетности, оценки хода выполнения проекта и систему принятия решений. Можно рекомендовать еженедельные собрания по статусу проекта, на которых анализируются риски, оцениваются результаты, достигнутые на предыдущей неделе, и уточняются задачи на новый период.

В модели Scrum рекомендуются ежедневные совещания по состоянию работ – «Stand Up Meeting», но мне кажется, что это применимо, скорее, для небольших рабочих групп от 3 до 5 разработчиков. Хотя в критические периоды проекта, приходилось проводить и ежедневные совещания.

Важно помнить, что организационная структура проекта – «живой» организм. Она начинает складываться на стадии планирования и может меняться в ходе проекта. Нестабильность организационной структуры (частые замены исполнителей) – серьезная проблема в управлении сложными программными проектами, поскольку существует время вхождения в контекст проекта, которое может измеряться месяцами.

Жизненный цикл проекта. Фазы и продукты

Ранее уже отмечалось, что каждый программный продукт имеет свой жизненный цикл, в который проект разработки очередного релиза входит как одна из фаз. Аналогично, каждый проект разработки ПО имеет свой собственный жизненный цикл, который состоит из четырех фаз (Рисунок 12).

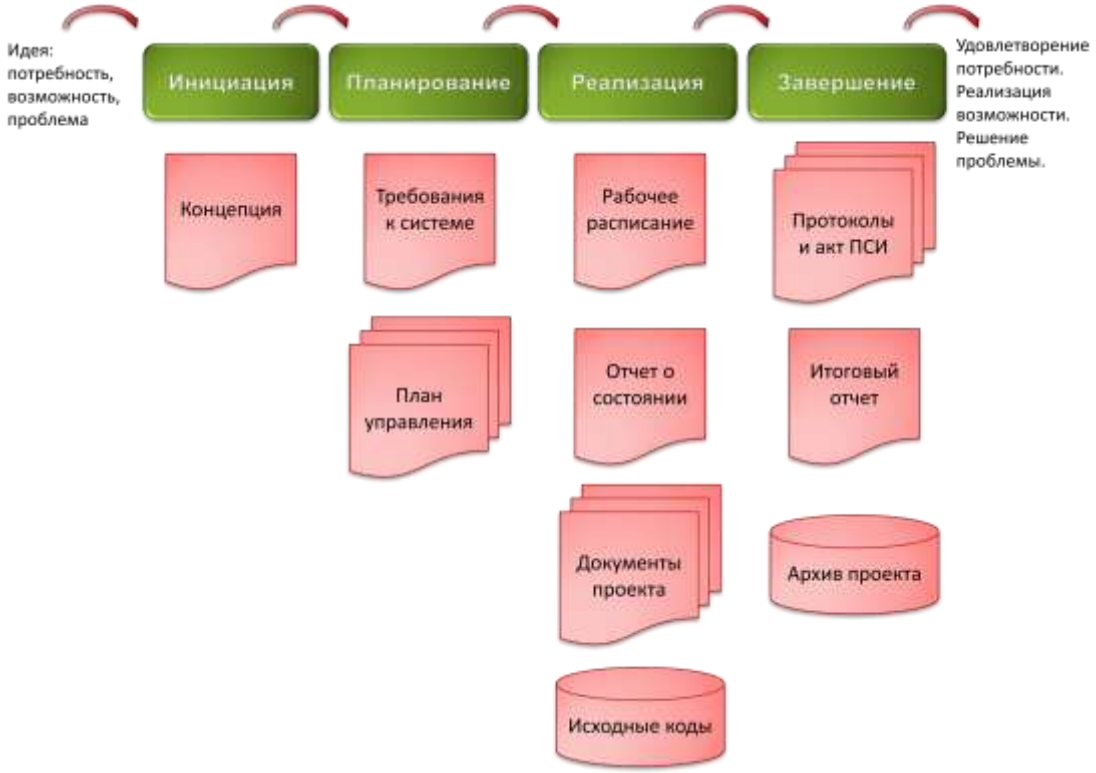

Рисунок 12. Жизненный цикл и основные продукты программного проекта

На фазе инициации проекта необходимо понять, *что и зачем* мы будем делать – разработать концепцию проекта. Фаза планирования определяет, *как* мы будем это делать. На фазе реализации происходит материализация наших идей в виде документированного и протестированного программного продукта. И, наконец, на фазе завершения мы должны подтвердить, что мы разработали именно тот продукт, который задумали в концепции проекта, а также провести приемо-сдаточные испытания (ПСИ) продукта на предмет соответствия его свойств, определенным ранее требованиям.

Как правило, редкий проект выполняется в соответствие с первоначальными планами, поэтому важным элементом фазы завершения является «обратная

связь»: анализ причин расхождения и усвоение уроков на будущее. Помним, что управляющая система без обратной связи не может быть устойчивой.

Более подробно о каждой фазе проекта и их продуктах будет рассказано в последующих лекциях.

Завершая обзор управления проектами «с высоты птичьего полета», необходимо упомянуть еще об одной особенности проекта по сравнению с операционной деятельностью. Если в операционной деятельности ресурсы расходуются более-менее равномерно по времени, то в проектном управлении расходование ресурсов в единицу времени имеет явно выраженное колоколообразное распределение (Рисунок 13)

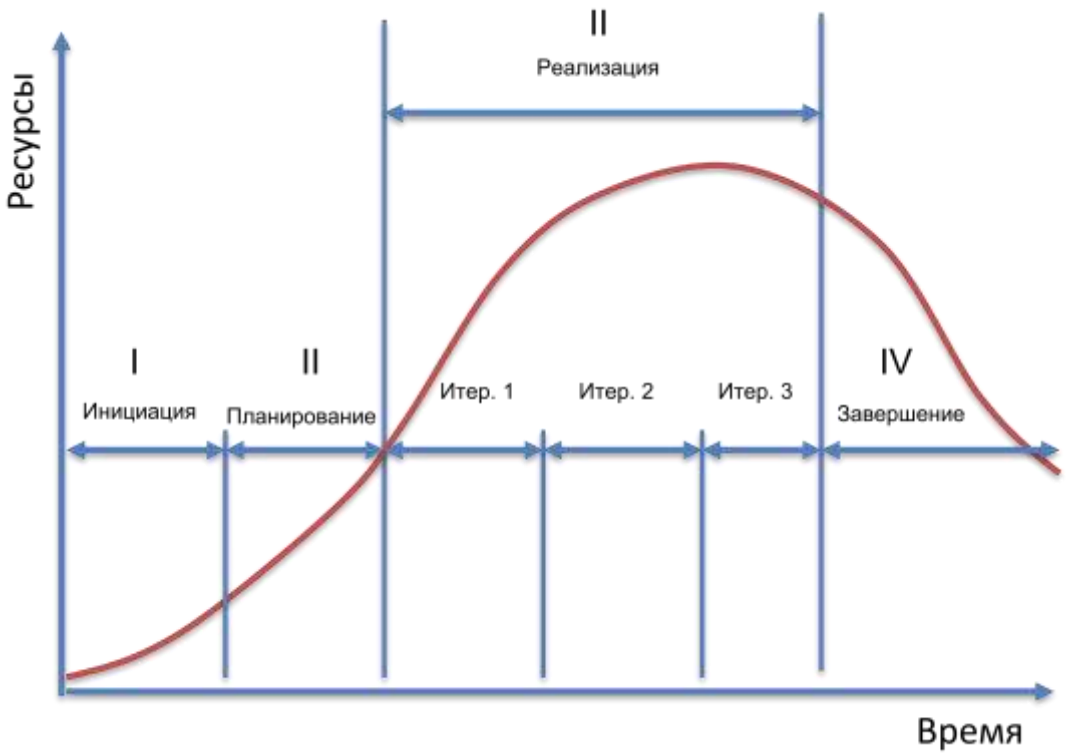

Рисунок 13. Распределение ресурсов по фазам проекта

Проект часто начинается с идеи, которая появляется у одного человека. Постепенно, по мере формулирования, анализа и оценки этой идеи, привлекаются дополнительные специалисты. Еще больше участников требуется на фазе планирования проекта. Пик потребления ресурсов приходится на фазу реализации.

В современных моделях разработки ПО реализация осуществляется на основе сочетания итеративного и инкрементального подходов.

Итеративность предполагает, что требования к системе и ее архитектура прорабатываются не один раз, а постепенно уточняются от итерации к итерации. Это означает, что на каждой итерации происходит полный цикл процессов разработки: уточнение требований, проектирование, кодирование, тестирование и документирование.

Инкрементальность состоит в том, что результатом каждой итерации является версия ПО, которая реализует часть функциональности будущего программного продукта и может быть введена в тестовую или опытную эксплуатацию, а также оценена заказчиком и будущими пользователями. Это означает, что после каждой итерации происходит прирост требуемого функционала, а нереализованных функций будущего продукта остается все меньше.

Сочетание итеративности и инкрементальности обеспечивает эффективность разработки и существенное снижение рисков по ходу проекта. Об этом мы еще будем говорить.

На последней фазе происходит постепенное высвобождение участников проектной команды. Следует помнить, что проект должен иметь четкое окончание во времени, после которого все работы по проекту закрываются, и на проект перестают тратиться ресурсы. Не должно оставаться «зависших» работ.

Выводы

Проект - этот средство стратегического развития. 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.

Участников типового проекта разработки ПО можно условно разделить на пять групп ролей:

1. Анализ. Извлечение, документирование и сопровождение требований к продукту.
2. Управление. Определение и управление производственными процессами.
3. Производство. Проектирование и разработка ПО.
4. Тестирование. Тестирование ПО.
5. Обеспечение. Производство дополнительных продуктов и услуг.

У программного проекта имеется четыре фактора, которые определяют его успешность:

1. Выполнен в соответствие со спецификациями.
2. Выполнен в срок.
3. Выполнен в пределах бюджета.
4. Каждый участник команды уходил с работы в 18:00 с чувством успеха.

Дополнительная литература и источники

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
2. Стивен Р. Кови, «7 навыков высокоэффективных людей. Мощные инструменты развития личности», 2-е изд., М., Альпина Бизнес Букс, 2007

3. Брукс Фредерик, "Мифический человеко-месяц, или Как создаются программные комплексы", Пер. с англ., СПб., Символ-Плюс, 1999.

4. Кьелл А. Нордстрем, Йонас Риддерстрале, «Бизнес в стиле фанк. Капитал пляшет под дудку таланта», Стокгольмская школа экономики в Санкт-Петербурге, 2005.

Лекция 3. Инициация проекта

Управление приоритетами проектов

Эффективные процессы инициации программного проекта минимум наполовину определяют его будущую успешность. Недостаточное внимание именно этой фазе проекта неизбежно приводит к существенным проблемам при планировании, реализации и завершении проекта.

Инициация состоит из процессов, способствующих формальной авторизации начала нового проекта или фазы проекта. Процессы инициации часто выполняются вне рамок проекта и связаны с организационными, программными или портфельными процессами. В ходе процесса инициации уточняются первоначальное описание содержания и ресурсы, которые организация планирует вложить. На этом этапе также выбирается менеджер проекта, если он еще не назначен, и документируются исходные допущения и ограничения. Эта информация заносится в Устав проекта и, если он одобряется, проект официально авторизуется.

Устав проекта [1] - документ, выпущенный инициатором или спонсором проекта, который формально узаконивает существование проекта и предоставляет менеджеру проекта полномочия использовать организационные ресурсы в операциях проекта.

В российской практике данный документ чаще называется Концепция проекта. *Концепция* (от лат. conceptio — понимание, система), определённый способ понимания, трактовки какого-либо предмета, явления, процесса, основная точка зрения на предмет и др., руководящая идея для их систематического освещения.

В компании, которая принимает решение о старте того или иного проекта разработки ПО, должна существовать единая система критериев для оценки его значимости. Система критериев должна позволять из множества возможных для реализации проектов выбрать наиболее приоритетные для компании.

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

- Финансовая ценность.
- Стратегическая ценность.
- 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 обоснованны.
- *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.

Например. Финансовая ценность проектов разработки ПО, проектов внедрения или сопровождения, которые выполняются в соответствие с заключенными коммерческими договорами, может быть оценена как высокая. Проект планового развития функциональности продуктов в соответствии с требованиями рынка, инициируемое менеджером продукта на основе анализа предложений отделов маркетинга, консалтинга, продаж и технической поддержки, может получить оценку финансовой ценности выше среднего, а проекты изменения технологических процессов или проекты внутренней автоматизации могут иметь среднюю финансовую ценность.

Одной финансовой ценности для определения приоритета проекта недостаточно. Например, ни одна компания разработчик ПО 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 выполнению 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 сверхвыгоды 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.

Концепция проекта

У каждого проекта должна быть концепция. Если проект небольшой, то для изложения концепции часто достаточно несколько абзацев. Однако, стартовать проект без концепции, это все равно, что отправлять корабль в плавание, не определив для него пункт назначения.

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

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

Название проекта

Цели проекта

Результаты проекта

Допущения и ограничения

Ключевые участники и заинтересованные стороны

Ресурсы проекта

Сроки

Риски

Критерии приемки

Обоснование полезности проекта

В качестве примера, который позволит иллюстрировать теоретическое изложение основ управления проектами, возьмем реальный проект разработки ПО для автоматизации одного из подразделений крупной производственной компании. Назовем его «Автоматизированная система продажи документации».

Краткая легенда проекта. Заказчик ОАО «XYZ» является одним из ведущих производителей сложных технических изделий. Отдел «123», входящий в ОАО «XYZ», отвечает за продажу дополнительной сопроводительной документации для клиентов ОАО.

Дополнительная документация не входит в стандартную поставку, поскольку владелец этого технического изделия не всегда сам его эксплуатирует, а передает в эксплуатацию другой компании, которая становится клиентом «XYZ», и закупает у нее эксплуатационную документацию. Ремонт и техобслуживание конкретного изделия может выполнять третья компания, которой уже потребуется детальная техническая документация по ремонту и обслуживанию. Она также становится клиентом «XYZ» и закупает у нее требуемую продукцию.

Основная функция отдела «123» - получение и обработка заказов на дополнительную документацию, согласно ежегодно рассылаемому каталогу. В связи с переездом отдела «123» в новое здание, была поставлена задача на разработку и поставку системы, автоматизирующей основную деятельность отдела «123».

Текст документа Концепция проекта, который будет приводиться в качестве примера, будем выделять цветом фона.

Цели и результаты проекта

Цели проекта должны отвечать на вопрос, *зачем* данный проект нужен. Цели проекта должны описывать бизнес-потребности и задачи, которые решаются в результате исполнения проекта. Целями проекта могут быть:

- Изменения в Компании. Например, автоматизация ряда бизнес-процессов для повышения эффективности основной производственной деятельности
- Реализация стратегических планов. Например, завоевание значительной доли растущего рынка за счет вывода на него нового продукта.
- Выполнение контрактов. Например, разработка программного обеспечения по заказу.
- Разрешение специфических проблем. Например, доработка программного продукта в целях приведения его в соответствие с изменениями в законодательстве.

Цели должны быть значимыми (направленными на достижение стратегических целей Компании), конкретными (специфичными для данного проекта),

измеримыми (т.е иметь проверяемые количественные оценки), реальными (достижимыми). Четкое определение бизнес-целей важно, поскольку существенно влияет на все процессы и решения в проекте. Проект должен быть закрыт, если признается, что достижение цели невозможно или стало нецелесообразным. Например, если реальные затраты на проект будут превосходить будущие доходы от его реализации.

Результаты проекта отвечают на вопрос, *что* должно быть получено после его завершения. Результаты проекта 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.

Следует помнить, что результаты проекта должны быть измеримыми. Это означает, что при оценке результатов проекта должна иметься возможность сделать заключение достигнуты оговоренные в концепции результаты или нет.

Соответствующий раздел документа концепция проекта создания «Автоматизированной системы продажи документации» будет выглядеть следующим образом.

1. Цели и результаты проекта
   - 1.1. Целью проекта является повышение эффективности основной производственной деятельности отдела «123».
   - 1.2. Дополнительными целями проекта являются:
     - 1.2.1. Установление долгосрочных отношений с важным заказчиком ОАО «XYZ».
     - 1.2.2. Выход на новый перспективный рынок современных B2C систем.
2. Результаты проекта должны обеспечить:
   - 2.1. Снижение затрат на обработку заявок.
   - 2.2. Снижение сроков обработки заявок.
   - 2.3. Повышение оперативности доступа к информации о наличии продукции.
   - 2.4. Повышение оперативности доступа к информации о прохождении заявок.
   - 2.5. Повышение надежности и полноты хранения информации о поступивших заявках и результатах их обработки.
3. Продуктами проекта являются:
   - 3.1. Прикладное ПО и документация пользователей.
   - 3.2. Базовое ПО.
   - 3.3. Оборудование ЛВС, рабочие станции, сервера и операционно-системное ПО.

3.4. Проведение пуско-наладочных работ и ввод в опытную эксплуатацию.

3.5. Обучение пользователей и администраторов системы.

3.6. Сопровождение системы на этапе опытной эксплуатации.

3.7. Передача системы в промышленную эксплуатацию.

4. Система должна автоматизировать следующие функции:

4.1. Авторизация и аутентификация пользователей.

4.2. Просмотр каталога продуктов.

4.3. Поиск продуктов по каталогу.

4.4. Заказ выбранных продуктов.

4.5. Просмотр информации о статусе заказа.

4.6. Информирование клиента об изменении статуса заказа.

4.7. Просмотр и обработка заказов исполнителями из службы продаж.

4.8. Просмотр статистики поступления и обработки заказов за период.

4.9. Подготовка и сопровождение каталога продукции.

Допущения и ограничения

Данный раздел описывает исходные допущения и ограничения. Допущения, как правило, тесно связаны с управлением рисками, о котором мы будем говорить далее. В разработке ПО часто приходится формулировать риски в виде допущений, тем самым передавая его заказчику. Например, оценивая проект разработки и внедрения по схеме с фиксированной ценой, мы должны записать в допущения предположение о том, что стоимость лицензий на стороннее ПО не изменится, до завершения проекта.

Ограничения, как правило, сокращают возможности проектной команды в выборе решений. В частности они могут содержать:

- Специфические нормативные требования. Например, обязательная сертификация продукта, услуги на соответствие определенным стандартам.
- Специфические технические требования. Например, разработка под заданную программно-аппаратную платформу.
- Специфические требования к защите информации.

В этом разделе также уместно сформулировать те требования к системе, которые могут ожидаться заказчиком по умолчанию, но не включаются в рамки данного проекта. Например, в данный раздел может быть включен пункт о том, что разработка программного интерфейса (API) для будущей интеграции с другими системами заказчика не входит в задачи данного проекта.

Содержание этого раздела для нашего проекта-примера выглядит следующим образом.

5. Допущения и ограничения

   5.1. Проектирование прикладного ПО выполняется с использованием UML[1].

   5.2. Средством разработки ПО является Symantec Visual Cafe for Java[2].

   5.3. В качестве промежуточного ПО сопровождения и поддержки каталога используется ОО БД «Poet»[3].

   5.4. Нагрузка на систему не должна быть более 100 одновременно работающих пользователей.

   5.5. В рамки проекта не входят:

      5.5.1. Защита системы от преднамеренного взлома.

      5.5.2. Разработка B2B API и интеграция с другими системами.

Ключевые участники и заинтересованные стороны

Одна из задач фазы инициации проекта это выявить и описать всех его участников. Согласно [1] к участникам проекта относятся все заинтересованные стороны (stakeholders), 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 также могут влиять на проект и его результаты поставки.

К ключевым участникам программного проекта, как правило, относятся:

- *Спонсор проекта* - лицо или группа лиц, предоставляющая финансовые ресурсы для проекта в любом виде.
- *Заказчик проекта* - лицо или организация, которые будут использовать продукт, услугу или результат проекта. Следует учитывать, что заказчик и спонсор проекта не всегда совпадают.
- *Пользователи* результатов проекта.
- *Куратор проекта* - представитель исполнителя, уполномоченный принимать решение о выделении ресурсов и изменениях в проекте.

[1] Проект стартовал в 2000 году, тема UML тогда была на слуху и даже оставались те, кто верил, что из модели на UML можно будет генерировать исходный код.
[2] Таково было требование Заказчика, поскольку этот инструмент использовали его программисты, которым предполагалось передавать систему на сопровождение.
[3] Еще один пример горячей темы и не оправдавшихся надежд – это объектно-ориентированные базы данных. У заказчика проекта уже были закуплены лицензии на эту базу данных и он очень хотел получить возврат от этих инвестиций. Поэтому ее использование в проекте стало одним из требований. К счастью, нам удалось быть достаточно убедительными и обосновать необходимость дополнительно использовать RDBMS Oracle для решения транзакционных задач. О том, к чему это привело, я подробно рассказал в своей книге: С. Архипенков, "Руководство командой разработчиков программного обеспечения. Прикладные мысли", Москва, 2008 (http://www.happy-pm.com/sw_team_management.pdf).

- *Руководитель проекта* - представитель исполнителя, ответственный за реализацию проекта в срок, в пределах бюджета и с заданным качеством.

- *Соисполнители проекта*. Субподрядчики и поставщики.

Содержание этого раздела в концепции-примере будет иметь вид.

6. Ключевые участники и заинтересованные стороны

   6.1. Спонсор проекта - директор Департамента информатизации ОАО «XYZ» В.Васильев.

   6.2. Заказчик – начальник Отдела «123» Ф.Федотов

   6.3. Пользователи автоматизированной системы:

   6.4. Клиенты ОАО «XYZ» (поиск и заказ документации).

   6.5. Руководство ОАО «XYZ» (анализ деятельности Отдела «123»).

   6.6. Сотрудники производственных департаментов ОАО «XYZ» (сопровождение каталога).

   6.7. Сотрудники Отдела «123» (обработка заявок и поставка документации).

   6.8. Сотрудники департамента информатизации ОАО «XYZ» (администрирование системы).

   6.9. Куратор проекта - начальник отдела заказных разработок И.Иванов.

   6.10. Руководитель проекта - ведущий специалист отдела заказных разработок МП П.Петров.

7. Соисполнители:

   7.1. Поставщик оборудования и операционно-системного ПО - ООО «Альфа».

   7.2. Поставщик базового ПО - ООО «Бета».

Ресурсы

Для того чтобы понять, сколько будет стоить реализация программного проекта, требуется определить и оценить ресурсы необходимые для его выполнения:

- Людские ресурсы и требования к квалификации персонала.

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

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

Специфика программного проекта заключается в том, что людские ресурсы вносят основной вклад в его стоимость. Все остальные затраты, как правило, незначительны, по сравнению с этим расходами. О том, как следует подходить к

оценкам трудозатрат на реализацию проекта разработки ПО, мы будем подробно говорить в следующих лекциях. На фазе инициации хорошей считается оценка трудозатрат с точностью от -50% до +100% [2].

Необходимо помнить, что помимо непосредственно программирования в проекте разработки ПО есть много других процессов, которые требуют ресурсы соответствующей квалификации, а само программирование составляет лишь четверть всех затрат. Распределение трудозатрат по основным производственным процессам при современном процессе разработки ПО выглядит в среднем следующим образом –

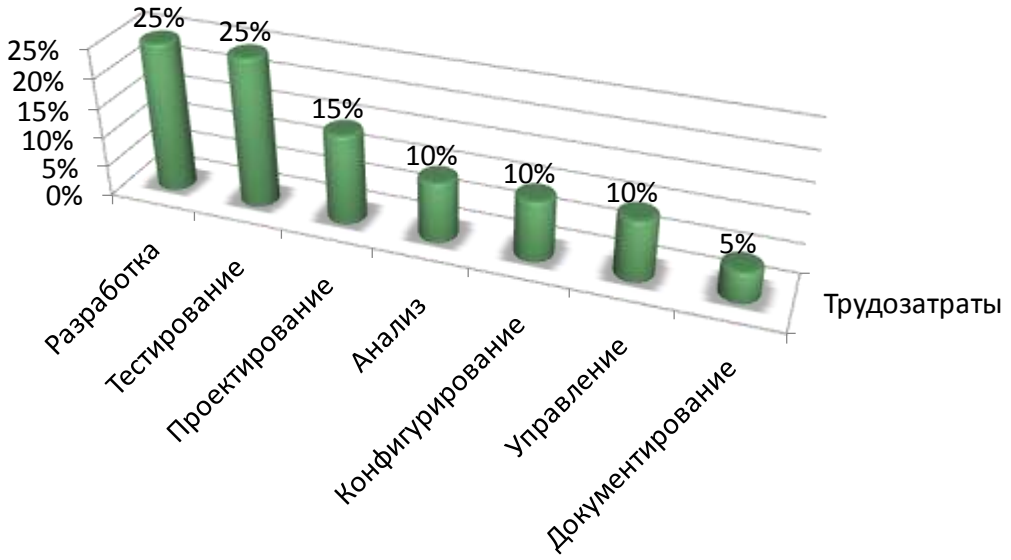

Рисунок 14.Распределение трудозатрат по основным производственным процессам при разработке ПО

Поэтому, если по вашей оценки для реализации требуемой функциональности в проекте необходимо написать 10 KSLOC (тысяч строк исходного программного кода), а ваши программисты пишут в среднем по 100 SLOC в день, то общие трудозатраты на проект будут не 100 чел.*дней, а не менее чем 400 чел.*дней. Остальные ресурсы потребуются на анализ и уточнение требований, проектирование, документирование, тестирование и другие проектные работы.

Прежде, чем определять численность и состав проектной команды для нашего примера, нам необходимо сделать оценку трудоемкости разработки ПО. В нашем случае такая экспертная оценка составила с учетом затрат на гарантийное сопровождение на этапе опытной эксплуатации 9000 чел.*час. Исходя из эмпирической кривой Б. Боэма (Рисунок 15), численность команды, близкая к оптимальной, составила 10 человек, из них

8. Ресурсы проекта

8.1. Требования к персоналу

8.1.1. 1 - руководитель проекта,

8.1.2. 1 - технический лидер (архитектура, проектирование),

8.1.3. 1 - системный аналитик (требования, тест-дизайн, документирование),

8.1.4. 4 - программисты (с учетом работ по конфигурационному управлению),

8.1.5. 3 - тестировщика.

8.2. Материальные и другие ресурсы

8.2.1. Сервер управления конфигурациями и поддержки системы контроля версий

8.2.2. 2 серверных комплекса (для разработки и тестирования):

8.2.3. Сервер приложений с установленным BEA Weblogic AS

8.2.4. Сервер оперативной БД с установленной Oracle RDBMS

8.2.5. Сервер каталога с установленной OODB “Poet”

8.3. Лицензии на средства разработки и тестирования:

8.3.1. Oracle Designer – 1 лицензия

8.3.2. Symantec Visual Cafe for Java - 5 лицензий.

8.3.3. IBM Rational Test Robot (1 лицензия разработчика + неограниченная лицензия на клиент).

8.4. Расходная часть бюджета проекта[4]

8.4.1. Разработка и сопровождение прикладного ПО:

8.4.1.1. 9000 чел.*час. * $40 = $360 000

8.4.2. Поставка оборудования и операционно-системного ПО:

8.4.2.1. 3 сервера * $10 000 = $30 000

8.4.3. Поставка базового ПО:

8.4.3.1. BEA Weblogic AS $20 000

8.4.3.2. Oracle RDBMS $20 000

Итого: $430 000

Сроки

Ф. Брукс [3] писал: «Чтобы родить ребенка требуется девять месяцев независимо от того, сколько женщин привлечено к решению данной задачи. Многие задачи программирования относятся к этому типу, поскольку отладка по своей сути носит последовательный характер».

Там же Брукс приводит исключительно полезную, но почему-то редко применяемую, эмпирическую формулу оценки срока проекта по его трудоемкости. Формула была выведена Барии Боэмом (Barry Boehm) на основе

[4] Мы в данном разделе оцениваем себестоимость проекта. Определение продажной цены проекта не входит в рамки данного курса.

анализа результатов 63 проектов разработки ПО, в основном в аэрокосмической области. Согласно этой формуле, для проекта, общая трудоемкость которого составляет N ч.*м. (человеко-месяцев), пожно утверждать что:

- Существует оптимальное, с точки зрения затрат, время выполнения графика для первой поставки: T = 2,5 (N ч.*м.)^1/3. То есть оптимальное время в месяцах пропорционально кубическому корню предполагаемого объема работ в человеко-месяцах. Следствием является кривая, дающая оптимальную численность проектной команды (Рисунок 15).

- Кривая стоимости медленно растет, если запланированный график длиннее оптимального. Работа занимает все отведенное для нее время.

- Кривая стоимости резко растет, если запланированный график короче оптимального. Практически ни один проект невозможно завершить быстрее, чем за ¾ расчетного оптимального графика вне зависимости от количества занятых в нем! (Рисунок 16)

Этот примечательный результат дает менеджеру программного проекта солидное подкрепление, когда высшее руководство требует принятия невозможного графика.

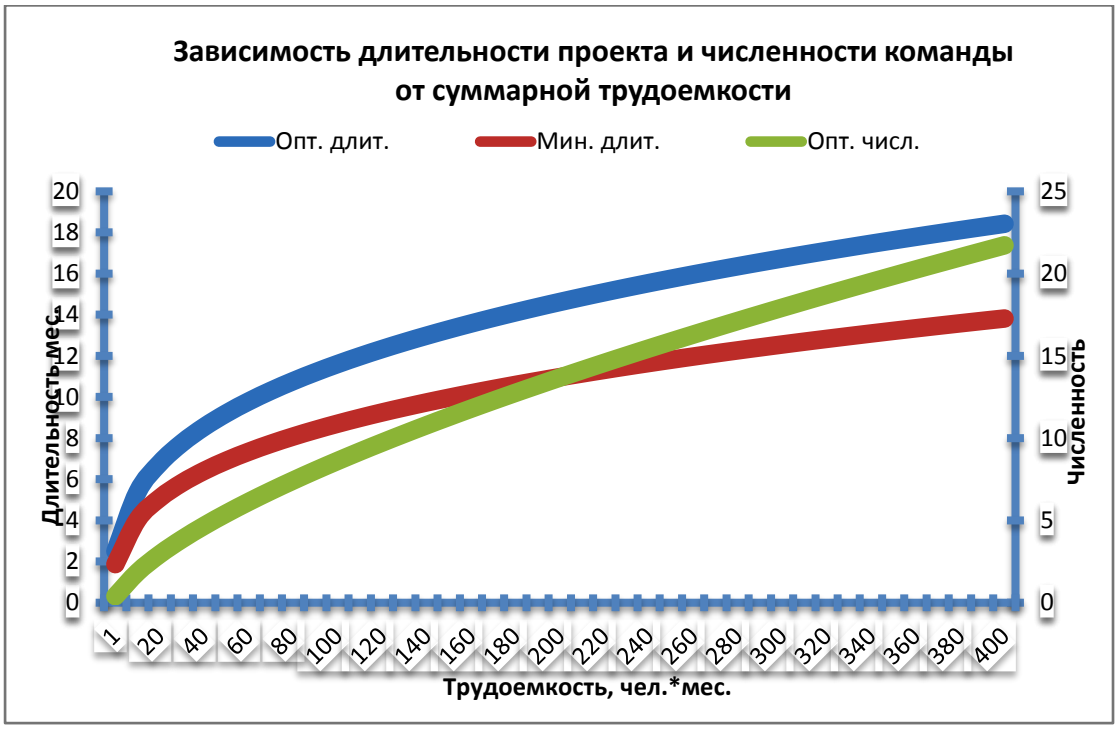

Рисунок 15. Закон Б.Боэма

Для сколь-нибудь серьезного программного проекта недостаточно определить только срок его завершения. Необходимо еще определить его этапы – контрольные точки, 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.

Как мы говорили ранее, современный проект разработки ПО должен реализовываться с применением инкрементального процесса. В этом случае контрольные точки должны соответствовать выпуску каждой промежуточной версии ПО, 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 8 недель.

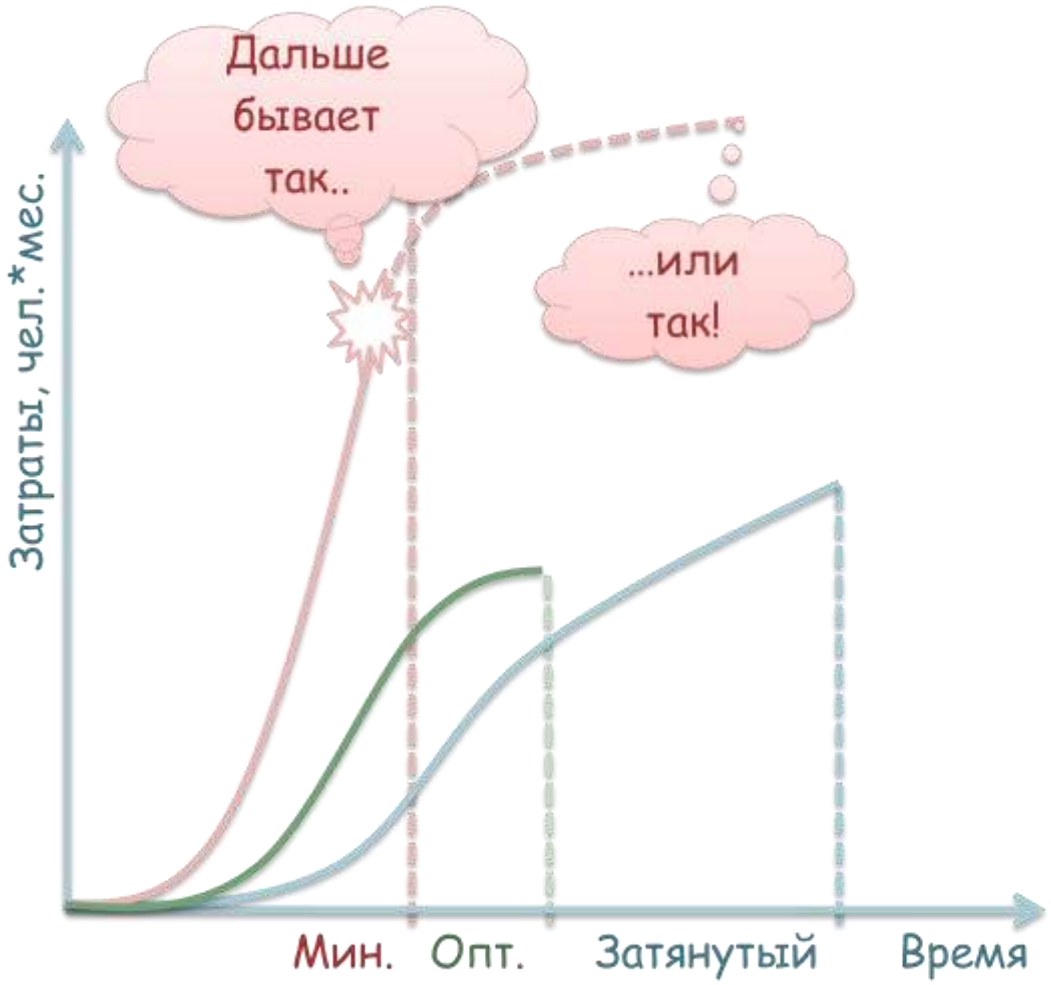

Рисунок 16. Следствия закона Б.Боэма

Соответствующий раздел концепции нашего проекта-примера будет иметь следующий вид.

9. Сроки проекта

    9.1. 03.03 старт

    9.2. 28.11 завершение

    9.3. Контрольные точки:

        9.3.1. 15.04 ТЗ утверждено

9.3.2. 30.04 1-я итерация завершена. Подсистема заказа документации передана в тестовую эксплуатацию (на серверах разработчика).

9.3.3. 15.05 Монтаж оборудования у заказчика завершен .

9.3.4. 30.05 Базовое ПО установлено у заказчика.

9.3.5. 15.06 2-я итерация завершена. Подсистема обработки заказов передана в тестовую эксплуатацию на оборудовании Заказчика

9.3.6. 02.09 3-я итерация завершена. Акт передачи системы в опытную эксплуатацию утвержден

9.3.7. 28.11 Система передана в промышленную эксплуатацию.

Риски

Риск - неопределенное событие или условие, наступление которого отрицательно или положительно сказывается на целях проекта [1].

Как правило, в случае возникновения негативного риска, почти всегда стоимость проекта увеличивается и происходит задержка в выполнении мероприятий, предусмотренных расписанием проекта. Управлению рисками проекта будет посвящена отдельная лекция.

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

В случае нашего проекта-примера раздел «риски» будет выглядеть следующим образом.

10. Риски проекта

10.1. Задачи системы поняты недостаточно полно. Понимание масштаба и рамок проекта недостаточно. Системы создаются на новой технологической платформе, сомнения в рыночной стабильности платформы. Суммарный уровень рисков следует оценить выше среднего.

Критерии приемки

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.

В рассматриваемом примере раздел «Критерии приемки» будет выглядеть следующим образом:

11. Критерии приемки. По итогам опытной эксплуатации система должна продемонстрировать следующие показатели:

11.1. Средние затраты сотрудников Отдела «123» на регламентную обработку одного

заказа не превышают 4 чел.*час.

11.2. Срок регламентной обработки 1-го заказа не более 2 недель.

11.3. Время поиска и предоставления информации о наличии дополнительной документации не более 1 мин.

11.4. Время предоставления информации о сделанных заказах и истории их обработки не более 1 мин.

11.5. Система хранит всю информацию о сделанных заказах и истории их обработки.

11.6. Показатель доступности системы 98%.

Обоснование полезности проекта

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

- Для кого предназначены результаты проекта.
- Описание текущей ситуации «As Is». Какие у потенциального заказчика существуют проблемы.
- Каким образом результаты проекта решают эти проблемы («To Be»).
- Насколько значимо для клиента решение данных проблем (оценка экономического эффекта).
- Какие преимущества в итоге из этого может извлечь компания-исполнитель проекта.

Соответствующий раздел в концепции проекта-примера будет иметь следующий вид.

12. Обоснование полезности проекта

12.1. Для Заказчика:

12.1.1. Повышение производительности обработки заказов в 2 раза.

12.1.1.1. “As Is”: 2500 заказов/год по 8 чел.*час.

12.1.1.2. “To Be”: 2500 заказов/год по 4 чел.*час.

12.1.1.3. Экономия: 2500 * 4 * $50 = $500 000 в год.

12.1.2. Повышение оперативности контроля

12.1.2.1. “As Is”: Ежемесячная отчетность.

12.1.2.2. “To Be”: Отчетность on-line.

12.1.3. Повышение удовлетворенности клиентов:

12.1.3.1. Сокращение срока обработки заказа в 2 раза.

12.1.3.2. Сокращение времени на поиск необходимой документации в 10 раз

12.1.3.3. Повышение оперативности обновления каталога 10 раз.

12.2. Для компании-исполнителя:

12.2.1. Высокая стратегическая ценность. Дает устойчивое увеличение рынка и завоевание нового рынка.

12.2.2. Финансовая ценность выше среднего. Ожидаемые доходы от проекта не менее чем в 1.3 раза превышают расходы.

Выводы

Эффективные процессы инициации программного проекта во многом определяют его будущую успешность. Недостаточное внимание этой фазе проекта неизбежно приводит к существенным проблемам при планировании, реализации и завершении.

Концепция проекта это ключевой документ, который используется для принятия решений в ходе всего проекта, а также на фазе приемки - для подтверждения результата.

Приоритет проекта определяется на основе оценки трех показателей:

- Финансовая ценность.
- Стратегическая ценность.
- Уровень рисков.

Дополнительная литература и источники

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
2. С. Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
3. Брукс Фредерик, «Мифический человеко-месяц, или Как создаются программные комплексы», Пер. с англ., СПб., Символ-Плюс, 1999.

Лекция 4. Планирование проекта

Уточнение содержания и состава работ

«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. Согласно [1]:

Иерархическая структура работ (ИСР) (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 ИСР служит концепция проекта, которая определяет продукты проекта и их основные характеристики. ИСР 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 ИСР составлена корректно, то любая работа, которая в нее не вошла не может считаться работой по проекту.

ИСР делит проект на подпроекты, пакеты работ, подпакеты. Каждый следующий уровень декомпозиции обеспечивает последовательную детализацию содержания проекта, что позволяет производить оценку сроков и объемов работ. ИСР должна включать все промежуточные и конечные продукты.

Выполнять декомпозицию работ проекта можно по-разному. Например, ГОСТ 19.102-77 предусматривает каскадный подход и определяет следующие стадии разработки программной системы:

1. Техническое задание
2. Эскизный проект
3. Технический проект
4. Рабочий проект
5. Внедрение

Если следовать этому стандарту, то на первом уровне ИСР должны находиться именно эти проектные продукты. Если бы пришлось разрабатывать АСУ для управления ядерным реактором или пилотируемым космическим аппаратом, то именно так и следовало поступать. Однако в коммерческой разработке ПО такой подход не эффективен. Как мы уже говорили, современный процесс разработки коммерческого ПО должен быть *инкрементальным*. Это означает, что на верхнем уровне декомпозиции нашего проекта должны находиться продукты проекта, а на следующем уровне - компоненты, из которых эти

продукты состоят. Компоненты далее могут быть декомпозированы на «фичи» - функции, которые они должны реализовывать.

Выделение компонентов, составляющих программный продукт, это элемент высокоуровневого проектирования, которое мы должны выполнить на фазе планирования проекта, не дожидаясь проработки всех функциональных требований к разрабатываемому ПО. Компонентами могут быть как прикладные подсистемы, так и инфраструктурные или ядерные, например, подсистема логирования, безопасности, библиотека визуальных компонентов GUI.

При составлении базового плана работ не стоит стремиться максимально детализировать все работы. ИСР не должна содержать слишком много уровней, достаточно 3-5. Например, ИСР нашего проекта-примера разработки «Автоматизированной системы продажи документации» может выглядеть следующим образом (Рисунок 17).

1. Проект разработки «Автоматизированной системы продажи документации»
    - 1.1. Подготовка технического задания на автоматизацию
        - 1.1.1.1. Проведение аналитического обследования
        - 1.1.1.2. Разработка функциональных требований
        - 1.1.1.3. Разработка требований базовому ПО
        - 1.1.1.4. Разработка требований к оборудованию и к операционно-системному ПО
        - 1.1.1.5. Согласование и утверждение ТЗ
        - *1.1.1.6. ТЗ утверждено*
    - 1.2. Поставка и монтаж оборудования
        - 1.2.1.Разработка спецификации на оборудование
        - 1.2.2.Закупка и поставка оборудования
        - 1.2.3.Монтаж оборудования
        - 1.2.4.Установка и настройка опреационно-системного ПО
        - *1.2.5.Монтаж оборудования завершен*
    - 1.3. Поставка и установка базового ПО
        - 1.3.1.Разработка спецификаций на базовое ПО
        - 1.3.2.Закупка базового ПО
        - 1.3.3.Развертывание и настройка базового ПО
        - *1.3.4.Базовое ПО установлено у заказчика*
    - 1.4. Разработка и тестирование прикладного ПО
        - 1.4.1.Разработка спецификаций на прикладное ПО
        - 1.4.2.Установка и конфигурирование рабочей среды
        - 1.4.3.Проектирование и разработка ПО
            - 1.4.3.1. Авторизация и аутентификация пользователей.
            - 1.4.3.2. Разработка подсистемы заказа документации
                - 1.4.3.2.1. Просмотр каталога продуктов.
                - 1.4.3.2.2. Поиск продуктов по каталогу.
                - 1.4.3.2.3. Заказ выбранных продуктов.
                - 1.4.3.2.4. Просмотр информации о статусе заказа.
                - 1.4.3.2.5. Информирование клиента об изменении статуса заказа.
                - *1.4.3.2.6. Подсистема заказа документации передана в тестовую эксплуатацию (на серверах разработчика).*
            - 1.4.3.3. Разработка подсистемы обработки заказов
                - 1.4.3.3.1. Просмотр и обработка заказов исполнителями из службы продаж.
                - 1.4.3.3.2. Просмотр статистики поступления и обработки заказов за период.
                - *1.4.3.3.3. Подсистема обработки заказов передана в тестовую эксплуатацию на оборудовании Заказчика*
            - 1.4.3.4. Разработка подсистемы сопровождения каталога
                - 1.4.3.4.1. Подготовка и сопровождение каталога продукции.
            - 1.4.3.5. Исправление ошибок

1.4.4. Тестирование ПО
1.4.4.1. Раунд 1
1.4.4.2. Раунд 2
1.4.4.3. Раунд 3
1.4.4.4. Выходное тестирование
1.4.5. Документирование прикладного ПО
1.5. Обучение пользователей
1.5.1. Подготовка учебных курсов
1.5.2. Обучение сотрудников Отдела 123
1.5.3. Обучение руководства ОАО XYZ
1.5.4. Обучение администраторов системы
1.6. Ввод в опытную эксплуатацию
1.6.1. Развертывание и настройка прикладного ПО
1.6.2. Проведение приемо-сдаточных испытаний
1.6.3. Акт передачи системы в опытную эксплуатацию утвержден
1.7. Сопровождение системы в период опытной эксплуатации
1.8. Система передана в промышленную эксплуатацию

Рисунок 17. Иерархическая структура работ проекта разработки «Автоматизированной системы продажи документации» (курсивом выделены контрольные точки проекта)

Должна быть установлена персональная ответственность за все части проекта (подпроекты и пакеты работ). Для каждого пакета работ должен быть четко определен результат на выходе. Работы и оценки проекта должны быть согласованы с ключевыми участниками команды, руководством компании-исполнителя и, при необходимости, с заказчиком. В результате согласования члены команды принимают на себя обязательства по реализации проекта, а руководство принимает на себя обязательства по обеспечению проекта необходимыми ресурсами.

ИСР является одним из основных инструментов (средств) 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.

Планирование управления содержанием

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 ПО.

Поэтому, сразу, как только удалось стабилизировать и согласовать ИСР, необходимо разработать план управления содержанием проекта. Для этого следует:

- Определить источники запросов на изменение.
- Установить порядок анализа, оценки и утверждения/отклонения изменения содержания.
- Определить порядок документирования изменений содержания.

- 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, наконец, менеджера проекта. Поэтому эта работа должна обязательно быть учтена в плане.

Планирование организационной структуры

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.

И еще. 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.

Планирование управления конфигурациям

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.

Планирование управления качеством

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 это важная работа, которая должна быть спланирована заранее и выполняться по ходу всего программного проекта, а не только во время приемо-сдаточных 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. И еще. 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:

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

Помимо перечисленных разделов план проекта должен включать еще:

- План управления рисками
- Оценку трудоемкости и сроков работ

Эти вопросы будут рассмотрены далее в специальных лекциях.

Базовое расписание 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*. 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»

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 ПО, которая должна предшествовать разработке прикладного ПО. В этом случае можно создавать «заглушки» эмулирующие работу базового ПО. 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 (Critical path)– 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.

Чтобы проиллюстрировать понятие критического пути рассмотрим пример «суперпроекта»[5]. Концепция проекта выглядит следующим образом.

Цель проекта. Сделать завтрак в постель

Результаты проекта. Завтрак в постели из вареного яйца, тоста и апельсинового сока.

Ресурсы. Имеется один оператор и обычное кухонное оборудование.

Сроки. Проект начинается на кухне в 8:00 и завершается в спальне.

Критерий приемки. Используются минимальные трудовые ресурсы и срок. Конечный продукт имеет высокое качество: яйцо свежесваренное, тост теплый, сок холодный.

Обоснование полезности. Проект служит достижению стратегических целей.

[5] Интернет-источник идеи, к сожалению, восстановить не удалось.

Иерархическая структура работ, ориентированная на конечный продукт, с оценкой их длительности представлена на Рисунок 18.

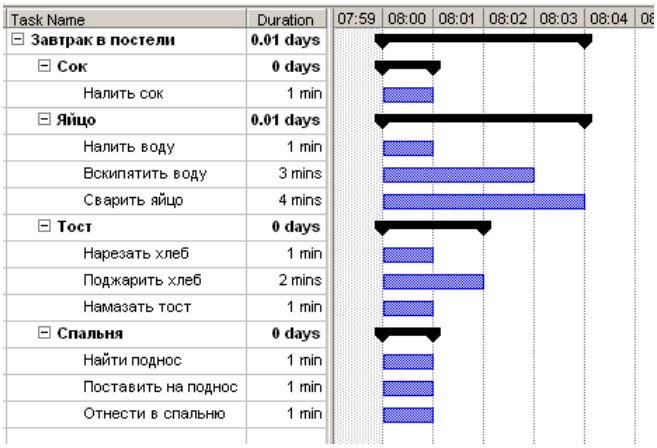

Рисунок 18. Иерархическая структура работ «суперпроекта»

На следующем шаги мы должны учесть зависимости между работами, например, нельзя жарить хлеб, пока мы его не нарезали.

С учетом зависимостей мы получим следующую диаграмму расписания нашего проекта (Рисунок 19)

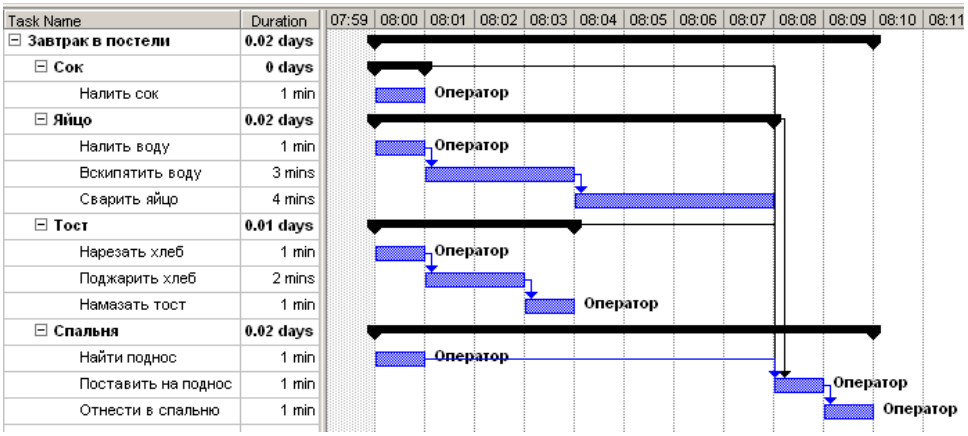

Рисунок 19. Диаграмма расписания «суперпроекта» с учетом зависимостей между работами.

В результате мы определили, что минимальный срок реализации нашего проекта составляет 10 минут. Однако мы не можем на этом остановиться, поскольку должны еще учесть ограничение по ресурсам. У нас только один оператор. Если мы посмотрим на диаграмму загруженности ресурсов (Рисунок 20), то увидим, что наш критический ресурс загружен на первой минуте на 400%. что недопустимо.

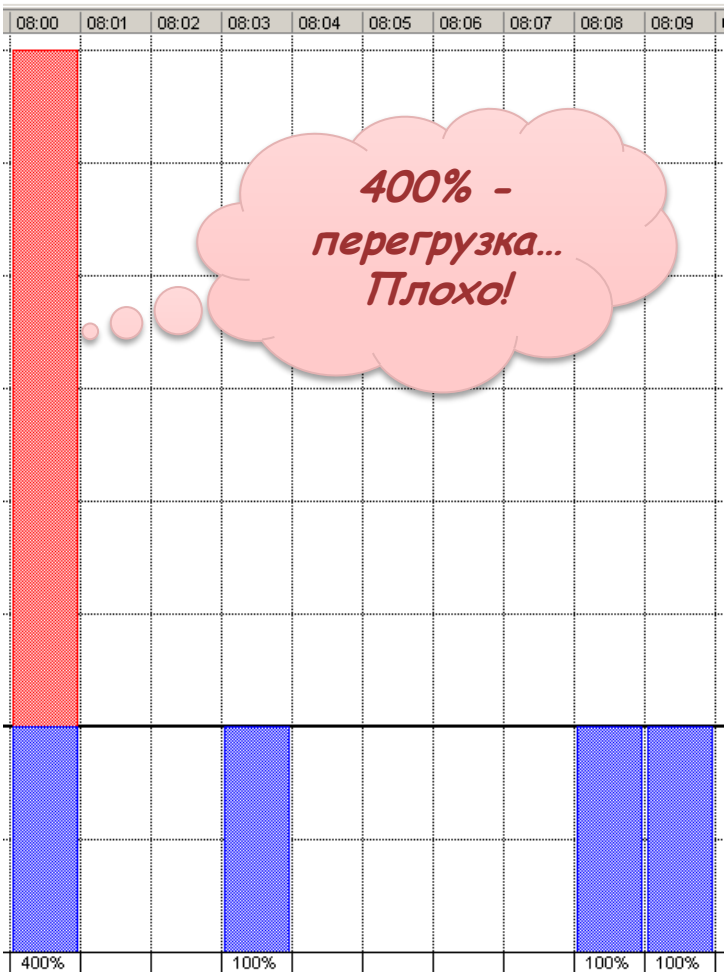

Рисунок 20. Диаграма загруженности ресурсов в «суперпроекте»

Следовательно, мы должны выполнить выравнивание ресурсов. Поскольку одним из критериев успеха проекта является его минимальная длительность, то если мы не хотим ее увеличивать, мы должны выявить критический путь в проекте (Рисунок 21) и не сдвигать работы, которые на нем находятся.

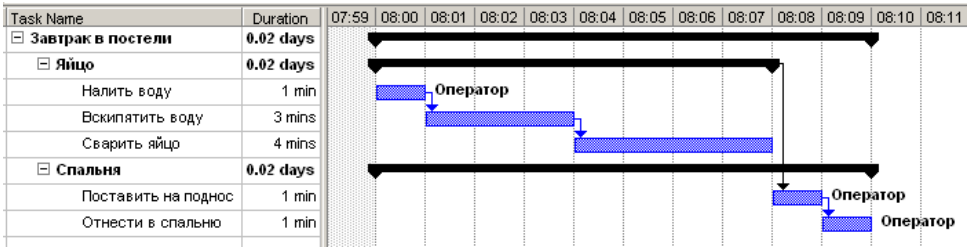

Рисунок 21. Критический путь в «суперпроекте»

Поэтому, после выравнивания ресурсов, расписание нашего проекта будет выглядеть следующим образом (Рисунок 22).

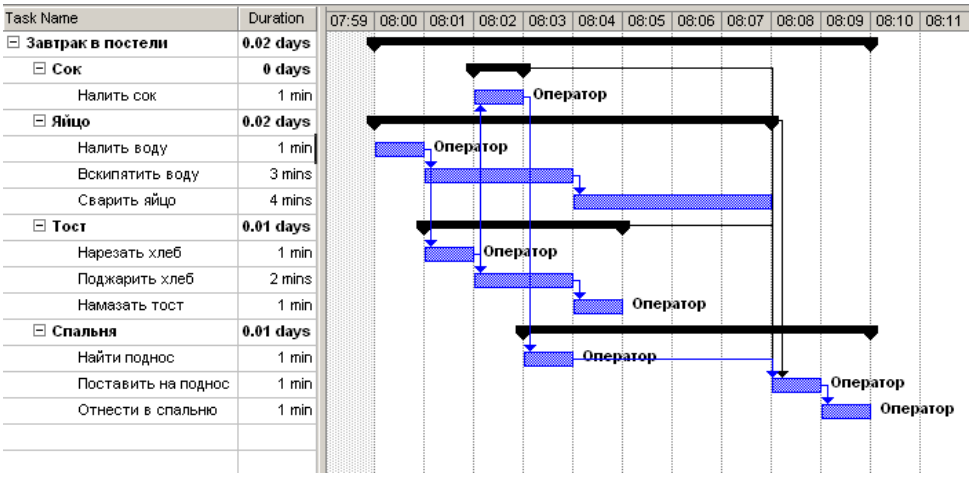

Рисунок 22. Расписание «суперпроекта» после выравнивания ресурсов

Теперь диаграмма загруженности ресурсов (Рисунок 23) выглядит приемлемо и у оператора даже появилось три минуты свободного времени на перекур. При этом общая длительность реализации проекта по-прежнему составляет 10 минут.

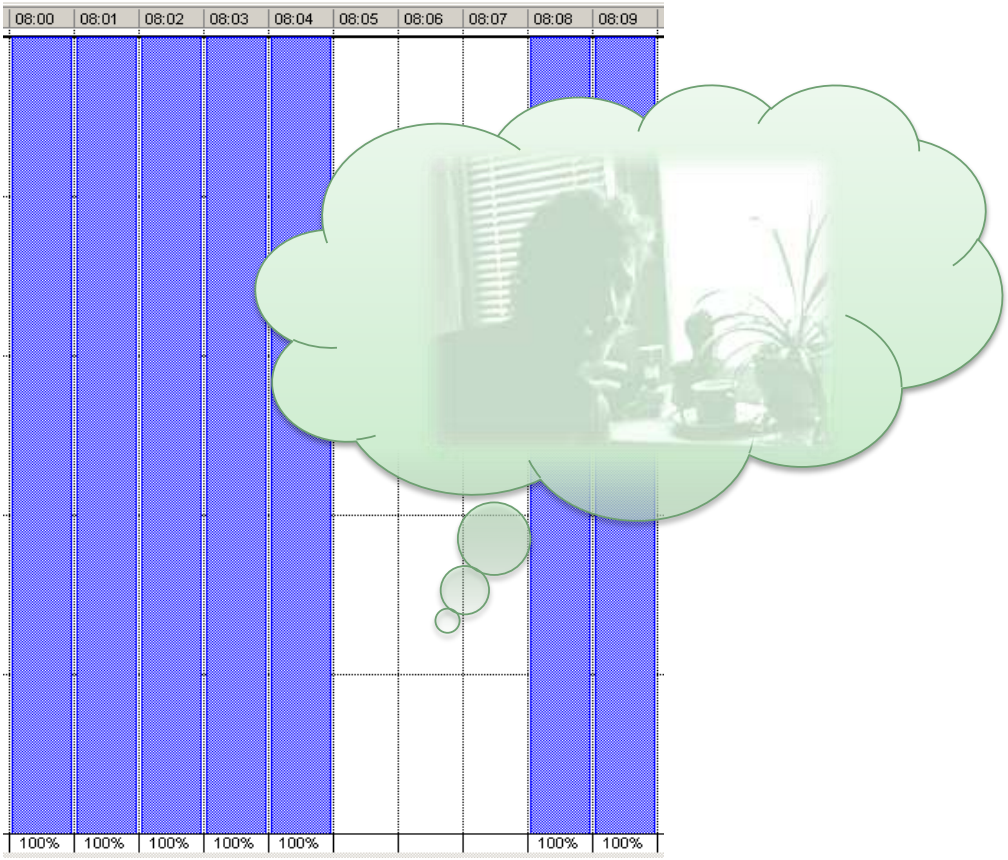

Рисунок 23. Диаграмма загруженности ресурсов после выравнивания

Выводы

На верхнем уровне ИСР должны находиться не процессы, а продукты проекта, на следующем уровне - компоненты из которых эти продукты состоят. Выделение компонентов, составляющих программный продукт, это элемент высокоуровневого проектирования, которое мы должны выполнить на фазе планирования проекта, не дожидаясь проработки всех функциональных требований к разрабатываемому ПО.

Помимо работ, непосредственно направленных на создание программного обеспечения, в плане проекта должны быть предусмотрены необходимые ресурсы для обеспечения работ по следующим процессам:

- управление содержанием;
- управление конфигурациями,
- управление качеством,
- управление рисками,
- управление проектом.

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.

Дополнительная литература и источники

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.

Лекция 5. Управление рисками проекта

Основные понятия

Том Демарко в своей книге [1] пишет: «Проект без риска – удел неудачников. Риски и выгода всегда ходят рука об руку». В первой лекции мы уже говорили о том, что, в силу специфики отрасли, программная инженерия остается и, в ближайшем будущем, будет оставаться производством с высоким уровнем рисков. Если задуматься, то все, что мы делаем, управляя проектом разработки ПО, направлено на борьбу с рисками не уложиться в срок, перерасходовать ресурсы, разработать не тот продукт, который требуется. Определение риска было дано в предыдущей лекции.

Риск это проблема, которая еще не возникла, а проблема – это риск, который материализовался. Риск характеризуется следующими характеристиками [2] (Рисунок 24):

- Причина или источник. Явление, обстоятельство обусловливающее наступление риска.
- Симптомы риска, указание на то, что событие риска произошло или вот-вот произойдет. Первопричина нам может быть не наблюдаема, например, заразились гриппом. Мы наблюдаем некоторые симптомы – поднялась температура.
- Последствия риска. Проблема или возможность, которая может реализоваться в проекте в результате произошедшего риска.
- Влияние риска. Влияние реализовавшегося риска на возможность достижения целей проекта. Воздействие обычно касается стоимости, графика и технических характеристик разрабатываемого продукта. Многие риски происходят частично и оказывают соразмерное отрицательное или положительное воздействие на проект.

Риск это всегда вероятность и последствия. Например, всегда есть вероятность того, что метеорит упадет на офис центра программных разработок, и это будет иметь катастрофические последствия для проекта. Однако вероятность наступления этого события настолько мала, что мы в большинстве проектов принимаем это риск и не пытаемся им управлять.

Майк Ньюэлл, вице-президент компании PSM Consulting, рассказывал, как он объясняет аудитории на своих лекциях, что такое риск. Он предлагает сыграть в кости на таких условиях, если на кубике выпадает шестерка, то выигрывает он. Если - любое другое число, то выигрывает слушатель. Ставка по 1 доллару. Обычно, большая часть аудитории соглашается сыграть на таких условиях. Майк поднимает ставки: $10, $100, $1000. Постепенно количество желающих поиграть становится все меньше и меньше. При ставке $1000, как правило, желающих рисковать не остается.

Принято [3] выделять две категории рисков:

- «Известные неизвестные». Это те риски, которые можно идентифицировать и подвергнуть анализу. В отношении таких рисков можно спланировать ответные действия.

- «Неизвестные неизвестные». Риски, которые невозможно идентифицировать и, следовательно, спланировать ответные действия.

Рисунок 24. Пример характеристик риска

Неизвестные риски это непредвиденные обстоятельства. Единственное, что мы можем в этом случае предпринять, это создать управленческий резерв бюджета проекта на случай незапланированных, но потенциально возможных изменений. На расходование этого резерва менеджер проекта, как правило, обязан получать одобрение вышестоящего руководства. Управленческие резервы на непредвиденные обстоятельства не входят в базовый план по стоимости проекта, но включаются в бюджет проекта. Они не распределяются по проекту, как бюджет, и поэтому не учитываются при расчете освоенного объема.

Девиз разработчиков ПО из Microsoft [2]:«Мы не боремся с рисками — мы ими управляем». Цели управления рисками проекта – снижение вероятности возникновения и/или значимости воздействия неблагоприятных для проекта событий. Адекватное управление рисками в компании – признак зрелости производственных процессов. Том Демарко пишет [1]: «Рассматривать только благоприятные сценарии и встраивать их в план проекта – настоящее ребячество. И все же мы постоянно так поступаем. …Если тех, кто говорит о возможных проблемах до открытия проекта, называют troublemakers, а тех, кто сдает проект спустя 2 месяца после обещанного срока, работая при этом по 60-80 часов в неделю, – героями, то у вас плохая команда».

Отказываться от управления проектными рисками это все равно, что в кинотеатре не иметь огнетушителей и плана эвакуации на случай пожара.

Планирование управления рисками

Управление рисками это определенная деятельность, которая выполняется в проекте от его начала до завершения. Как и любая другая работа в проекте управление рисками требует времени и затрат ресурсов. Поэтому эта работа обязательно должна планироваться. Планирование управления рисками – это процесс определения подходов и планирования операций по управлению

рисками проекта. Тщательное и подробное планирование управления рисками позволяет:

- выделить достаточное количество времени и ресурсов для выполнения операций по управлению рисками,
- определить общие основания для оценки рисков,
- повысить вероятность успешного достижения результатов проекта.

Планирование управления рисками должен быть завершено на ранней стадии планирования проекта, поскольку оно крайне важно для успешного выполнения других процессов.

В соответствие с [3] исходными данными для планирования управления рисками служат:

- Отношение к риску и толерантность к риску организаций и лиц, участвующих в проекте, оказывает влияние на план управления проектом. Оно должно быть зафиксировано в изложении основных принципов и подходов к управлению рисками.
- Стандарты организации. Организации могут иметь заранее разработанные подходы к управлению рисками, например категории рисков, общие определение понятий и терминов, стандартные шаблоны, схемы распределения ролей и ответственности, а также определенные уровни полномочий для принятия решений.
- Описание содержания проекта подробно описывает результаты поставки проекта и работы, необходимые для создания этих результатов поставки.
- План управления проектом, формальный документ, в котором указано, как будет исполняться проект и как будет происходить мониторинг и управление проектом.

План управления рисками обычно включает в себя следующие элементы:

- Определение подходов, инструментов и источников данных, которые могут использоваться для управления рисками в данном проекте.
- Распределение ролей и ответственности. Список позиций выполнения, поддержки и управления рисками для каждого вида операций, включенных в план управления рисками, назначение сотрудников на эти позиции и разъяснение их ответственности.
- Выделение ресурсов и оценка стоимости мероприятий, необходимых для управления рисками. Эти данные включаются в базовый план по стоимости проекта.
- Определение сроков и частоты выполнения процесса управления рисками на протяжении всего жизненного цикла проекта, а также определение операций по управлению рисками, которые необходимо включить в расписание проекта.
- Категории рисков. Структура, на основании которой производится систематическая и всесторонняя идентификация рисков с нужной

степенью детализации. Такую структуру можно разработать с помощью составления иерархической структуры рисков (Рисунок 25).

- Общие подходы для определения уровней вероятности, шкалы воздействия и близости рисков на проект.

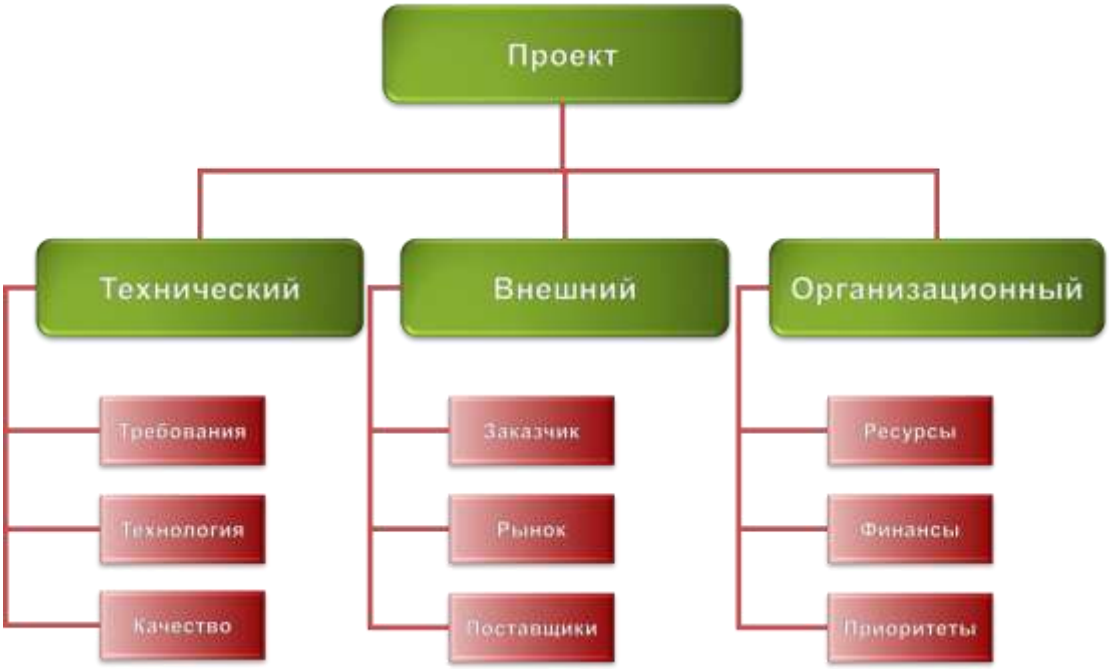

Рисунок 25. Пример иерархической структуры рисков проекта

Шкала оценки воздействия отражает значимость риска (Таблица 2) в случае его возникновения. Шкала оценки воздействия может различаться в зависимости от потенциально затронутой риском цели, типа и размера проекта, принятыми в организации стратегиями и его финансовым состоянием, а также от чувствительности организации к конкретному виду воздействий.

Вес	Значение	Критерий
3	Катастрофические	Потери более $100K
2	Критичные	Потери от $10K до $100K
1	Умеренные	Потери менее $10K

Таблица 2. Пример шкалы оценки воздействия рисков

Хотя риск может воздействовать и на сроки проекта, и на качество получаемого продукта, но все эти отклонения могут быть оценены в денежном эквиваленте. Например, последствия задержка по срокам для заказной разработки может быть выражена в сумме денежных санкций, определенных в контракте.

Похожая шкала может быть применена для оценки вероятности наступления риска (Таблица 3).

Вес	Значение	Критерий
3	Очень вероятно	Шансы наступления весьма велики
2	Возможно	Шансы равны
1	Мало вероятно	Наступление события весьма сомнительно

Таблица 3. Пример шкалы оценки вероятности осуществления риска

Еще одной важной характеристикой риска является близость его наступления. Естественно, что при прочих равных условиях рискам, которые могут осуществиться уже завтра, следует сегодня уделять больше внимания, чем тем, которые могут произойти не ранее, чем через полгода. Для шкалы оценки близости риска может быть применена, например, следующая градация: очень скоро, не очень скоро, очень нескоро.

Идентификация рисков

Идентификация рисков – это выявление рисков, способных повлиять на проект, и документальное оформление их характеристик. Это итеративный процесс, который периодически повторяется на всем протяжении проекта, поскольку в рамках его жизненного цикла могут обнаруживаться новые риски.

Исходные данные для выявления и описания характеристик рисков могут браться из разных источников.

В первую очередь это база знаний организации. Информация о выполнении прежних проектов может быть доступна в архивах предыдущих проектов. Следует помнить, что проблемы завершенных и выполняемых проектов, это, как правило, риски в новых проектах.

Другим источником данных о рисках проекта может служить разнообразная информация из открытых источников, научных работ, маркетинговая аналитика и другие исследовательские работы в данной области. Наконец, многие форумы по программированию могут дать бесценную информацию о возникших ранее проблемах в похожих проектах.

Каждый проект задумывается и разрабатывается на основании ряда гипотез, сценариев и допущений. Как правило, в описании содержания проекта перечисляются принятые допущения - факторы, которые для целей планирования считаются верными, реальными или определенными без привлечения доказательств. Неопределенность в допущениях проекта следует также обязательно рассматривать в качестве потенциального источника возникновения рисков проекта. Анализ допущения позволяет идентифицировать риски проекта, происходящие от неточности, несовместимости или неполноты допущений.

Для сбора информации о рисках могут применяться различные подходы. Среди этих подходов наиболее распространены:

- Опрос экспертов

- Мозговой штурм
- Метод Дельфи
- Карточки Кроуфорда

Цель опроса экспертов – идентифицировать и оценить риски путем интервью подходящих квалифицированных специалистов. Специалисты высказывают своё мнение о рисках и дают им оценку, исходя из своих знаний, опыта и имеющейся информации. Этот метод может помочь избежать повторного наступления на одни и те же грабли.

Перед опросом эксперт должен получить всю необходимую вводную информацию. Деятельность экспертов необходимо направлять, задавая вопросы. Во время опроса вся информация, выдаваемая экспертом, должна записываться и сохраняться. При работе с несколькими экспертами выходная информация обобщается и доводится до сведения всех задействованных экспертов.

К участию в мозговом штурме привлекаются квалифицированные специалисты, которым дают «домашнее задание» - подготовить свои суждения по определенной категории рисков. Затем проводится общее собрание, на котором специалисты по очереди высказывают свои мнения о рисках. Важно: споры и замечания 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.

Метод Дельфи во многом похож на метод мозгового штурма. Однако есть важные отличия. Во-первых, при применении этого метода эксперты участвуют в опросе анонимно. Поэтому результат характеризуется меньшей субъективностью, меньшей предвзятостью и меньшим влиянием отдельных экспертов. Во-вторых, опрос экспертов проводится в несколько этапов. На каждом этапе модератор рассылает анкеты, собирает и обрабатывает ответы. Результаты опроса рассылаются экспертам снова для уточнения их мнений и оценок. Такой подход позволяет достичь некоего общего мнения специалистов о рисках.

Для быстрого выявления рисков можно воспользоваться еще одной из методик социометрии является известной как "Карточки Кроуфорда" [5] .

Суть этой методики в следующем. Собирается группа экспертов 7-10 человек. Каждому участнику мини-исследования раздается по десять карточек (для этого вполне подойдет обычная бумага для записок). Ведущий 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 должно быть. Ведущий делает небольшую 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 подобрана хорошо (в том смысле, что в нее входят люди с различными точками зрения), 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.

В качестве источника информации при выявлении рисков могут служить различные доступные контрольные списки рисков проектов разработки ПО,

которые следует проанализировать на применимость к данному конкретному проекту.

Например, Барии Боэм [6] приводит список 10 наиболее распространенных рисков программного проекта:

1. Дефицит специалистов.
2. Нереалистичные сроки и бюджет.
3. Реализация несоответствующей функциональности.
4. Разработка неправильного пользовательского интерфейса.
5. “Золотая сервировка”, перфекционизм, ненужная оптимизация и оттачивание деталей.
6. Непрекращающийся поток изменений.
7. Нехватка информации о внешних компонентах, определяющих окружение системы или вовлеченных в интеграцию.
8. Недостатки в работах, выполняемых внешними (по отношению к проекту) ресурсами.
9. Недостаточная производительность получаемой системы.
10. “Разрыв” в квалификации специалистов разных областей знаний.

Демарко и Листер [1] приводят свой список из пяти наиболее важных источников рисков любого проекта разработки ПО:

1. Изъяны календарного планирования
2. Текучесть кадров
3. Раздувание требований
4. Нарушение спецификаций
5. Низкая производительность

Не существует исчерпывающих контрольных списков рисков программного проекта, поэтому необходимо внимательно анализировать особенности каждого конкретного проекта.

Результатом идентификации рисков должен стать список рисков с описанием их основных характеристик: причины, условия, последствий и ущерба.

Если вернуться к примеру проекта создания «Автоматизированной системы продажи документации», который мы рассматривали в предыдущих лекциях, то список главных выявленных рисков может выглядеть следующим образом:

Таблица 4 Список рисков проекта создания «Автоматизированной системы продажи документации»

Причина	Условия	Последствия	Ущерб
Требования не ясны.	Отсутствие описания сценариев использования системы.	Задержка начала разработки прикладного ПО. Большой объем переработок.	Задержки в сроках сдачи готового продукта и дополнительные трудозатраты.
Недостаток квалифицированных кадров.	Архитектура и код низкого качества.	Большое число ошибок. Большие затраты на их исправление.	Задержки в сроках сдачи готового продукта и дополнительные трудозатраты.
Текучесть кадров.	Частая смена участников команды.	Низкая производительность при вводе новых участников в проект.	Задержки в сроках сдачи готового продукта и дополнительные трудозатраты.

За процессом идентификации рисков следует процесс их качественного анализа.

Качественный анализ рисков

Качественный анализ рисков включает в себя расстановку рангов для идентифицированных рисков. При анализе вероятности и влияния предполагается, что никаких мер по предупреждению рисков не производится.

Качественный анализ рисков включает:

- Определение вероятности реализации рисков.
- Определение тяжести последствий реализации рисков.
- Определения ранга риска по матрице «вероятность - последствия».
- Определение близости наступления риска.
- Оценка качества использованной информации.

Для качественной оценки вероятности реализации риска и определения тяжести последствий его реализации применяется, как правило, общепринятые в организации шкалы, примеры которых мы приводили ранее.

Для определения ранга риска используется матрица вероятностей и последствий (Рисунок 26). Ранг риска определяется произведением веса вероятности и значимости последствий.

Могут, конечно, существовать и более сложные шкал для оценок вероятностей, значимости последствий и ранга рисков. Встречались шкалы, которые

содержали до 10 градаций. Но, на мой взгляд, наиболее прагматичный подход – это использовать трехуровневое ранжирование.

Продолжая рассмотрение примера проекта создания «Автоматизированной системы продажи документации», матрица рангов главных выявленных рисков может выглядеть следующим образом (Таблица 5).

Таблица 5. Матрица рангов главных выявленных рисков проекта создания «Автоматизированной системы продажи документации»

Причина	Вероятность	Воздействие	Ранг
Требования не ясны	Очень вероятно	Катастрофические	9
Недостаток квалифицированных кадров.	Очень вероятно	Критичные	6
Текучесть кадров.	Возможно	Критичные	4

Рисунок 26 Ранг риска и матрица вероятностей и последствий

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:

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

Результатом качественного анализа рисков является их подробное описание (Таблица 6).

Таблица 6. Пример карточки с описанием риска

Номер: R-101	**Категория**: Технологический.
Причина: Недостаток квалифицированных кадров.	**Симптомы**: Разработчики будут использовать новую платформу - J2EE.
Последствия: Низкая производительность разработки	**Воздействие**: Увеличение сроков и трудоемкости разработки.
Вероятность: Очень вероятно.	**Степень воздействия**: Критичная.
Близость: Очень скоро.	**Ранг**: 6.
Исходные данные: «Содержание проекта», «План обеспечения ресурсами», Протоколы совещаний №21 от 01.06.2008, №27 от 25.06.2008.	

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

Количественный анализ рисков

Количественный анализ производится в отношении тех рисков, которые в процессе качественного анализа были квалифицированы как имеющие высокий и средний ранг.

Для количественного анализа рисков могут быть использованы следующие методы:

- Анализ чувствительности.
- Анализ дерева решений.
- 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». Рисунок 27 представляет пример такой диаграммы, которая отражает влияние на проектные трудозатраты различных факторов профессионализма разработчиков ПО [7].

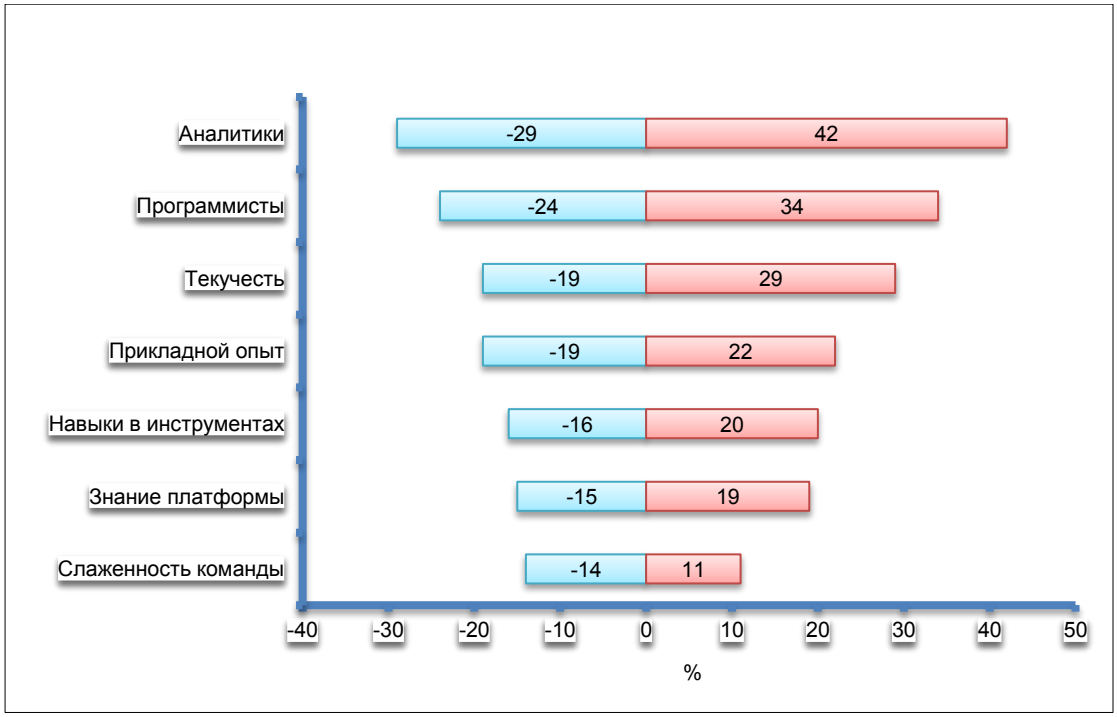

Рисунок 27. Влияние факторов профессионализма разработчиков ПО на трудозатраты по проекту.

Анализ последствий возможных решений проводится на основе изучения диаграммы дерева решений, которая описывает рассматриваемую ситуацию с учетом каждой из имеющихся возможностей выбора и возможного сценария. Рисунок 28 представляет пример диаграммы дерева решений на дугах которой проставлены вероятности и затраты при развитии событий по тому или иному сценарию. Критерием для принятия решения служит математическое ожидание потерь от его принятия.

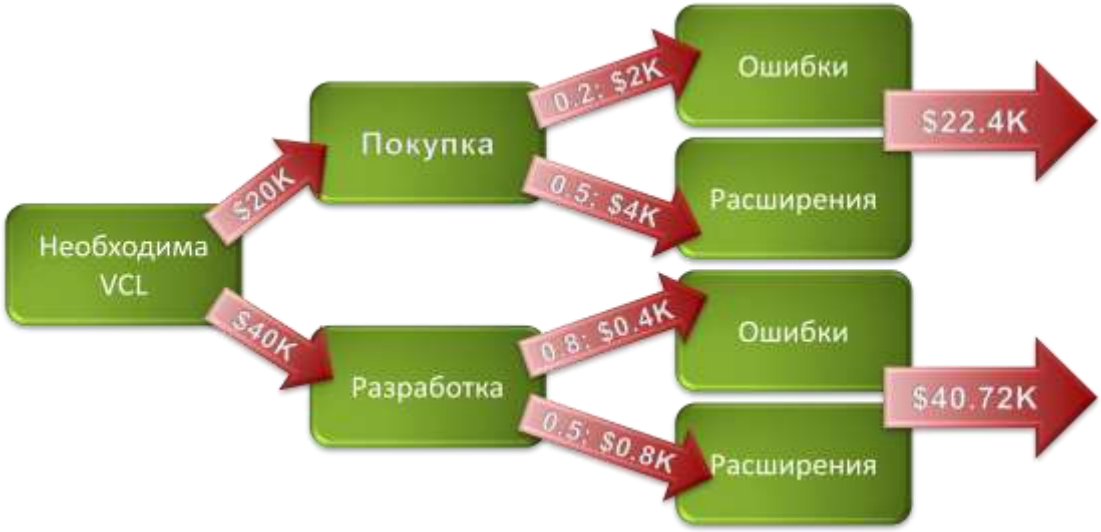

Рисунок 28. Пример анализ дерева решений при выборе покупать или производить необходимую для проекта библиотеку визуальных компонентов (VCL).

При моделировании рисков проекта используется модель для определения последствий от воздействия подробно описанных неопределенностей на результаты проекта в целом. Моделирование обычно проводится с помощью метода Монте-Карло.

Интересный пример подобной модели - система Riskology от Демарко и Листера, который иллюстрирует применение метода Монте-Карло для получения информации о том, какой запас времени будет необходим для того, чтобы преодолеть влияние всех неуправляемых рисков проекта, приведен в источнике [8]. Модель позволяет учесть пять основных (Рисунок 29) и пять дополнительных рисков проекта.

НАЗВАНИЕ РИСКА	ОПИСАНИЕ	СТАТУС
КАЛЕНДПЛАН	Изъяны календарного планирования	ВКЛ
ТЕКУЧКА	Текучесть кадров	ВКЛ
РАЗДУВАНИЕ	Раздувание требований	ВКЛ
СПЕЦИФИКАЦИИ	Нарушение спецификаций	ВКЛ
ПРОИЗВОД	Низкая производительность	ВКЛ

Рисунок 29. Пять основных факторов риска программного проекта, учитываемые в модели Riskology

Характеристики предопределенных в системе Riskology рисков пользователь может изменить, задав значения минимальной, максимальной и наиболее

вероятной задержки сроков сдачи проекта из-за влияния данного риска. Можно включить в модель дополнительные собственные риски. Результат моделирования по методу Монте-Карло будет представлен в виде гистограммы распределения срока завершения оцениваемого проекта (Рисунок 30).

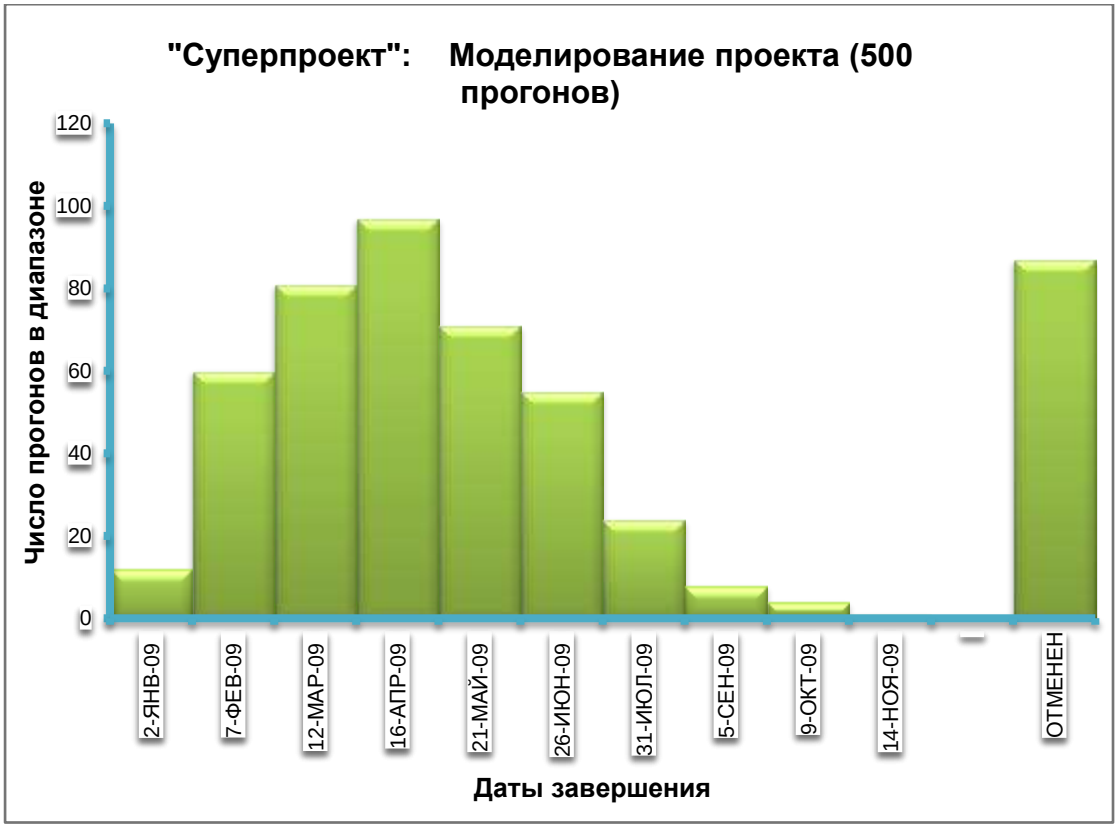

Рисунок 30. Гистограмма распределения возможного срока завершения проекта, рассчитанная по результатам моделирования методом Монте-Карло

На диаграмме также приведено количество случаев, примерно 80 из 500 прогонов, в которых проект, согласно результатам моделирования, был отменен до своего завершения.

Планирование реагирования на риски

Планирование реагирования на риски – это процесс разработки путей и определения действий по увеличению возможностей и снижению угроз для целей проекта. Данный процесс начинается после проведения качественного и количественного анализа рисков.

Запланированные операции по реагированию на риски должны соответствовать серьезности риска, быть экономически эффективными в решении проблемы, своевременными, реалистичными в контексте проекта и согласованными со всеми участниками.

Согласно [3] возможны четыре метода реагирования на риски:

- Уклонение от риска (risk avoidance).

- Передача риска (risk transference).
- Снижение рисков (risk mitigation).
- Принятие риска (risk acceptance).

Уклонение от риска предполагает изменение плана управления проектом таким образом, чтобы исключить угрозу, вызванную негативным риском, оградить цели проекта от последствий риска или ослабить цели, находящиеся под угрозой (например, уменьшить содержание проекта). Некоторые риски, возникающие на ранних стадиях проекта, можно избежать при помощи уточнения требований, получения дополнительной информации или проведения экспертизы. Например, уклониться от риска можно, если отказаться от реализации рискованного функционального требования или самостоятельно разработать необходимый программный компонент, вместо ожидания поставок продукта от субподрядчика.

Передача риска подразумевает переложение негативных последствий угрозы с ответственностью за реагирование на риск на третью сторону. Передача риска просто переносит ответственность за его управление другой стороне, но риск при этом никуда не девается. Передача риска практически всегда предполагает выплату премии за риск стороне, принимающей на себя риск. Например, заказ на стороне разработки рискованного компонента по фиксированной цене. В IT часто приходится формулировать риски в виде допущений, тем самым передавая его заказчику. Например, оценивая проект внедрения, мы можем записать допущение о том, что производитель не изменит стоимость лицензий на базовое ПО.

Снижение рисков предполагает понижение вероятности и/или последствий негативного рискованного события до приемлемых пределов. Принятие предупредительных мер по снижению вероятности наступления риска или его последствий часто оказываются более эффективными, нежели усилия по устранению негативных последствий, предпринимаемые после наступления события риска. Например, раннее разрешение архитектурных рисков снижает потери при досрочном закрытии проекта. Или регулярная ревизия поставок заказчиком может снизить вероятность риска его неудовлетворенности конечным результатом. Если в проектной команде высока вероятность увольнения сотрудников, то введение на начальной стадии в проект дополнительных (избыточных) людских ресурсов снижает потери при увольнении членов команды, поскольку не будет затрат на «въезд» в проектный контекст новых участников.

И, наконец, принятие риска означает, что команда проекта осознанно приняла решение не изменять план управления проектом в связи с риском или не нашла подходящей стратегии реагирования. Мы вынуждены принимать все «неизвестные риски».

Принятие это то, что всегда происходит, когда мы вообще не управляем рисками. Если же мы управляем рисками, то мы можем страховать риски, закладывая резерв в оценки срока завершения и/или трудозатрат. Проактивное отношение к принятым рискам может состоять в разработке план реагирования на риски. Этот план может быть введен в действие только при заранее определенных условиях, если есть уверенность и достаточное количество признаков того, что данный план будет успешно выполнен.

Важно помнить о вторичных рисках (Secondary Risks), возникающих в результате применения реагирования на риски, которые тоже должны быть

идентифицированы, проанализированы и при необходимости включены в список управляемых рисков.

Главные риски программных проектов и способы реагирования

Мой список из пяти главных причин провала программных проектов - следующий:

- Требования заказчика отсутствуют / не полны / подвержены частым изменениям.
- Отсутствие необходимых ресурсов и опыта.
- Отсутствие рабочего взаимодействия с заказчиком.
- Неполнота планирования. «Забытые работы».
- Ошибки в оценках трудоемкостей и сроков работ.

Это звучит банально, но сколько бы раз об этом не твердили ранее, по-прежнему, приходится сталкиваться с программными проектами, в которых отсутствуют какие-либо определенные цели и требования. Цитата из жизни: «Была бы разработана хорошая программа, а какой процесс автоматизировать с ее помощью, мы найдем». К этому можно добавить только одно: «Когда человек не знает, к какой пристани он держит путь, для него никакой ветер не будет попутным» (Сенека Луций Аней, философ, 65-3 до н.э.)

К часто упускаемым требованиям можно отнести:

- Функциональные
  - Программы установки, настройки, конфигурации.
  - Миграция данных.
  - Интерфейсы с внешними системами.
  - Справочная система.
- Общесистемные
  - Производительность.
  - Надежность.
  - Открытость.
  - Масштабируемость.
  - Безопасность.
  - Кросплатформенность.
  - Эргономичность.

Как правило, эти требования «всплывают» при подготовке и проведении приемо-сдаточных испытаний и могут сильно задержать проект по времени и

увеличить трудозатраты на его реализацию. Чтобы этого не происходило, следует достигать соглашения с заказчиком по всем перечисленным пунктам предпочтительнее еще на стадии инициации проекта. Например, если требования портируемости продукта на разные аппаратно-программные платформы нет, то это целесообразно включить в раздел концепции с допущениями проекта.

Если вероятность изменений требований проекта высока, то возможны следующие подходы для реагирования на данный риск:

- Переоценка проекта каждый раз, когда требования добавляются / изменяются (уклонение).
- Итерационная разработка. Контракт с компенсацией затрат на основе «Time & Materials» (передача риска Заказчику).
- Учет в оценках трудоемкости и сроков возможности роста требований, например, на 50% (резервирование риска).

И еще, при сборе требований следует соблюдать принцип минимализма Вольтера: «Рассказ закончен не тогда, когда в него нечего добавить, а тогда, когда из него нечего больше выкинуть». Для большинства программных продуктов применим принцип Парето: 80% ценности продукта заключены лишь в 20% требований к нему.

Если у нас в проекте недостаточно квалифицированных специалистов, то мы можем снизить последствия этого риска, применив следующие действия:

- Привлечь экспертов-консультантов на начальных этапах.
- Учитывать в оценках трудоемкости издержки на обучение сотрудников.
- Уменьшать потери от текучести кадров, привлекая на начальном этапе избыточное число участников.
- Учесть в оценках «время разгона» для новых сотрудников.

Для установления открытых и доверительных отношений с заказчиком, необходимо предпринимать следующие шаги:

- Постоянное взаимодействие.
- Согласование пользовательских интерфейсов и разработка прототипа продукта.
- Периодические поставки тестовых версий конечным пользователям для их оценки.

При планировании работ по проекту часто «забывают»:

- Обучение.
- Координация работ.
- Уточнение требований.
- Управление конфигурациями.

- Разработка и поддержка скриптов автосборки.
- Разработка автотестов.
- Создание тестовых данных.
- Обработка запросов на изменения.

И еще. Не стоит надеяться, что участники проекта будут каждую неделю по 40 часов работать именно над вашим проектом. Есть множество причин, по которым они не смогут работать по проекту 100% своего времени. К списку наиболее распространенных причин этого относятся:

- Сопровождение действующих систем.
- Повышение квалификации.
- Участие в подготовке технико-коммерческих предложений.
- Участие в презентациях.
- Административная работа.
- Отпуска, праздники, больничные.

Рекомендация, планировать, что разработчики, которые назначены в ваш проект на 100% будут реально работать над вашими задачами в среднем от 24 до 32 часов в неделю.

Ошибкам в оценках трудоемкости и сроков проекта и походам, которые позволяют их минимизировать, буде посвящена следующая лекция.

Управление проектом, направленное на снижение рисков

На стадии инициации проекта оценка его трудоемкости имеет погрешность от -50% до +100% [4]. Это, если оценка хорошая! А если плохая, то неопределенность, а, следовательно, и риски сорвать сроки и превысить плановую трудоемкость, могут быть в разы больше. Если не прилагать специальных усилий этот «дамоклов меч» неопределенности будет висеть над проектом на всем его протяжении (Рисунок 31).

Проектом следует управлять так, чтобы риски несвоевременной сдачи и перерасхода ресурсов постоянно снижались.

Ранее мы уже говорили о том, что 80% ценности разработки обусловлена лишь 20% требований к продукту, без реализации которых продукт для заказчика становится просто ненужным. Остальные требования, как правило, так называемые «украшательства», от части которых заказчик, как правило, может отказаться, чтобы получить проект в срок. Поэтому следует в первую очередь реализовывать ключевые функциональные требования.

Но есть и еще архитектурные риски. Известно, что закон Парето применим и к потреблению вычислительных ресурсов: 80% потребления ресурсов (время и память) приходится на 20% компонентов. Поэтому, необходимо реализовывать архитектурно-значимые требования так же в первую очередь, создавая «представительный» прототип будущей системы, который «простреливает» весь стек, применяемых технологий. Прототип позволит измерить и оценить

общесистемные свойства будущего продукта: доступность, быстродействие, надежность, масштабируемость и проч. (Рисунок 32)

Ошибка - реализовывать сначала легкие требования, чтобы продемонстрировать быстрый прогресс проекта.

Рисунок 31. Неопределенность не уменьшается, если управление не направлено на раннее разрешение рисков

Рисунок 32. Определение приоритетов требований на первые итерации проекта

Управление, нацеленное на снижение рисков, позволяет существенно снизить неопределенность на ранних стадиях проекта (Рисунок 33).

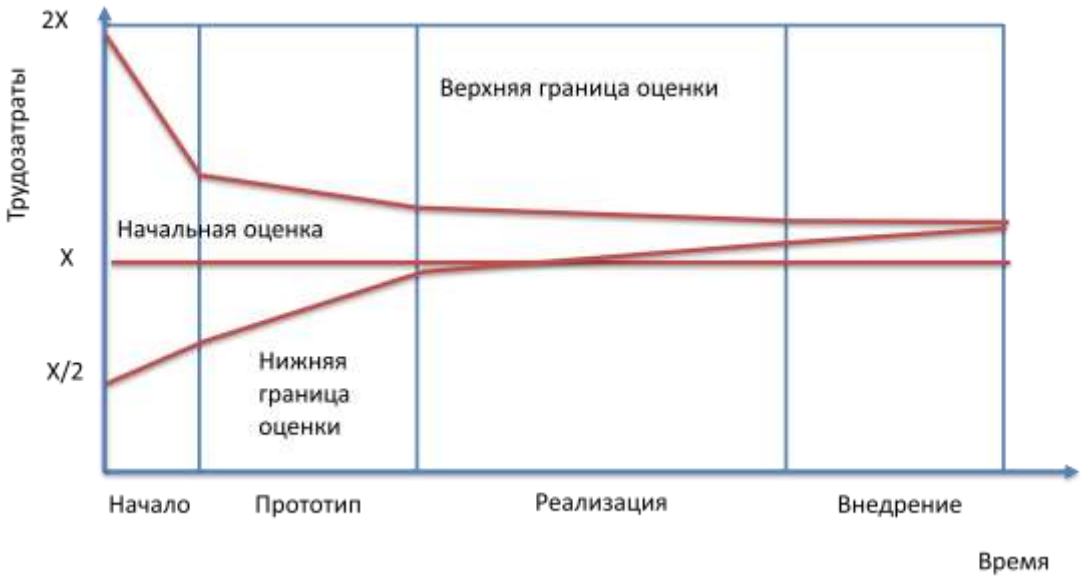

Рисунок 33. Управление, нацеленное на снижение рисков, позволяет уменьшать неопределенность

Проработка ключевых функциональных требований и детальное планирование их реализации позволяет уменьшить разброс начальных оценок, примерно, в 2 раза: от -30% до +50%. Детальное проектирование и разработка прототипа будущей системы позволит получить еще более точные оценки общей трудоемкости: от -10% до +15%.

Может оказаться так, что по результатам прототипирования, уточненные оценки суммарной трудоемкости окажутся неприемлемыми. В этом случае проект придется закрыть досрочно, но потери при этом, будут значительно меньше, чем в случае, если то же самое произойдет, когда проект уже в 2 раза превысит первоначальную оценку трудоемкости.

Если с заказчиком не удается найти взаимоприемлемое решение при первоначальной оценке проекта, то разумно попытаться договориться о выполнении проекта в 2 этапа с самостоятельным финансированием:

1. Исследование. Бизнес-анализ, уточнение требований, проектирование и прототипироание решения, уточнение суммарных оценок трудозатрат. Эта работа, как правило, требует 10 % общих трудозатрат и 20% времени всего проекта.

2. Непосредственно реализация. Если уточненные оценки трудозатрат окажутся приемлемыми для заказчика.

С вменяемыми заказчиками это часто удается.

Мониторинг и контроль рисков

Управление рисками должно осуществляться на протяжении всего проекта. Не вести мониторинг рисков в ходе проекта – все равно, что не следить за уровнем топлива при поездке на автомобиле.

Мониторинг и управление рисками – 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 управления рисками используются различные методики, например, анализ трендов и отклонений, для выполнения которых необходимы количественные данные об исполнении, собранные в процессе выполнения проекта.

Мониторинг и управление рисками включает в себя следующие задачи:

- Пересмотр рисков.
- Аудит рисков.
- 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. Неплохо начинать каждый статус митинг с вопроса: «Ну и какие еще неприятности нас ожидают?» 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 (см. Лекция 8. Реализация проекта). 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.

Выводы

Отказываться от управления проектными рисками это все равно, что в кинотеатре не иметь огнетушителей и плана эвакуации на случай пожара.

Все, что мы делаем, управляя проектом разработки ПО, должно быть направлено на борьбу с рисками не уложиться в срок, перерасходовать ресурсы, разработать не тот продукт, который требуется.

Цели управления рисками проекта – снижение вероятности возникновения и/или значимости воздействия неблагоприятных для проекта событий.

Главные причины провала программных проектов:

- Требования заказчика отсутствуют / не полны / подвержены частым изменениям.
- Отсутствие необходимых ресурсов и опыта.
- Отсутствие рабочего взаимодействия с заказчиком.

- Неполнота планирования. «Забытые работы».
- Ошибки в оценках трудоемкостей и сроков работ.

Дополнительная литература и источники

1. Том ДеМарко, Тимоти Листер, «Вальсируя с Медведями. Управление рисками в проектах по разработке программного обеспечения», М., Компания p.m.Office, 2005.
2. «Microsoft Solutions Framework. Дисциплина управления рисками MSF», вер. 1.1, 2002
3. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
4. С.Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
5. Ньюэл М.В., «Управление проектами для профессионалов. Руководство по подготовке к сдаче сертификационного экзамена PMP», КУДИЦ-Образ, 2006.
6. Barry W. Boehm. «A Spiral Model of Software Development and Enhancement», Computer, May 1988.
7. Barry Boehm, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice-Hall, 2000
8. www.systemsguild.com/riskology © 2005 Том ДеМарко, Тимоти Листер.

Лекция 6. Оценка трудоемкости и сроков разработки ПО

Оценка - вероятностное утверждение

Стив Макконнелл [1] пишет, что мы от природы склонны верить, что сложные формулы вида

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

всегда обеспечивают более точные результаты, чем простые формулы

Трудоемкость = КоличествоФакторов x СредниеЗатратыНаФактор

Однако, далеко не всегда это так. Сложные формулы, как правило, очень чувствительны к точности большого числа параметров (в приведенном примере формул COCOMO II содержится 21 параметр), которые надо задать, чтобы получить требуемые оценки.

Первое, что необходимо понимать при оценке проекта, это то, что любая оценка это всегда вероятностное утверждение. Если мы просто скажем, что трудоемкость данного пакета работ составляет **M** чел.*мес. (Рисунок 34), то это будет плохой оценкой потому, что одно единственное число ничего не скажет нам о вероятности того, что на реализацию этого пакета потребуется не более, чем **M** чел.*мес. Вряд ли мы можем считать себя «предсказамусами», которые точно знают что произойдет в будущем и сколько потребуется затрат на реализацию этого пакета работ.

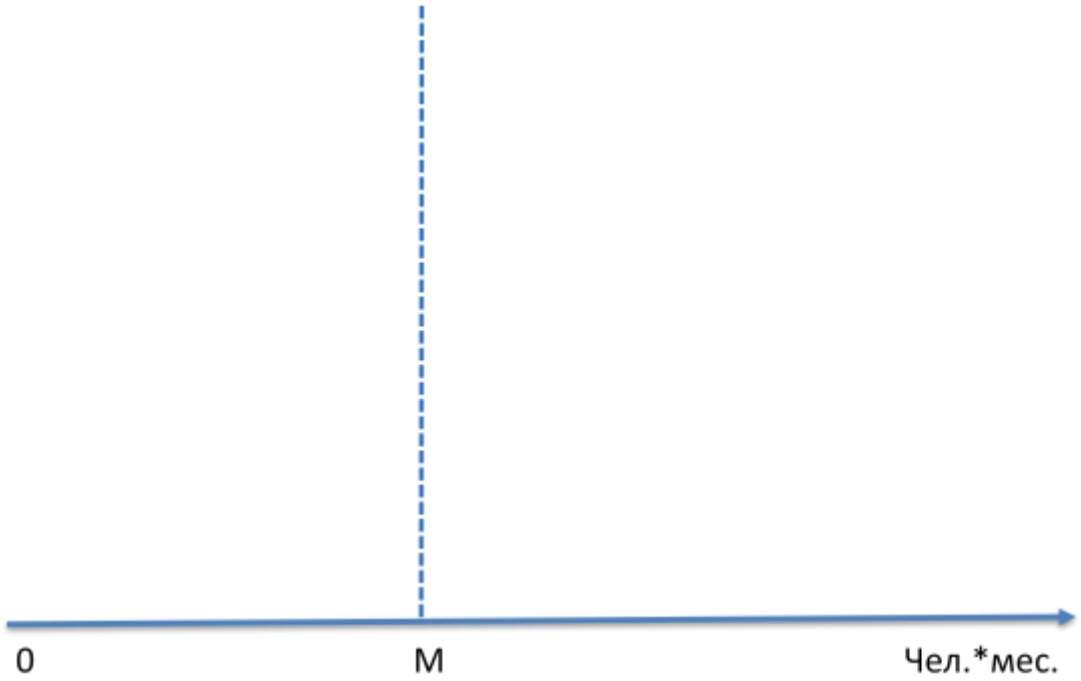

Рисунок 34. Точечная оценка трудоемкости пакета работ ничего не скажет нам о вероятности того, что на реализацию этого пакета потребуется не более, чем **M** чел.*мес.

Для того, чтобы понять, откуда берется неопределенность, рассмотрим простейший пример, попытаемся оценить трудоемкость добавления поля ввода телефонного номера клиента к уже существующей форме. Менеджер, наблюдающий работу программистов только со стороны, скажет, что эта работа потребует не больше 15 минут рабочего времени. Человек, умудренный программистским опытом, скажет, что эта работа может занять от 2 до 200 часов, и чтобы дать более точную оценку ему надо получить ответы на ряд вопросов:

- Может ли вводиться несколько номеров?
- Должна ли быть проверка номеров на действительность?
- Простая или сложная проверка?
- Если реализуем простую проверку, то не захочет ли клиент заменить ее на более сложную?
- Должна ли проверка работать для иностранных номеров?
- Можно ли воспользоваться готовым решением?
- Каково должно быть качество реализации? Вероятность ошибки после поставки?
- Сколько времени потребуется на реализацию и отладку? (зависит от конкретного исполнителя).

Называя такую «размытую» оценку опытный программист резервирует все риски разработки, связанные с перечисленными неопределенностями данного требования, которые он вынужден принимать на себя, не имея в данный момент необходимой уточняющей информации.

То, что наша оценка должна быть вероятностным утверждением, означает, что для нее существует некоторое распределение вероятности (Рисунок 35), которое может быть очень широким (высокая неопределенность) или достаточно узким (низкая неопределенность).

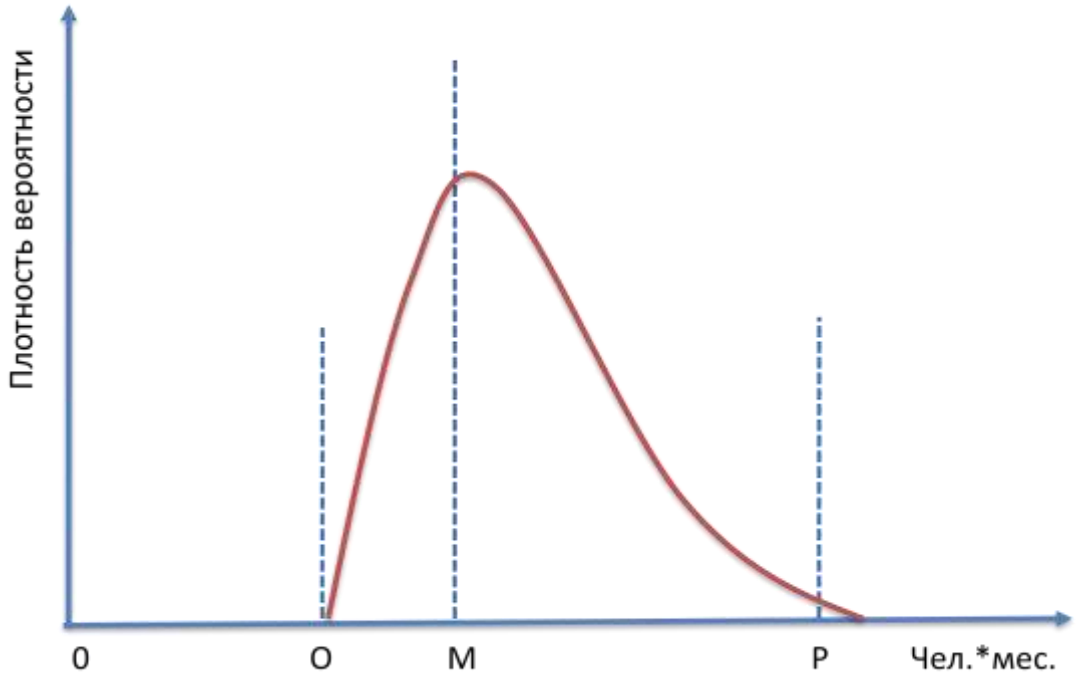

Рисунок 35. Оценка – всегда вероятностная величина

Если *M* – наиболее вероятное значение, то это не означает что это хорошая оценка, поскольку вероятность того, что фактическая трудоемкость превысит эту оценку, составляет более 50%.

Какая оценка может считаться хорошей? Стив Макконнелл утверждает [1]: «Хорошей считается оценка, которая обеспечивает достаточно ясное представление реального состояния проекта и позволяет руководителю проекта принимать хорошие решения относительно того, как управлять проектом для достижения целей».

Негативные последствия «агрессивного» расписания

В программостроении уже стало банальностью то, что разработчики без достаточного основания называют слишком оптимистичные сроки. Среди руководителей даже распространено неписаное правило: умножать на 2 оценку трудоемкости, которую сделал программист. Это пессимистичный подход. Реалисты умножают на $\pi = 3.14$.

Действительно, так иногда приходится поступать, если это программист, который только вчера отладил свою первую программу «Hello world!». Но если помочь молодым специалистам научится анализировать задачу, проектировать решение, составлять план работы, эффективно его реализовывать и анализировать полученные результаты, то можно будет не вспоминать, чему равно число π.

Еще один распространенный источник занижения сроков - необоснованные ожидания на применение новых технологий и средств разработки. Эти

ожидания, как правило, не оправдываются. Согласно статистике, приведенной Демарко, средняя производительность в программном производстве растет всего лишь на 3-5% в год.

Часто «агрессивное» расписание проекта появляется из-за того, что руководство и/или заказчик боятся переоценить проект, полагая, что согласно закону Паркинсона, работы по проекту займут все отведенное для него время. Следствием подобных опасений является, как правило, директивное занижение сроков реализации проекта.

Не реалистичность оценок один из серьезнейших демотивирующих факторов для участников. Недооценка приводит к ошибкам планирования и неэффективному взаимодействию. Например, было запланировано тестирование, а релиз еще не готов. Следствие - простой тестировщиков увеличение трудозатрат.

Если расписание излишне агрессивное, то с целью сэкономить время, недостаточно внимания уделяется анализу требований и проектированию. Исправление ошибок, допущенных на этих этапах, приведет к существенным дополнительным затратам.

Половина всех ошибок программирования возникают из-за стресса, вызванного излишним давлением фактора сроков. Ошибки исправляются наспех, обходными путями. В результате будет получен большой проблемный код и постоянно растущие затраты на исправление ошибок и внесение изменений. Позднее обнаружение ошибок приводит к тому, что затраты на их исправление увеличиваются в 50-100 раз.

Мне пришлось наблюдать проект, который вместо первоначально слишком оптимистично запланированных шести месяцев растянулся на три года. Хотя, если бы он был адекватно оценен, то он мог бы быть реализован за один год. Нереальные сроки, постоянное давление, сверхурочные, авралы приводят к тому, что затраты на проект растут экспоненциально и неограниченно.

Если участники проектной команды адекватно мотивированы на выполнение проектных работ с наименьшими затратами, то, на мой взгляд, этого достаточно, чтобы проект был реализован в минимально возможные сроки. О мотивации мы еще будем говорить (см. Лекция 7. Формирование команды).

Прагматичный подход. Метод PERT

Использование собственного опыта или опыта коллег, полученного в похожих проектах, это наиболее прагматичный подход, который позволяет получить достаточно реалистичные оценки трудоемкости и срока реализации программного проекта, быстро и без больших затрат.

Инженерный метод оценки трудоемкости проекта PERT (Program / Project Evaluation and Review Technique) был разработан в 1958 году в ходе проекта по созданию баллистических ракет морского базирования «Поларис». Входом для данного метода оценки служит список элементарных пакетов работ. Для инженерного подхода нет необходимости точно знать закон распределения нашей оценки трудоемкости каждого такого элементарного пакета. Диапазон неопределенности достаточно охарактеризовать тремя оценками:

Mi – наиболее вероятная оценка трудозатрат.

Oi – минимально возможные трудозатраты на реализацию пакета работ. Ни один риск не реализовался. Быстрее точно не сделаем. Вероятность такого, что мы уложимся в эти затраты, равна 0.

Pi – пессимистическая оценка трудозатрат. Все риски реализовались.

Оценку средней трудоемкости по каждому элементарному пакету можно определит по формуле:

$$Ei = (Pi + 4Mi + Oi)/6.$$

Для расчета среднеквадратичного отклонения используется формула:

$$CKOi = (Pi – Oi)/6.$$

Если наши оценки трудоемкости элементарных пакетов работ статистически независимы, а не испорчены, например, необоснованным оптимизмом то, согласно центральной предельной теореме теории вероятностей суммарная трудоемкость проекта может быть рассчитана по формуле:

$$E = \Sigma Ei$$

А среднеквадратичное отклонение для оценки суммарной трудоемкости будет составлять:

$$CKO = \sqrt{\sum CKO_i^2}$$

Тогда для оценки суммарной трудоемкости проекта, которую мы не превысим с вероятностью 95%, можно применить формулу:

$$E_{95\%} = E + 2 * CKO.$$

Это значит, что вероятность того, что проект превысит данную оценку трудоемкости составляет всего 5%. А это уже вполне приемлемая оценка, под которой может расписаться профессиональный менеджер.

Список элементарных пакетов работ, который используется при оценке трудоемкости, как правило, берется из нижнего уровня ИСР проекта. Но может быть использован и накопленный опыт аналогичных разработок. Проиллюстрирую данный подход на примере реального проекта. В Ассоциации CBOSS задачей проекта, который нам с коллегами посчастливилось реализовывать, была разработка на основе стандартов J2EE общесистемного ПО для перевода рабочих мест CBOSS на новую трехзвенную архитектуру. Был разработан набор стандартных компонентов и сервисов, из которых как из конструктора можно эффективно и качественно собирать прикладные подсистемы. Высокоуровневая архитектура реализовывала стандартный паттерн MVC (Рисунок 36), каждый из компонентов которого имел «точки расширения» для прикладной разработки, которые на рисунке выделены красным светом.

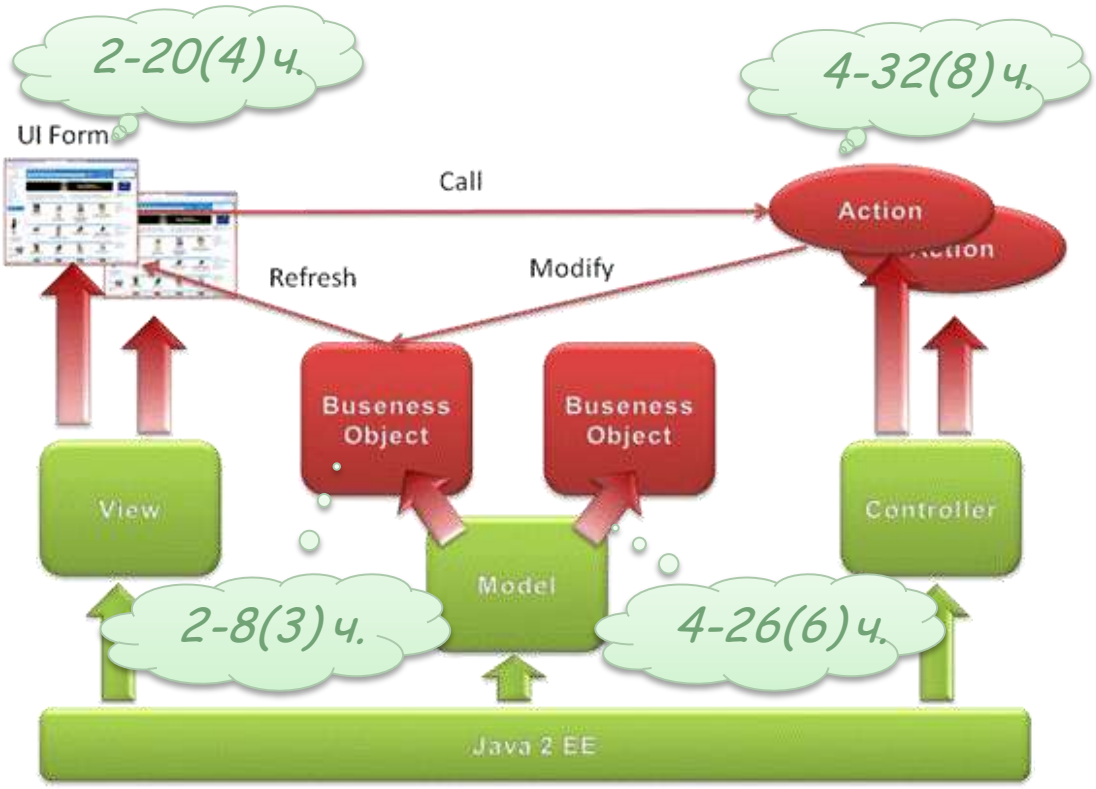

Рисунок 36. Высокоуровневая архитектура J2EE фреймворка для разработки приложений.

Такими точками расширения являлись:

- Пользовательский экран (UI Form), который собирался из готовых визуальных компонентов.
- Обработчики (Action), которые обрабатывали на сервере приложений события от активных визуальных компонентов, входящих в состав экрана.
- Объекты (Business Obj), которые моделировали прикладную область, и к которым обращались обработчики событий.

Так вот, хотя все разрабатываемые рабочие места различались по функциональности и сложности, накопленная статистика фактических трудозатрат на разработку прикладных систем позволяла нам оценивать проекты разработки нового приложения достаточно быстро и с высокой достоверностью.

Согласно этой статистике, разработка и отладка требовала у программиста:

- для одного экрана - от 2 до 20 часов (наиболее вероятно – 4 часа);
- для одного обработчика событий - от 4 до 32 часов (наиболее вероятно – 8 часов);
- для нового бизнес-объекта - от 2 до 8 часов (наиболее вероятно – 3 часа);
- для добавление нового бизнес-метода - от 2 до 26 часов (наиболее вероятно – 6 часов).

Весь проект прикладной разработки измерялся в «попугаях»:

- K_{UI} – количество пользовательских экранов.
- K_{Act} – количество обработчиков событий.
- K_{BO} – количество новых бизнес-объектов.
- K_{BM} – количество новых или модифицируемых бизнес-методов.

Если новое разрабатываемое приложение содержит 20 пользовательских экранов, 60 обработчиков событий, 16 новых бизнес-объекта и 40 новых бизнес-методов, которые необходимо добавить, как в новые, так и в уже существующие бизнес-объекты, тогда, согласно нашей статистике,

$E_{UI} = (2 + 4*4 + 20) / 6 = 6.7$ *чел.*час.*, $СКО_{UI} = (20 - 2) / 6 = 3$ *чел.*час*

$E_{Act} = (4 + 4*8 + 32) / 6 = 11.3$ *чел.*час.*, $СКО_{Atct} = (32 - 4) / 6 = 4.7$ *чел.*час*

$E_{BO} = (2 +4*3 + 8) / 6 = 3.7$ *чел.*час.*, $СКО_{BO} = (8 - 2) / 6 = 1$ *чел.*час*

$E_{BM} = (2 + 4*6 + 26) / 6 = 8.7$ *чел.*час.*, $СКО_{BM} = (26 - 2) / 6 = 4$ *чел.*час*

Для средней трудоемкости работ по кодированию в проекте может быть получена следующая оценка:

$$E = 20 * 6.7 + 60 * 11.3 + 16 * 3.7 + 40 * 8.7 \approx 1220 \text{ чел.*час.}$$

$$СКО = \sqrt{20*3^2 + 60*4.7^2 + 16*1^2 + 40*4^2} =$$
$$= \sqrt{180 + 1325 + 16 + 640} \approx \ 46 \ чел.*час.$$

Тогда для оценки суммарной трудоемкости проекта, которую мы не превысим с вероятностью 95%, получим

$$E_{95\%} = 1220 + 2 * 46 \approx 1300 \text{ чел.*час.}$$

Хотя относительная погрешность в оценке трудоемкости каждой такой элементарной работы составляла десятки процентов, для нашего проекта, в котором было таких «попугаев» было 136, относительная погрешность оценки суммарной трудоемкости, сделанной по методу PERT, составила, приблизительно, лишь 4%.

Даже если у нас очень размытые оценки трудоемкости каждой из элементарных работ, но они независимы, то ошибки мы делаем как в меньшую, так и большую стороны. Поэтому при фактической реализации проекта эти ошибки будут компенсироваться, что позволяет нам оценить общие трудозатраты по проекту существенно точнее, чем трудозатраты на каждую элементарную работу. Но это утверждение будет справедливо только в том случае, если наша ИСР содержит *все* необходимые работы, которые должны быть выполнены для получения *всех* продуктов проекта.

Полученную оценку трудоемкости кодирования необходимо умножить на четыре, поскольку помним (см. Лекция 3. Инициация проекта), что кодирование составляет только 25% общих трудозатрат проекта. Поэтому суммарная трудоемкость нашего проекта составит, приблизительно, 5200 чел.*час.

Как мы уже говорили ранее, если сотрудник на 100% назначен на проект, это, как правило, не означает, что он все 40 часов в неделю будет тратить на проектные работы. Тратить он будет 60 – 80% своего рабочего времени. Поэтому, в месяц сотрудник будет работать по проекту, примерно, 165 * 0.8 = 132 чел.*час/мес. Следовательно, трудоемкость проекта в человеко-месяцах составит, приблизительно 5200 / 132 ≈ 40.

Тогда согласно формуле Б.Боэма (Рисунок 15) оптимальная продолжительность проекта составит:

*T = 2.5 * (40) ^ 1/3 = 8.5 месяцев,*

а средняя численность команды – 5 человек.

Помним, что потребление ресурсов в проекте неравномерно (Рисунок 13), поэтому начинать проект должны 1-3 человека, а на стадии реализации начальная численность команды может быть увеличена в несколько раз.

Если же собственный опыт аналогичных проектов отсутствует, а коллеги-эксперты недоступны, то нам не остается ничего другого, как использовать формальные методики, основанные на обобщенном отраслевом опыте. Среди них наибольшее распространение получили два подхода:

- FPA IFPUG - метод функциональных точек,
- метод COCOMO II, Constructive Cost Model.

Обзор метода функциональных точек

Анализ функциональных точек - стандартный метод измерения размера программного продукта с точки зрения пользователей системы. Метод разработан Аланом Альбрехтом (Alan Albrecht) в середине 70-х. Метод был впервые опубликован в 1979 году. В 1986 году была сформирована Международная Ассоциация Пользователей Функциональных Точек (International Function Point User Group – IFPUG), которая опубликовала несколько ревизий метода [2].

Метод предназначен для оценки на основе логической модели объема программного продукта количеством функционала, востребованного заказчиком и поставляемого разработчиком. Несомненным достоинством метода является то, что измерения не зависят от технологической платформы, на которой будет разрабатываться продукт, и он обеспечивает единообразный подход к оценке всех проектов в компании.

При анализе методом функциональных точек надо выполнить следующую последовательность шагов (Рисунок 37):

1. Определение типа оценки.
2. Определение области оценки и границ продукта.
3. Подсчет функциональных точек, связанных с данными.
4. Подсчет функциональных точек, связанных с транзакциями.
5. Определение суммарного количества не выровненных функциональных точек (UFP).

6. Определение значения фактора выравнивания (FAV).

7. Расчет количества выровненных функциональных точек (AFP).

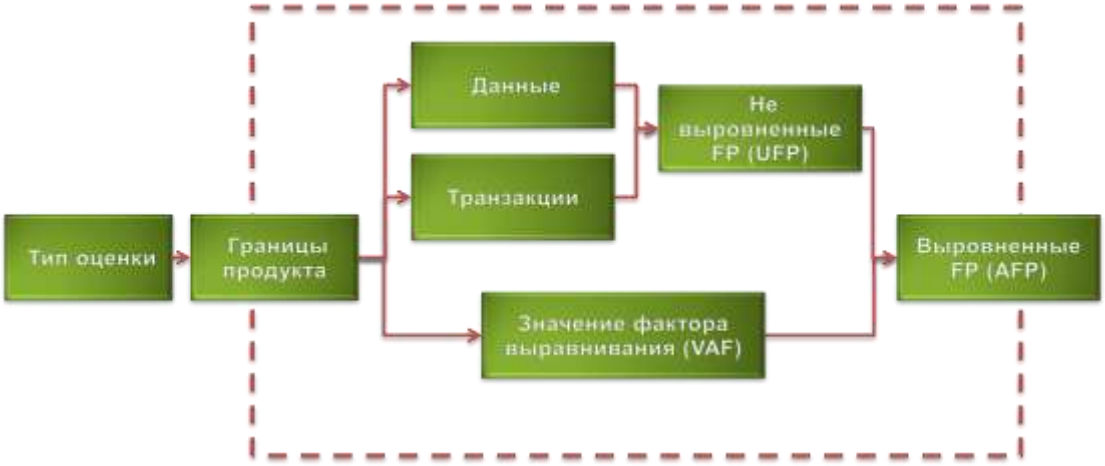

Рисунок 37. Процедура анализа по методу функциональных точек

Определение типа оценки

Первое, что необходимо сделать, это определить тип выполняемой оценки. Метод предусматривает оценки трех типов:

1. *Проект разработки.* Оценивается количество функциональности поставляемой пользователям в первом релизе продукта.

2. *Проект развития.* Оценивается в функциональных точках проект доработки: добавление, изменение и удаление функционала.

3. *Продукт.* Оценивается объем уже существующего и установленного продукта.

Определение области оценки и границ продукта

Второй шаг – это определение области оценки и границ продукта. В зависимости от типа область оценки может включать:

- Все разрабатываемые функции (для проекта разработки)

- Все добавляемые, изменяемые и удаляемые функции (для проектов поддержки)

- Только функции, реально используемые, или все функции (при оценке продукта и/или продуктов).

Третий шаг. Границы продукта (Рисунок 38) определяют:

- Что является «внешним» по отношению к оцениваемому продукту.

- Где располагается «граница системы», через которую проходят транзакции передаваемые или принимаемые продуктом, с точки зрения пользователя.

- Какие данные поддерживаются приложением, а какие – внешние.

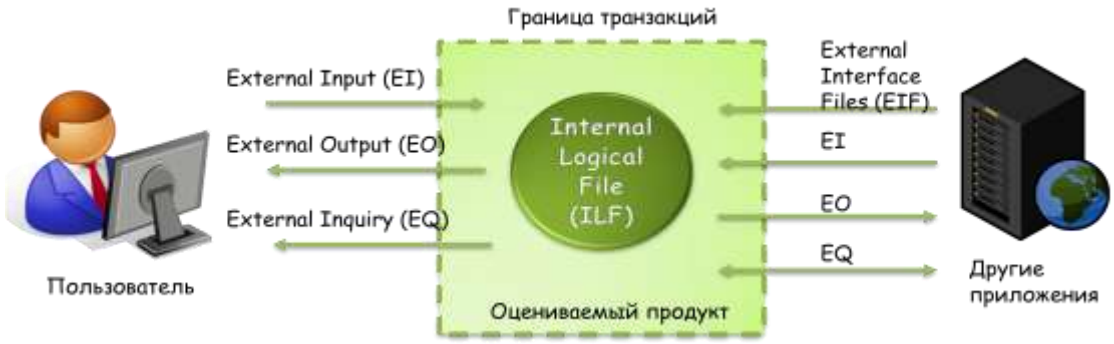

Рисунок 38. Границы продукта в методе функциональных точек

К логическим данным системы относятся:

- Внутренние логические файлы (ILFs) – выделяемые пользователем логически связанные группы данных или блоки управляющей информации, которые поддерживаются внутри продукта.

- Внешние интерфейсные файлы (EIFs) - выделяемые пользователем логически связанные группы данных или блоки управляющей информации, на которые ссылается продукт, но которые поддерживаются вне продукта.

Примером логических данных (информационных объектов) могут служить: клиент, счет, тарифный план, услуга.

Подсчет функциональных точек, связанных с данными

Третий шаг - подсчет функциональных точек, связанных с данными. Сначала определяется сложность данных по следующим показателям:

- DET (data element type) – неповторяемое уникальное поле данных, например, Имя Клиента – 1 DET; Адрес Клиента (индекс, страна, область, район, город, улица, дом, корпус, квартира) – 9 DET's

- RET (record element type) – логическая группа данных, например, адрес, паспорт, телефонный номер.

Оценка количества не выровненных функциональных точек, зависит от сложности данных, которая определяется на основании матрицы сложности (Таблица 7).

Таблица 7. Матрица сложности данных

	1-19 DET	**20-50 DET**	**50+ DET**
1 RET	Low	Low	Average
2-5 RET	Low	Average	High
6+ RET	Average	High	High

Оценка данных в не выровненных функциональных точках (UFP) подсчитывается по-разному для внутренних логических файлов (ILFs) и для внешних интерфейсных файлов (EIFs) (Таблица 8) в зависимости от их сложности.

Таблица 8. Оценка данных в не выровненных функциональных точках (UFP) для внутренних логических файлов (ILFs) и внешних интерфейсных файлов (EIFs)

Сложность данных	Количество UFP (ILF)	Количество UFP (EIF)
Low	7	5
Average	10	7
High	15	10

Для иллюстрации рассмотрим пример оценки в не выровненных функциональных точках объекта данных «Клиент» (Рисунок 39).

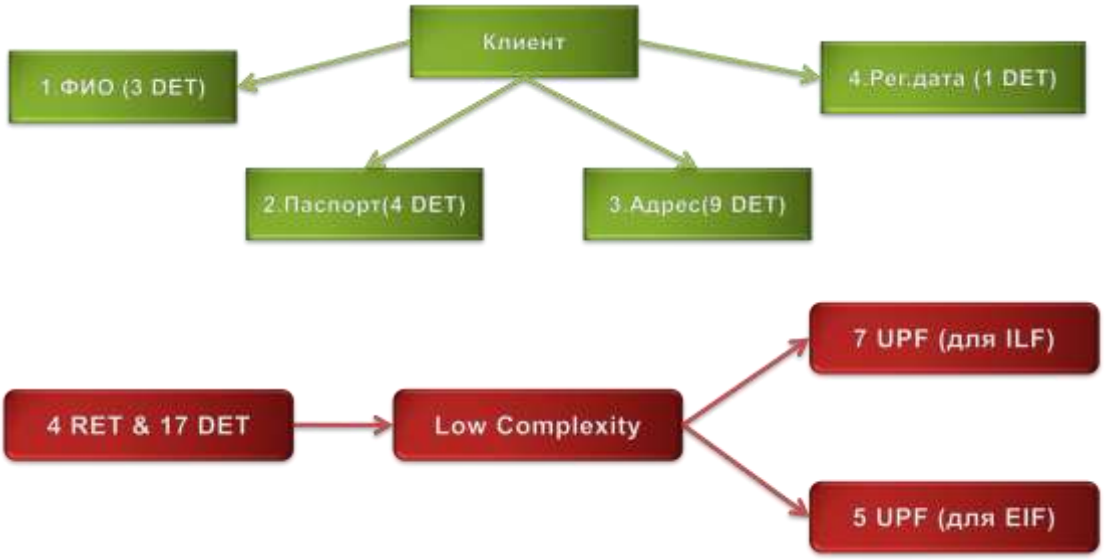

Рисунок 39. Пример оценки в не выровненных функциональных точках объекта данных «Клиент».

Объект «Клиент» содержит четыре логических группы данных, которые в совокупности состоят из 15 неповторяемых уникальное полей данных. Согласно матрице (Таблица 7), нам следует оценить сложность этого объекта данных, как «Low». Теперь, если оцениваемый объект относится к внутренним логическим файлам, то согласно Таблица 8 его сложность будет 7 не выровненных функциональных точек (UPF). Если же объект является внешним интерфейсным файлом, то его сложность составит 5 UPF.

Подсчет функциональных точек, связанных с транзакциями

Подсчет функциональных точек, связанных с транзакциями – это четвертый шаг анализа по методу функциональных точек.

Транзакция – это элементарный неделимый замкнутый процесс, представляющий значение для пользователя и переводящий продукт из одного консистентного состояния в другое.

В методе различаются следующие типы транзакций (Таблица 9):

- EI (external inputs) – внешние входные транзакции, элементарная операция по обработке данных или управляющей информации, поступающих в систему из вне.

- EO (external outputs) – внешние выходные транзакции, элементарная операция по генерации данных или управляющей информации, которые выходят за пределы системы. Предполагает определенную логику обработки или вычислений информации из одного или более ILF.

- EQ (external inquiries) – внешние запросы, элементарная операция, которая в ответ на внешний запрос извлекает данные или управляющую информацию из ILF или EIF.

Таблица 9. Основные отличия между типами транзакций. Легенда: О – основная; Д – дополнительная; NA – не применима.

Функция	**Тип транзакции**		
	EI	**EO**	**EQ**
Изменяет поведение системы	О	Д	NA
Поддержка одного или более ILF	О	Д	NA
Представление информации пользователю	Д	О	О

Оценка сложности транзакции основывается на следующих ее характеристиках:

- FTR (file type referenced) – позволяет подсчитать количество различных файлов (информационных объектов) типа ILF и/или EIF модифицируемых или считываемых в транзакции.

- DET (data element type) – неповторяемое уникальное поле данных. Примеры. EI: поле ввода, кнопка. EO: поле данных отчета, сообщение об ошибке. EQ: поле ввода для поиска, поле вывода результата поиска.

Для оценки сложности транзакций служат матрицы, которые представлены в Таблица 10 и Таблица 11.

- Таблица 10. Матрица сложности внешних входных транзакций (EI)

EI	**1-4 DET**	**5-15 DET**	**16+ DET**
0-1 FTR	Low	Low	Average
2 FTR	Low	Average	High
3+ FTR	Average	High	High

- Таблица 11. Матрица сложности внешних выходных транзакций и внешних запросов (EO & EQ)

EO & EQ	1-5 DET	6-19 DET	20+ DET
0-1 FTR	Low	Low	Average
2-3 FTR	Low	Average	High
4+ FTR	Average	High	High

Оценка транзакций в не выровненных функциональных точках (UFP) может быть получена из матрицы (Таблица 12)

Таблица 12. Сложность транзакций в не выровненных функциональных точках (UFP)

Сложность транзакций	Количество UFP (EI & EQ)	Количество UFP (EO)
Low	3	4
Average	4	5
High	6	7

В качестве примера, рассмотрим оценку управляющей транзакции (EI) для диалогового окна, задающего параметры проверки орфографии в MS Office Outlook (Рисунок 40).

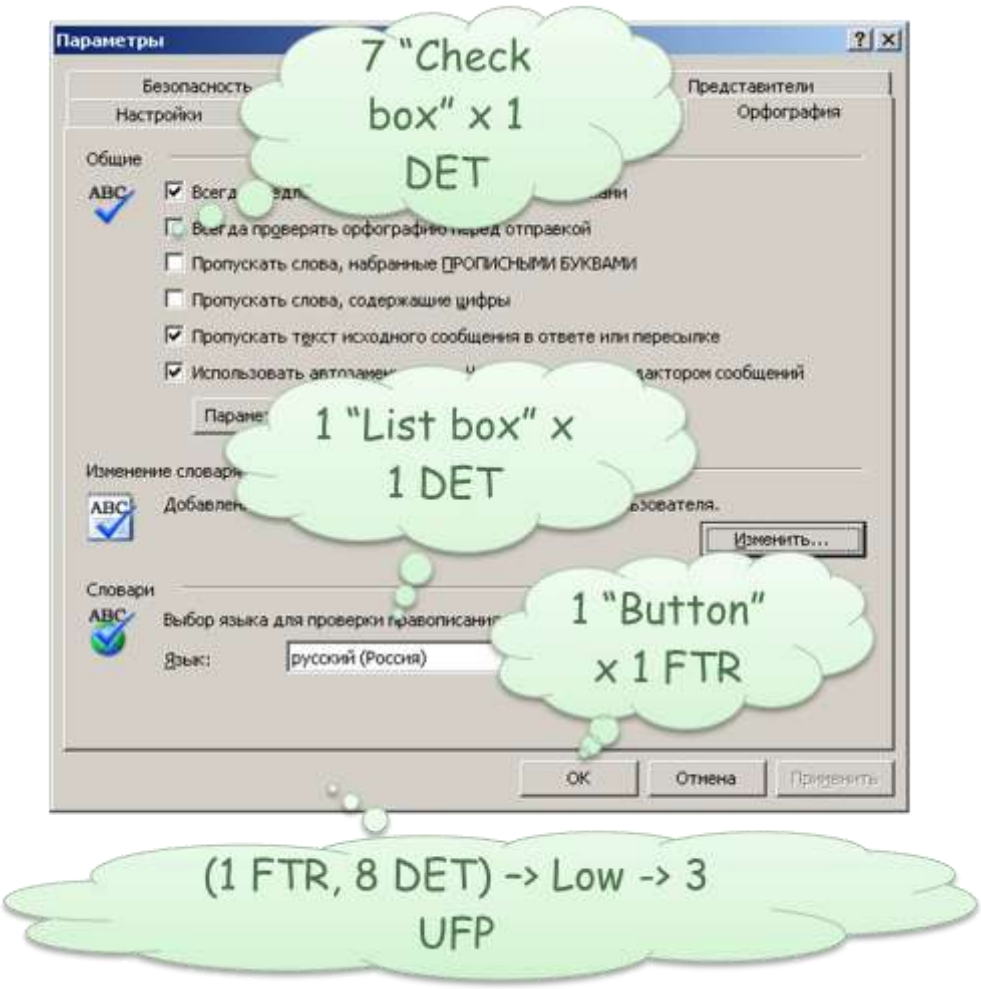

Рисунок 40. Диалоговое окно, управляющее проверкой орфографии в MS Office Outlook

Каждый “Check box” оценивается, как 1 DET. Выпадающий список - 1 DET. Каждая управляющая кнопка должна рассматриваться как отдельная транзакция. Например, если оценивать управляющую транзакцию по кнопке «OK», то, для данной транзакции мы имеем 1 FTR и 8 DET. Поэтому, согласно матрице (Таблица 10), мы можем оценить сложность транзакции, как Low. И, наконец, в соответствие с матрицей (Таблица 12), данная транзакция должна быть оценена в 3 не выровненных функциональных точек (UFP).

Определение суммарного количества не выровненных функциональных точек (UFP)

Общий объем продукта в не выровненных функциональных точках (UFP) определяется путем суммирования по всем информационным объектам (ILF, EIF) и элементарным операциям (транзакциям EI, EO, EQ).

$$UFP = \sum_{ILF} UFPi + \sum_{EIF} UFPi + \sum_{EI} UFPi + \sum_{EO} UFPi + \sum_{EQ} UFPi$$

Определение значения фактора выравнивания (FAV)

Помимо функциональных требований на продукт накладываются общесистемные требования, которые ограничивают разработчиков в выборе решения и увеличивают сложность разработки. Для учета этой сложности

применяется фактор выравнивания (VAF). Значение фактора VAF зависит от 14 параметров, которые определяют системные характеристики продукта:

1. *Обмен данными* (0 – продукт представляет собой автономное приложение; 5 – продукт обменивается данными по более, чем одному телекоммуникационному протоколу).

2. *Распределенная обработка данных* (0 – продукт не перемещает данные; 5 – распределенная обработка данных выполняется несколькими компонентами системы).

3. *Производительность* (0 – пользовательские требования по производительности не установлены; 5 – время отклика сильно ограничено критично для всех бизнес-операций, для удовлетворения требованиям необходимы специальные проектные решения и инструменты анализа.

4. *Ограничения по аппаратным ресурсам* (0 – нет ограничений; 5 – продукт целиком должен функционировать на определенном процессоре и не может быть распределен).

5. *Транзакционная нагрузка* (0 – транзакций не много, без пиков; 5 – число транзакций велико и неравномерно, требуются специальные решения и инструменты).

6. *Интенсивность взаимодействия с пользователем* (0 – все транзакции обрабатываются в пакетном режиме; 5 – более 30% транзакций – интерактивные).

7. *Эргономика* (эффективность работы конечных пользователей) (0 – нет специальных требований; 5 – требования по эффективности очень жесткие).

8. *Интенсивность изменения данных* (ILF) пользователями (0 – не требуются; 5 – изменения интенсивные, жесткие требования по восстановлению).

9. *Сложность обработки* (0 – обработка минимальна; 5 – требования безопасности, логическая и математическая сложность, многопоточность).

10. *Повторное использование* (0 – не требуется; 5 – продукт разрабатывается как стандартный многоразовый компонент).

11. *Удобство инсталляции* (0 – нет требований; 5 – установка и обновление ПО производится автоматически).

12. *Удобство администрирования* (0 – не требуется; 5 – система автоматически самовосстанавливается).

13. *Портируемость* (0 – продукт имеет только 1 инсталляцию на единственном процессоре; 5 – система является распределенной и предполагает установку на различные «железо» и ОС).

14. *Гибкость* (0 – не требуется; 5 – гибкая система запросов и построение произвольных отчетов, модель данных изменяется пользователем в интерактивном режиме).

14 системных параметров (degree of influence, DI) оцениваются по шкале от 0 до 5. Расчет суммарного эффекта 14 системных характеристик (total degree of influence, TDI) осуществляется простым суммированием:

$$TDI = \Sigma\, DI$$

Расчет значения фактора выравнивания производится по формуле

$$VAF = (TDI * 0.01) + 0.65$$

Например, если, каждый из 14 системных параметров получил оценку 3, то их суммарный эффект составит TDI = 3 * 14 = 42. В этом случае значение фактора выравнивания будет: VAF = (42 * 0.01) + 0.65 = 1.07

Расчет количества выровненных функциональных точек (AFP)

Дальнейшая оценка в выровненных функциональных точках зависит от типа оценки. Начальное оценка количества выровненных функциональных точек для программного приложения определяется по следующей формуле:

$$AFP = UFP * VAF.$$

Она учитывает только новую функциональностсть, которая реализуется в продукте. Проект разработки продукта оценивается в DFP (development functional point) по формуле:

$$DFP = (UFP + CFP) * VAF,$$

где *CFP* (conversion functional point) – функциональные точки, подсчитанные для дополнительной функциональности, которая потребуется при установке продукта, например, миграции данных.

Проект доработки и совершенствования продукта оценивается в EFP (enhancement functional point) по формуле:

$$EFP = (ADD + CHGA + CFP) * VAFA + (DEL * VAFB),$$

где

ADD - функциональные точки для добавленной функциональности;

CHGA - функциональные точки для измененных функций, рассчитанные после модификации;

VAFA – величина фактора выравнивания рассчитанного после завершения проекта;

DEL – объем удаленной функциональности;

VAFB - величина фактора выравнивания рассчитанного до начала проекта.

Суммарное влияние процедуры выравнивания лежит в пределах +/- 35% относительно объема рассчитанного в UFP.

Метод анализа функциональных точек ничего не говорит о трудоемкости разработки оцененного продукта. Вопрос решается просто, если компания разработчик имеет собственную статистику трудозатрат на реализацию

функциональных точек. Если такой статистики нет, то для оценки трудоемкости и сроков проекта можно использовать метод COCOMO II.

Основы методики COCOMO II

Методика COCOMO позволяет оценить трудоемкость и время разработки программного продукта. Впервые была опубликована Бари Боэмом [3] в 1981 году в виде результат анализа 63 проектов компании «TRW Aerospace». В 1997 методика была усовершенствована и получила название COCOMO II. Калибровка параметров производилась по 161 проекту разработки. В модели используется формула регрессии с параметрами, определяемыми на основе отраслевых данных и характеристик конкретного проекта.

Различаются две стадии оценки проекта: *предварительная* оценка на начальной фазе и *детальная* оценка после проработки архитектуры.

Формула оценки трудоемкости проекта в чел.*мес. имеет вид:

$$PM = A \times SIZE^{E} \times \prod_{i=1}^{n} EM_i$$

$$A = 2{,}94$$

$$E = B + 0{,}01 \times \sum_{j=1}^{5} SF_j$$

$$B = 0{,}91$$

где

$SIZE$ - размер продукта в KSLOC

EM_i - множители трудоемкости

SF_j - факторы масштаба

$n = 7$ - для предварительной оценки

$n = 17$ - для детальной оценки

Главной особенностью методики является то, что для того, чтобы оценить трудоемкость, необходимо знать размер программного продукта в тысячах строках исходного кода (KSLOC, Kilo Source Lines Of Code). Размер программного продукта может быть, например, оценен экспертами с применением метода PERT.

Если мы провели анализ продукта методом функциональных точек, то его размер может быть рассчитан с использованием собственных статистических данных или с использованием статистики по отрасли [5] (Таблица 13).

Таблица 13. Оценка количества строк, необходимых на реализацию одной не выровненной функциональной точки для некоторых распространенных языков программирования.

Язык программирования	**Оценка количества строк**		
	Наиболее вероятная	**Оптимистичная**	**Пессимистичная**

Assembler	172	86	320
C	148	9	704
C++	60	29	178
C#	59	51	66
J2EE	61	50	100
JavaScript	56	44	65
PL/SQL	46	14	110
Visual Basic	50	14	276

Факторы масштаба

В методике используются пять факторов масштаба SF_j, которые определяются следующими характеристиками проекта:

1. PREC – прецедентность, наличие опыт аналогичных разработок (Very Low – опыт в продукте и платформе отсутствует; Extra High – продукт и платформа полностью знакомы)

2. FLEX – гибкость процесса разработки (Very Low – процесс строго детерминирован; Extra High – определены только общие цели).

3. RESL – архитектура и разрешение рисков (Very Low – риски неизвестны/не проанализированы; Extra High – риски разрешены на 100%)

4. TEAM – сработанность команды (Very Low – формальные взаимодействия; Extra High – полное доверие, взаимозаменяемость и взаимопомощь).

5. PMAT – зрелость процессов (Very Low – CMM Level 1; Extra High – CMM Level 5)

Значение фактора масштаб, в зависимости от оценки его уровня, приведены в Таблица 14

Таблица 14. Значение фактора масштаба, в зависимости от оценки его уровня

Фактор масштаба	**Оценка уровня фактора**					
	Very Low	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PREC	6.20	4.96	3.72	2.48	1.24	0.00
FLEX	5.07	4.05	3.04	2.03	1.01	0.00
RESL	7.07	5.65	4.24	2.83	1.41	0.00
TEAM	5.48	4.38	3.29	2.19	1.10	0.00

PMAT	7.80	6.24	4.68	3.12	1.56	0.00

Множители трудоемкости

В нашу задачу не входит детальное описание метода COCOMO II, поэтому мы рассмотрим только случай предварительной оценки трудоемкости программного проекта. Для этой оценки необходимо оценить для проекта уровень семи множителей трудоемкости M_i:

1. PERS – квалификация персонала (Extra Low – аналитики и программисты имеют низшую квалификацию, текучесть больше 45%; Extra High - аналитики и программисты имеют высшую квалификацию, текучесть меньше 4%)

2. RCPX – сложность и надежность продукта (Extra Low – продукт простой, специальных требований по надежности нет, БД маленькая, документация не требуется; Extra High - продукт очень сложный, требования по надежности жесткие, БД сверхбольшая, документация требуется в полном объеме)

3. RUSE – разработка для повторного использования (Low – не требуется; Extra High – требуется переиспользование в других продуктах)

4. PDIF – сложность платформы разработки (Extra Low – специальные ограничения по памяти и быстродействию отсутствуют, платформа стабильна; Extra High – жесткие ограничения по памяти и быстродействию, платформа нестабильна)

5. PREX – опыт персонала (Extra Low – новое приложение, инструменты и платформа; Extra High - приложение, инструменты и платформа хорошо известны)

6. FCIL – оборудование (Extra Low – инструменты простейшие, коммуникации затруднены; Extra High – интегрированные средства поддержки жизненного цикла, интерактивные мультимедиа коммуникации)

7. SCED – сжатие расписания (Very Low – 75% от номинальной длительности; Very High – 160% от номинальной длительности)

Влияние множителей трудоемкости в зависимости от их уровня определяется их числовыми значениями, которые представлены в матрице, приведенной ниже, (Таблица 15).

Таблица 15. Значения множителей трудоемкости, в зависимости от оценки их уровня

EM$_i$	**Оценка уровня множителя трудоемкости**						
	Extra Low	**Very Low**	**Low**	**Nominal**	**High**	**Very High**	**Extra High**
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.5
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72
RUSE	n/a	n/a	0.95	1.00	1.07	1.15	1.24

PDIF	n/a	n/a	0.87	1.00	1.29	1.81	2.61
PREX	1.59	1.33	1.22	1.00	0.87	0.74	0.62
FCIL	1.43	1.30	1.10	1.0	0.87	0.73	0.62
SCED	n/a	1.43	1.14	1.00	1.00	1.00	n/a

Из этой таблицы, в частности, следует, если в нашем проекте низкая квалификация аналитиков, то его трудоемкость возрастет примерно в 4 раза по сравнению с проектом, в котором участвуют аналитики экстра-класса. И это не выдумки теоретиков, а отраслевая статистика!

Оценка многокомпонентного продукта

Как мы отмечали ранее (см. Лекция 4. Планирование проекта), для того чтобы адекватно спланировать проект и оценить его трудоемкость, необходимо выполнить предварительное проектирование программного продукта. В результате декомпозиции мы получаем некоторое количество компонентов (N), которые составляют программный продукт.

Следует понимать, что суммарная трудоемкость проекта не равна простой сумме трудоемкостей разработки каждого из компонентов:

$$PM \neq \sum_{k=1}^{N} PM_k$$

Простая сумма не учитывает взаимосвязи компонентов и трудозатраты на их интеграцию.

Методика COCOMO II определяет следующую последовательность вычисления трудоемкости проекта при многокомпонентной разработке.

1. Суммарный размер продукта рассчитывается, как сумма размеров его компонентов:

$$SIZE^A = \sum_{k=1}^{N} SIZE_k$$

2. Базовая трудоемкость проекта рассчитывается по формуле:

$$PM^B = A \times (SIZE^A)^E \times SCED$$

3. Затем рассчитывается базовая трудоемкость каждого компонента:

$$PM_k^B = PM^B \times \frac{SIZE_k}{SIZE^A}$$

4. На следующем шаге рассчитывается оценка трудоемкости компонентов с учетом всех множителей трудоемкости, кроме множителя *SCED*.

$$PM'_k = PM_k^B \times \prod_{i=1}^{6} EM_i$$

5. И, наконец, итоговая трудоемкость проекта определятся по формуле:

$$PM = \sum_{k=1}^{N} PM'_k$$

Оценка длительности проекта

Длительность проекта в методике COCOMO II рассчитывается по формуле:

$$TDEV = C \times (PM_{NS})^{D+0,2\times 0,01\times \sum_{j=1}^{5} SF_j} \times \frac{SCED}{100},$$

где

$C = 3{,}67;\ D = 0{,}28;$

PM_{NS} – трудоемкость проекта без учета множителя $SCED$, определяющего сжатие расписания.

Выводы

Оценка трудоемкости должна быть вероятностным утверждением. Это означает, что для нее существует некоторое распределение вероятности, которое может быть очень широким, если неопределенность высокая, или достаточно узким, если неопределенность низкая.

Использование собственного опыта или опыта коллег, полученного в похожих проектах, это наиболее прагматичный подход, который позволяет получить достаточно реалистичные оценки трудоемкости и срока реализации программного проекта, быстро и без больших затрат.

Если собственный опыт аналогичных проектов отсутствует, а коллеги-эксперты недоступны, то необходимо использовать формальные методики, основанные на обобщенном отраслевом опыте. Среди них наибольшее распространение получили два подхода:

- FPA IFPUG - метод функциональных точек,
- метод COCOMO II, Constructive Cost Model.

Не реалистичность оценок один из серьезнейших демотивирующих факторов для участников проектной команды. Недооценка приводит к ошибкам планирования и неэффективному взаимодействию. Агрессивные сроки, постоянное давление, сверхурочные, авралы служат причиной того, что затраты на проект растут экспоненциально и неограниченно.

Дополнительная литература и источники

1. С. Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
2. Function Point Counting Practices Manual, Release 4.2, IFPUG, 2004.
3. Barry Boehm. «Software engineering economics». Englewood Cliffs, NJ:Prentice-Hall, 1981

4. Barry Boehm, et al. «Software cost estimation with COCOMO II». Englewood Cliffs, NJ:Prentice-Hall, 2000.

5. «Function Point Programming Languages Table», Quantitative Software Management, Inc. , 2005.

Лекция 7. Формирование команды

Лидерство и управление

Свое представление о вопросах, связанных с формированием и руководством командами разработчиков ПО, я подробно изложил в книге [1]. В этой лекции остановимся только на ключевых моментах этой деятельности.

В работе руководителя проекта есть две стороны: управление и лидерство, которые одинаково важны и не могут существовать в отрыве друг от друга. Нельзя быть лидером материальных ресурсов, денежных потоков, планов, графиков и рисков. Ими необходимо управлять. Потому что у вещей нет права и свободы выбора, присущих только человеку.

Интеллектуальными людьми невозможно управлять. Творческие команды можно только направлять и вести. «Высокопроизводительное управление в отсутствие эффективного лидерства подобно упорядочению расстановки стульев на палубе тонущего «Титаника». Никакой успех в управлении не компенсирует провала в лидерстве» [2].

Эффективные команды не образуются сами по себе, они кристаллизуются вокруг признанного лидера. Как не бывает лидеров без последователей, так и не бывает команд без лидеров. Поэтому первый шаг руководителя при создании эффективной команды - это стать лидером, вокруг которого сможет сплотиться рабочий коллектив. Лидера нельзя назначить.

Лидерство, это в первую очередь, это умение управлять своей собственной жизнью и только потом другими людьми. Сверхвысокое значение коэффициента интеллекта IQ, к сожалению, в этом не поможет. Личная эффективность человека на 80% определяется его коэффициентом эмоционального интеллекта *EQ* (Emotional Intelligence) [3] – способностью понимать и эффективно взаимодействовать с другими людьми. Хорошая новость. В отличие от IQ, который формируется в ранней молодости и затем практически не меняется, EQ можно повышать на протяжении всей жизни. Если, конечно, прилагать к этому усилия.

Лидер должен получить признание команды. Для того этого необходимо:

- Признание командой профессиональной компетентности и превосходства лидера.

- Полное доверие команды к действиям и решениям лидера, признание его человеческих качеств, убежденность в его честности, порядочности, вера в его искренность и добросовестность.

Если руководитель не смог стать лидером, он будет вынужден применять в своей практике управленческие антипаттерны [1]. Для такого руководителя характерны чрезмерная настороженность, скрытность, неспособность делегировать полномочия. Он исходит из предпосылок индустриальной эпохи Генри Форда: «Работники ленивы, поэтому им необходимы внешние стимулы для работы. У людей нет честолюбия, и они стараются избавиться от ответственности. Чтобы заставить людей трудиться, необходимо использовать принуждение, контроль и угрозу наказания». Манфред Кетс де Врис [3] называет данное отклонение «параноидальным управлением».

Бесполезно пытаться мотивировать участников команды на успех проекта, не исключив из своего руководящего арсенала практики демотивации – антипаттерны, применение которых в управлении творческими коллективами не приносит ничего, кроме вреда. Вместо мотивирования сотрудников на успех, применение антипаттернов мотивирует их на избежание риска и негативных для себя последствий, подавляет свободу, самостоятельность, творчество и инициативу. Это приводит к деструктивному подчинению, когда все работают строго по инструкции и только в соответствие с указаниями руководства, и полному отсутствию личной ответственности исполнителей, «А какие ко мне претензии? Как сказали, так я и сделал!» В результате - низкая эффективность и качество работы, ухудшение морального климата. Вместо доверия и сотрудничества в коллективе процветают подозрительность и формальное взаимодействие. А вы никогда не видели, как тимлид беседует с программистом только «под протокол» и с подписями на каждом листе? Наконец, применение антипаттернов - это стрессы, усталость участников, личные проблемы, увольнение наиболее профессиональных сотрудников и провал проекта.

Эффективный лидер обязан обладать следующими компетенциями:

- Видение целей и стратегии их достижения.
- Глубокий анализ проблем и поиск новых возможностей
- Нацеленность на успех, стремление получить наилучшие результаты.
- Способность сочувствия, понимания состояния участников команды.
- Искренность и открытость в общении.
- Навыки в разрешении конфликтов.
- Умение создавать творческую атмосферу и положительный микроклимат.
- Терпимость, умение принимать людей какие они есть, принятие их права на собственное мнение и на ошибку.
- Умение мотивировать правильное профессиональное поведение членов команды.
- Стремление выявлять и реализовывать индивидуальные возможности для профессионального роста каждого.
- Способность активно "обеспечивать", "доставать", "выбивать" и т.д.

Не существует одной лучшей стратегии руководства. В зависимости от готовности участников рабочей группы выполнять задания руководителя, он должен использовать одну из 4-х стратегий [4]:

1. «Директивное управление». Руководитель говорит, указывает, направляет, устанавливает. Жесткое назначение работ, строгий контроль сроков и результатов.
2. «Объяснения». Лидер "продает", объясняет, проясняет, убеждает. Сочетание директивного и коллективного управления. Объяснение своих решений.

3. «Участие». Лидер участвует, поощряет, сотрудничает, проявляет преданность. Приоритетное коллективное принятие решений, обмен идеями, поддержка инициативы подчиненных.

4. «Делегирование». Лидер делегирует, наблюдает, обслуживает. «Не мешать» - пассивное управление сформировавшегося лидера.

Применение этих стратегий можно проиллюстрировать на примерах (Рисунок 41).

1. Вас назначили руководителем в новый коллектив. Вы еще не получили признания, а дело делать надо. Стратегия: «Директивное управление».

2. Вы были участником команды. Вас назначили руководителем этой команды. Доверие есть, а уверенности в правильности ваших действий нет. Стратегия: «Объяснения».

3. Вас назначили руководителем в новый коллектив. Все знают о ваших прежних сложных и успешных проектах. Все признают ваше превосходство, но доверия к вам нет. Никто не знает, какой ценой были достигнуты ваши победы. Стратегия: «Участие».

4. Между вами и участниками установлено взаимное доверие. Все достаточно мотивированы на успех проекта. Каждый сам себе может быть руководителем. Стратегия: «Делегирование».

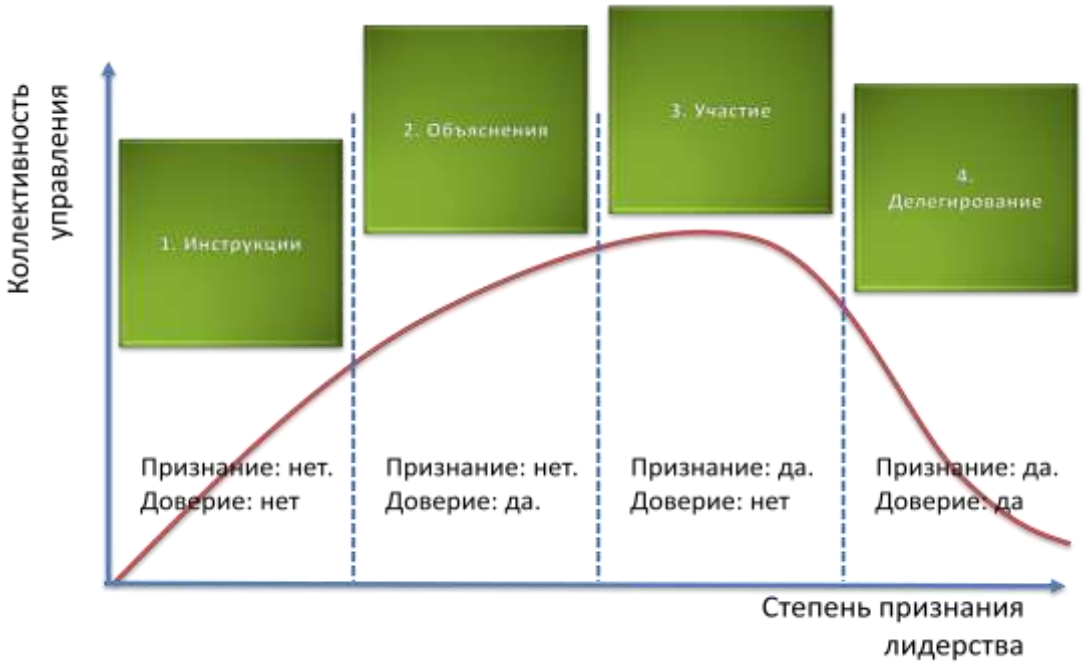

Рисунок 41. Ситуационное лидерство.

Основные усилия руководителя, если он стремится получить наивысшую производительность рабочей группы, должны быть направлены на изучение и изменение объекта управления: людей и их взаимодействия. Следовательно, задачу адаптивного управления мы можем разделить на две подзадачи:

1. Обеспечить эффективность каждого участника рабочей группы.

2. Обеспечить эффективные процессы взаимодействия.

Правильные люди

Рональд Рейган говорил: «Окружите себя самыми лучшими людьми, которых вы только сможете найти, передайте им в руки власть и не мешайте им».

За годы работы у меня сформировалось собственное видение правильного командного поведения. Эффективный командный игрок:

- Занимает активную позицию, стремится расширить свою ответственность и увеличить личный вклад в общее дело.

- Постоянно приобретет новые профессиональные знания и опыт, выдвигает новые идеи, направленные на повышение эффективности достижения общих целей, добивается распространения своих знаний, опыта и идей среди коллег.

- Получает удовольствие от своей работы, гордится ее результатами и стремится, чтобы эти же чувства испытывали все коллеги.

- Четко осознает свои личные и общие цели, понимает их взаимообусловленность, настойчиво стремится к их достижению.

- Уверен в себе и в своих коллегах, объективно оценивает их достижения и успехи, внимательно относится к их интересам и мнениям, активно ищет взаимовыгодное решение в конфликтах.

- Является оптимистом, при этом твердо знает, что окружающий мир несовершенен; воспринимает каждую новую проблему, как дополнительную возможность подтвердить собственный профессионализм в своих глазах и во мнении коллег.

Вместе с тем, встречаются патологии поведения, которые, на мой взгляд, неприемлемы в команде:

- *Непорядочность*. Лживость, отсутствие совести и чувства справедливости, способность на низкие поступки.

- *Синдром острого дефицита эмпатии*. Эгоцентризм. Неуважение и невнимание к партнерам. Склонность к отрицательным оценкам других. Грубость. «Каждый сам за себя! – никто тебе не поможет!» «Человек человеку волк!»

- *«Звезданутость».* Завышенная самооценка. Подчеркивание собственного превосходства. Умничанье. Человек сильно переоценивает свой личный вклад в общее дело и поэтому считает, что он должен работать меньше, чем его «менее способные» коллеги.

- *Вульгарный анархизм*. Вольница – это полная безответственность, свобода от каких либо обязательств перед другими, ничем не сдерживаемые проявления чувств, действия или поступки. «Произвольничать, поступать самовольно, в обиду другим, нагло, дерзко» (с) В.Даль. Не путать со «свободой»!

- *«Социальный паразитизм»*. Стремление прожить вольготно за чужой счет там, где ответственность размыта, а личный вклад трудно четко выделить.

Моя рекомендация: лечить патологию «хирургически» – избавляться от проблемных людей. Воспитывают в детском саду, ну еще немного в начальной школе. Дальше люди воспитываются только самостоятельно, а окружающие могут лишь помогать или не мешать в этом процессе. Убежден, что каждый взрослый человек имеет то, к чему он осознанно или неосознанно стремится. Нянчиться и воспитывать человека - это значит ограждать его от проблем, закрывать ему путь к переосмыслению своего опыта и развитию, «загонять болезнь внутрь» при помощи «социального аспирина».

Я знаю ровно четыре необходимых и достаточных условия для того, чтобы сотрудник эффективно решил поставленную мной задачу. Это

1. Понимание целей работы.
2. Умение ее делать.
3. Возможность ее сделать.
4. Желание ее сделать.

Для того чтобы обеспечить выполнение этих условий, руководитель должен уметь эффективно выполнять четыре функции:

1. *Направлять*. Если сотрудник не понимает что делать, задача руководителя - обеспечить общее видение целей и стратегии их достижения.
2. *Обучать*. Если сотрудник не умеет, задача руководителя – «обучать», быть наставником и образцом для подражания.
3. *Помогать*. Если у сотрудника не может выполнить работу, задача руководителя – «помогать», обеспечить исполнителя всем необходимым, убрать препятствия с его пути.
4. *Вдохновлять*. Если у сотрудника не достаточно желание выполнить работу, задача руководителя – «вдохновить», обеспечить адекватную мотивацию участника на протяжении всего проекта.

Мотивация

Известно, что ни одна задача не будет решена за любое, отведенное на это время, если человек не захочет ее сделать. Он всегда найдет для оправдания этого 100 «объективных» причин, вместо того, чтобы найти хотя бы один способ решения задачи. У каждого участника рабочей группы должна быть личная цель (внутренняя мотивация), которую он сможет достичь, продвигая проект к успеху.

Начните с себя! Вам нужно четко понимать, в чем состоит ваш выигрыш в случае успешного завершения проекта. Добиться от участников приверженности проекту больше, чем имеете вы сами, вам не удастся. Если у участника нет такой значимой личной цели, избавьтесь от него. Иначе вам придется потратить все свое время на «промывание его мозгов» и попытки мотивировать его на эффективную работу.

Мотивация должна начинаться с подбора сотрудников в команду. В старой экономике людей нанимали за умения и обучали нужному отношению к делу. В новой экономике необходимо поступать с точностью до наоборот: нанимать за нужное отношение к делу и учить необходимым умениям.

Люди не рождаются победителями, они ими становятся. Кандидата стоит нанимать только в случае, если вы можете предложить ему возможность стать победителем. Настоящий лидер предлагает не работу, а возможности.

Все люди разные, а ситуаций, в которых они могут находиться в ходе проекта, бесчисленное множество. Бойтесь стереотипов. Если вы не учитываете индивидуальные особенности конкретной личности, то эффективность ваших взаимодействий сильно снижается. Модель объекта управления нам неизвестна, следовательно, не может существовать исчерпывающий набор правил, типа «если…, то…», по которым смог бы действовать руководитель. Поэтому, сколько людей и ситуаций, столько и вариантов решений должен иметь эффективный руководитель в своем запасе. «Если у руководителя в руках только молоток, то все вокруг будут похожи на гвозди».

Руководитель при поиске решения опирается на свой багаж знаний и умений. Он пытается понять каждого участника, классифицировать состояние, найти в своем опыте похожую ситуацию и адаптировать ранее использованное успешное решение применительно к данному конкретному случаю. Таким образом, руководитель стремится помочь человеку (объекту управления) перейти в новое более эффективное с точки зрения целей проекта состояние.

Затем руководитель должен наблюдать за результатами своего воздействия – это и есть дополнительный контур обратной связи. Необходимо помнить, что понять человека можно, только слушая и *слыша*, что он говорит. Руководитель, который в течение недели не пообщался индивидуально с каждым из своих прямых подчиненных, зря получает зарплату. И совсем не обязательно разговор должен идти о статусе проектных работ. Порой, достаточно поговорить о погоде, кино или футболе. После этого руководитель анализирует полученные результаты и аккумулирует новый опыт (положительный или отрицательный) в своей «базе знаний».

Чем опытней руководитель, тем точнее он может распознать и классифицировать сложившуюся ситуацию, тем больше в его «базе знаний» прецедентов, используя которые, он может синтезировать решение для данного конкретного случая. Именно поэтому в управлении программными проектами в первую очередь ценится опыт руководителя и только потом, возможно, его звания и знания.

Программист состоит из четырех компонентов: тело, сердце, разум и душа.

1. Телу необходимы деньги и безопасность.

2. Сердцу - любовь и признание.

3. Разуму – развитие и самосовершенствование.

4. Душе – самореализация.

Предоставьте все это вашим сотрудникам, и эффективность их труда возрастет многократно. В противном случае люди, которые хотят побеждать, найдут все это в другой команде, а в вашей останутся только неудачники.

Эффективное взаимодействие

Ранее мы уже говорили, что процесс производства программного обеспечения, применяемый в проекте, должен основываться на итеративности, инкрементальности, самоуправляемости команды и адаптивности. Главный принцип: *не люди должны строиться под выбранную модель процесса, а модель процесса должна подстраиваться под конкретную команду*, чтобы обеспечить ее наивысшую производительность.

В программной инженерии многие уже признали, что наиболее эффективные производственные процессы складываются в самоуправляемых и самоорганизующихся рабочих командах. Идеи командного менеджмента на Западе зародилась в начале 80-х годов. Эффективность команд в новых экономических условиях одними из первых оценили такие гиганты, как Procter & Gamble и Boeing [5]. Доктрина командного менеджмента предполагает ясность общих ценностей и целей, самоорганизацию и самоуправление совместной деятельностью, взаимный контроль, взаимопомощь и взаимозаменяемость, коллективную ответственность за результаты труда, всемерное развитие и использование индивидуального и группового потенциалов.

Если на Западе идеи командного менеджмента только зарождаются, то для России они имеют давние традиции. С XIII века в России существовали производственные артели - различные формы добровольных объединений людей с целью осуществления общей хозяйственной деятельности. Артель была добровольным товариществом совершенно равноправных работников, призванных на основе взаимопомощи и взаимовыручки решать практически любые хозяйственные и производственные задачи. Известный российский писатель и революционер, А.И. Герцен видел в коллективизме, в уникальности существующей сельской общины, в городской артели и в самобытной воинской организации казачества характерную черту жизни и психологии русского народа. Позднее во времена СССР широкое распространение на производстве получили рабочие бригады, а в научно-исследовательских институтах - временные научные творческие коллективы, которые объединялись на основе общности цели, взаимопомощи и коллективной ответственности за результат.

Истинный российский коллективизм (соборность) не имеет ничего общего с вульгарным коллективизмом: со стадностью, подавлением личности («я нет, есть только мы!», «я – последняя буква алфавита!»), уступчивостью, групповой психологией и слепым подчинением меньшинства большинству. Суть российского коллективизма в том, что человек ощущает себя элементом органичной системы, где он выполняет свою функцию, свою задачу, совершенно особенную, которую не выполняет никто другой и выполнить не может. Эту задачу он выполняет совершенно сознательно, стремясь к максимальной реализации своих целей.

Руководителю недостаточно стать лидером, надо еще суметь сплотить команду. Эксперты в области командного менеджмента выделяют 4 обязательные последовательные стадии, через которые должна пройти рабочая группа прежде, чем она станет эффективной командой.

1. *Forming*. Формирование. Характеризуется избытком энтузиазма, связанного с новизной. Люди должны преодолеть внутренние противоречия, переболеть конфликтами прежде, чем сформируется действительно спаянный коллектив. На этом этапе многое зависит от руководителя. Он должен четко поставить цели членам команды, верно определить роль каждого в проекте.

2. *Storming*. Разногласия и конфликты. Самый сложный и опасный период. Мотивация новизны уже исчезла, а сильные и глубокие стимулы у команды еще не появились. Неизбежные сложности или неудачи порождают конфликты и «поиск виновных». Участники команды методом проб и ошибок вырабатывают наиболее эффективные процессы взаимодействия. Руководителю на этом этапе важно обеспечить открытую коммуникацию в команде. Конфликты не следует прятать или разрубать. Споры необходимо разруливать спокойно, терпеливо и тщательно.

3. *Norming*. Становление. В команде растет доверие, люди начинают замечать в коллегах не только проблемные, но и сильные стороны. Закрепляются и оттачиваются наиболее эффективные процессы взаимодействия. На смену битве амбиций приходит продуктивное сотрудничество. Четче становится разделение труда, исчезает дублирование функций. Руководитель перестает находиться в состоянии постоянного аврала, работа по построению команды на этом этапе – уже не тушение пожара, а скрупулезный труд по отработке общих норм и правил.

4. *Performing*. Отдача. Команда работает эффективно, высок командный дух, люди хорошо знают друг друга и умеют использовать сильные стороны коллег. Все стремятся придерживаться выработанных общих процессов. Высок уровень доверия. Это лучший период для раскрытия индивидуальных талантов.

Часто случается, что рабочая группа вязнет на одной из стадий и никогда не достигает плато наивысшей производительности.

Если команда прошла все стадии формирования и вышла на фазу «Performing», не стоит полагать, что менеджер проекта может делегировать все свои полномочия и отправиться в отпуск. Задача менеджера на этом этапе - «точить пилу». Это значит поддерживать требуемый уровень мотивации, быть штурманом, искать новые пути и открывать новые возможности. Постоянно наблюдать и оценивать эффективность всех процессов, применяемых в проекте. Искать ответ на вопросы: «Что угрожает проекту?» «Что лишнее мы делаем?» «Что можно делать проще?» Работать на сокращение ненужных усилий вместо того, чтобы «стремиться к новым героическим подвигам».

Если руководитель не будет прилагать дополнительные усилия команда, рано или поздно, начнет «сползать» с плато наивысшей эффективности в состояние застоя и стагнации (Рисунок 42). Помните, что окружение и команда изменяются по ходу проекта. Прежняя мотивация ослабевает или перестает действовать. Изменяйте правила и процессы. Отказывайтесь от того, что перестало действовать или стало работать неэффективно. «Встряхивайте» (*Reforming*) и возвращайте команду в стадию «Forming». Это позволит ей снова, пройдя через все этапы становления, выйти на новый более высокий уровень производительности. Разумеется, делать это следует, после сдачи очередного релиза программного продукта, ну и, возможно, в случае глубокого кризиса проекта.

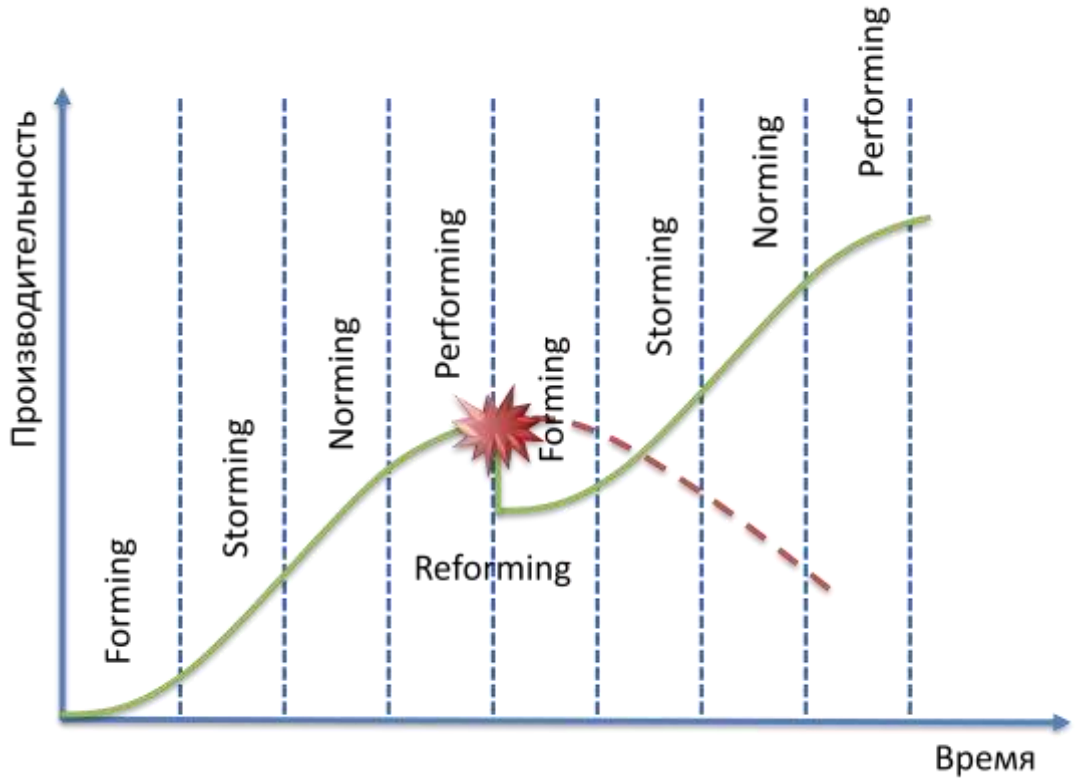

Рисунок 42. Reforming. «Встряхивание» и перевод команды проекта на новый, более высокий, уровень производительности.

Выводы

Эффективные команды не образуются сами по себе, они кристаллизуются вокруг признанного лидера. Для того чтобы стать лидером, необходимо:

- Признание командой профессиональной компетентности и превосходства лидера.
- Полное доверие команды к действиям и решениям лидера, признание его человеческих качеств, убежденность в его честности, порядочности, вера в его искренность и добросовестность.

Мотивация должна начинаться с подбора сотрудников в команду. В старой экономике людей нанимали за умения и обучали нужному отношению к делу. В новой экономике необходимо поступать с точностью до наоборот: нанимать за нужное отношение к делу и учить необходимым умениям.

Рабочая группа прежде, чем она станет эффективной командой, должна пройти четыре обязательных последовательных стадии: 1) Forming, 2) Storming, 3) Norming, 4) Performing.

Если руководитель не будет прилагать дополнительные усилия команда, рано или поздно, начнет «сползать» с плато наивысшей эффективности в состояние застоя и стагнации. Задача менеджера на этом этапе - «точить пилу»: поддерживать требуемый уровень мотивации, быть штурманом, искать новые пути и открывать новые возможности. Четыре стадии развития команды

должны циклически повторяться, чтобы обеспечить непрерывный рост производительности.

Дополнительная литература и источники

1. С. Архипенков, "Руководство командой разработчиков программного обеспечения. Прикладные мысли", 2008 (http://www.happy-pm.com/sw_team_management.pdf).

2. Стивен Р. Кови, «7 навыков высокоэффективных людей. Мощные инструменты развития личности», 2-е изд., М., Альпина Бизнес Букс, 2007.

3. Манфред Кетс де Врис, «Мистика лидерства. Развитие эмоционального интеллекта», М., Альпина Бизнес Букс, 2005.

4. Hersey P., Blanchard K.H. “Management of Organizational Behavior”, 6th ed., Englewood Cliffs: Prentice-Hall, 1993.

5. Л. Томпсон, «Создание команды», М., Вершина, 2005.

Лекция 8. Реализация проекта

Рабочее планирование

Управление - это расчленение, анализ, определение последовательности действий, конкретная реализация. Управление фокусируется на нижнем уровне: как мне сделать это наилучшим образом? Эта компетенция руководителя определяет эффективность движения по выбранному пути. Как правило, менеджеры, вышедшие из программистов, без особого труда овладевают необходимыми управленческими навыками.

Базовое расписание, составленное на этапе планирования проекта, служит ориентиром для мониторинга состояния дел на макроуровне. Для оперативного управления проектом используется рабочий план. Рабочее планирование рекомендуется выполнять методом «набегающей волны»: работа, которую надо будет выполнить в ближайшей перспективе, подробно планируется на низшем уровне ИСР, а далеко отстоящая работа планируется на сравнительно высоком уровне ИСР.

Элементарная работа, как правило, представляет собой отдельное функциональное требование к программному продукту или запрос на изменение, над которым последовательно работают: бизнес-аналитик, проектировщик, разработчик, тестировщик и документалист. Трудоемкость элементарной работы каждого из исполнителей должна быть от 4 до 20 чел.*час. Если трудоемкость задачи не укладывается в эти пределы, следует провести декомпозицию работы.

Для рабочего планирования целесообразно использовать систему управления задачами или багтрекинга, поскольку она позволяет задавать последовательность переходов задачи от исполнителя к исполнителю, управлять приоритетами работ и адекватно отслеживать их статус: анализ, проектирование, кодирование, тестирование, документирование. Работа должна считаться законченной только тогда, когда реализация требования протестирована и документирована.

В зависимости от уровня профессионализма и зрелости команды проекта распределение работ может осуществляться либо директивно с жесткой постановкой срока и контролем исполнения каждой задачи, либо эти полномочия делегируются исполнителям. В этом случае они сами выбирают задачи последовательно в соответствие с приоритетами, а их выполнение анализируется периодически на статус митинге. Можно рекомендовать еженедельные собрания по статусу проекта всей команды или, если проект достаточно большой, то ключевых его частников: руководителей подпроектов и лидеров команд. Хорошее время для этого утро понедельника, поскольку участники проекта, особенно студенты, которые совмещают учебу и работу, часто работают в выходные, но, разумеется, не потому, что аврал, а потому что им так удобнее. Обсуждаются, как правило, всего три вопроса:

- Угрозы и проблемы.
- Анализ результатов за неделю.
- Уточнение приоритетов задач на новую неделю.

Как правило, нет смысла оценивать процент реализации работы в промежуточном состоянии, поскольку, если задача передана в тестирование, то

это вовсе не означает, что 70% работы сделаны. На этапе тестирования может быть выявлена ошибка проектирования и вся работа начнется заново. Рекомендация - использовать правило «50/100». Если работа по задаче начата, то следует учитывать ее, как выполненную на 50%. А 100% поучает только протестированная и документированная работа.

Принципы количественного управления

«Тем, что нельзя измерить, нельзя управлять». Измерения по проекту необходимо выполнять регулярно, не реже одного раза в 1-2 недели. Для каждого измеримого показателя должны быть определены его плановые значения. Для каждого планового значения должны быть определены три области критичности отклонений:

- Допустимые отклонения. Предполагается, что никаких управляющих воздействий не требуется.
- Критичные отклонения. Требуется тщательный анализ причин отклонения и при необходимости применение корректирующих действий.
- Недопустимые отклонения. Требуется срочный анализ причин отклонения и обязательное применение корректирующих действий.

Измерения необходимо производить регулярно. Цель – выявить причины наступивших или возможных критичных и недопустимых отклонений. Результатом анализа должны стать планирование корректирующих действий по компенсации недопустимых отклонений, их реализация и мониторинг результативности применения этих корректирующих действий.

Все измерения необходимо сохранять в репозитарии проекта. Измерения, накопленные в ходе проекте, являются наиболее достоверной основой при детальной оценке и планировании работ на следующих итерациях проекта.

Поскольку главная задача менеджера удержать проект в пределах «железного» треугольника, то, в первую очередь, необходимо анализировать отклонения проекта по срокам и затратам. Делается это при помощи метода освоенного объема [1]. Приходилось сталкиваться с мнением, что этот метод не применим в управлении программными проектами. Это действительно так, если мы используем «водопадную» модель процесса разработки. Но если ИСР проекта, ориентирована на инкрементальную разработку, то это означает, что на верхних уровнях декомпозиции находятся компоненты проектного продукта и их функционал, а не производственные процессы. Следовательно, если в проекте реализованы, протестированы и документированы 50 % функциональных требований, то есть все основания полагать, что осталась приблизительно половина проектных работ.

Суть метода оценки проекта по освоенному объему заключается в следующем. Сначала оценивается отклонение от графика ***SV*** (*Shedule Variance*) в денежных единицах:

$$SV = EV - PV,$$

где

EV (Earned Value) - освоенный объем. Плановая стоимость выполненных работ. Объем выполненных работ, выраженный в терминах одобренного бюджета,

выделенного на эти работы для плановой операции и элемента иерархической структуры работ;

PV (Planned Value) - плановый объем. Плановая стоимость запланированных работ. Утвержденный бюджет, выделенный на плановые работы, выполняемые в рамках плановой операции или элемента иерархической структуры работ.

Например. Пусть мы на текущий момент реализовали (протестировали и документировали) 20 функциональных требований, на каждое из которых было запланировано затратить по 40 чел.*час. по 1000 руб, то освоенный объем будет

$$EV = 20 * 40 * 1000 = 800\,000 \text{ руб.}$$

Если же на текущий момент планировалось реализовать только 15 требований, то плановый объем будет

$$PV = 15 * 40 * 1000 = 600\,000 \text{ руб.}$$

Следовательно, мы опережаем график (отклонение от графика положительное) на величину

$$SV = EV – PV = 800\,000 – 600\,000 = 200\,000 \text{ руб.}$$

Как пересчитывается отклонение от графика, выраженное в денежных единицах, в сокращение сроков проекта иллюстрируется на Рисунок 43.

Если мы опережаем график, то это не обязательно означает что проект идет успешно. Хорошо это или плохо зависит от значения другого показателя метода освоенного объема: ***CV*** (Cost Variance) - отклонения по затратам, которое оценивается по формуле:

$$CV = EV – AC$$

где

AC, (Actual Cost) - фактические затраты. Фактическая стоимость выполненных работ. Фактические затраты на выполнение работ за определенный период в рамках плановой операции или элемента иерархической структуры работ.

Например, если мы для того что сократить время работ по проекту работали 25% времени сверхурочно и в выходные дни с двойной оплатой, то фактические трудозатраты составили:

$$AC = 20 * (30 * 1000 + 10 * 2000) = 1\,000\,000 \text{ руб.}$$

Поэтому отклонения по затратам в нашем случае будет

$$CV = EV – AC\ PV = 800\,000 – 1\,000\,000 = -200\,000 \text{ руб.}$$

Отрицательное значение отклонения по затратам означает, что мы превысили бюджет, что, в общем случае, не очень хорошо. Но если срок завершения проекта для нас имеет высший приоритет, и наши прогнозируемые затраты по завершению проекта не превышают плановых с учетом управленческого резерва (Рисунок 43), то в этом случае можно считать, что проект выполняется успешно.

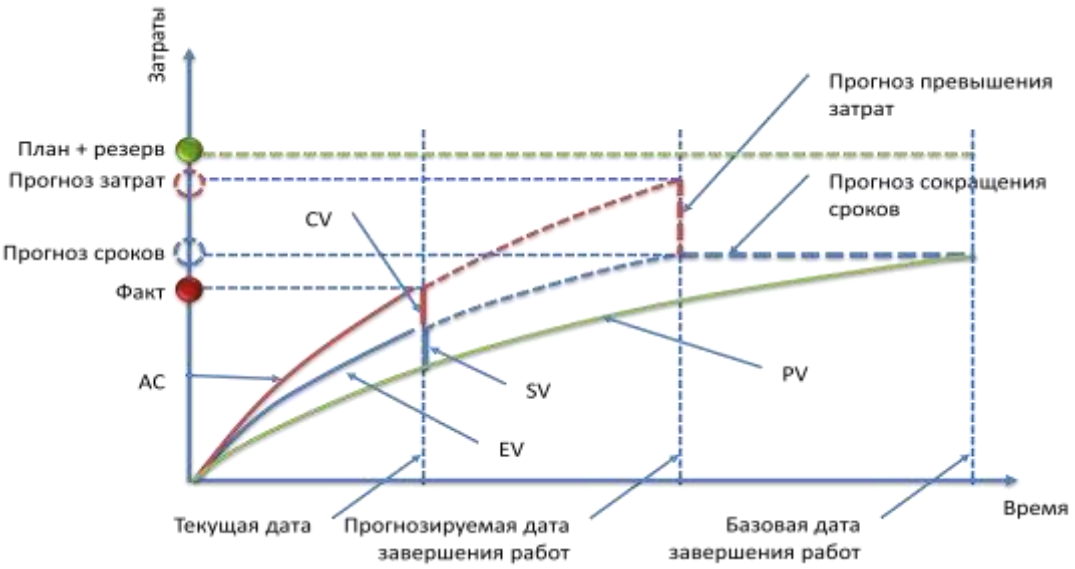

Рисунок 43. Оценка и прогноз показателей по методу освоенного объема

Отклонение от бюджета и по срокам в абсолютных денежных единицах недостаточно для характеристики проектов разных масштабов. Более наглядны относительные показатели: индекс выполнения сроков ***SPI*** (Schedule Performance Index)

$$SPI = EV / PV$$

и индекс выполнения стоимости ***CPI*** (Cost Performance Index)

$$CPI = EV / AC,$$

которые характеризуют проект независимо от его размера. Если значения обоих индексов больше 1, то это свидетельствуют о благополучном состоянии в проекте.

Какие еще измеримые показатели целесообразно применять в управлении программным проектом?

В первую очередь это *показатель прогресса проекта*, доля реализованных и проверенных высокоуровневых требований к проекту, например отношение числа завершенных сценариев использования продукта к их общему числу.

Другой показатель - *стабильность проекта*, общее количество принятых (утвержденных спонсором или заказчиком) изменений в плане управления проектом. Чем выше нестабильность в проекте, тем больше сложность в управлении работами и ниже производительность участников.

Если кто-то думает что код это решение проблемы, то это не так. Код это новый источник проблем. Поэтому всегда следует измерять текущий *размер проекта* - количество строк исходного кода, добавленных, измененных и удаленных в ходе выполнения проекта разработки ПО. Чем больше объем исходного кода, тем больше времени потребуется на внесение изменений и исправление ошибок.

При увеличении объема проектного продукта трудозатраты на каждую новую строку исходного кода увеличиваются. Если за номинал взять

производительность проектной команды при производстве продукта в 10 KSLOC, то та же команда на проекте в 100 KSLOC покажет производительность в 1.3 – 1.7 раз меньшую, а на проекте в 1000 KSLOC следует ожидать, что производительность снизится в 1.6 – 3.0 раза [2].

Большой объем кода так же потребует большего количества людей на его сопровождение. Поскольку, даже если будет выявляться только несколько критических ошибок в год, то для того чтобы их исправить в приемлемые сроки, например, за 24 часа, в продукте общим объемом в 1000 KSLOC то один программист с этим не справится. Это связано с тем, что для того чтобы исправить ошибку в ограниченные сроки необходимо оперативно выявить и устранить ее причину, а для этого надо хорошо знать архитектуру и код программного продукта. Чтобы эффективно сопровождать продукт подобного объема необходимо иметь в «горячем» резерве примерно 20 разработчиков, потому что 50 KSLOC, на мой взгляд, это предельный объем кода, который может удерживать в голове и эффективно сопровождать один человек. И еще проблема: чем этих людей занимать в свободное от исправлений ошибок время, если нет новых проектов развития продукта.

Следующий важный показатель состояния проекта – это *средняя производительность*, отношение текущего размера проекта к фактическим затратам по проекту. С. Макконнелл [2] приводит следующие показатели (минимальное, максимальное и среднее значение) производительности в KSLOC на один чел.*мес. фактических затрат для стандартных типов проектов объемом в 100 KSLOC:

- 300-7000 (800) - интранет система.
- 200-7000 (600) - бизнес система.
- 100-2000 (300) - Интернет система.
- 50-600 (100) - системное ПО, телекоммуникации.
- 20-300 (50) - системы реального времени.

Высокая производительность в проекте – это далеко не всегда хороший признак. Приходилось встречаться с проектами, в которых вследствие активного применения метода «copy+past», средняя производительность в разработке бизнес системы достигала 2000 SLOC/чел.*мес. Однако для реализации требуемого функционала было написано в 3 – 4 раза больше кода, чем это могло бы потребоваться при адекватной проработке архитектуры.

Еще одна группа количественных показателей, которые следует наблюдать в ходе реализации проекта, характеризует *качество программного продукта*:

- Дефектность продукта – количество выявленных дефектов на единицу объема продукта (например, KSLOC).
- Доля не устраненных дефектов - отношение количества незакрытых максимально критичных и критичных дефектов к количеству выявленных несоответствий.
- Средние затраты на сопровождение – средние трудозатраты на исправление одного дефекта. Высокое значение этого показателя может свидетельствовать о некачественной архитектуре программного продукта.

- Документированность кода - определяет процент строк исходного кода с комментарии по отношению к общему количеству строк.

Следует подчеркнуть, что наблюдать надо за средними по проекту значениями показателей, и ни в коем случае не пытаться измерять индивидуальные характеристики производительности и качества. Главные причины, почему это не следует делать, заключаются в том, что, во-первых, в этом случае вместо слаженной командной работы мы получим личную конкуренцию, а, во-вторых, наиболее «продвинутые» разработчики станут работать на формальные показатели, а не на достижение целей проекта.

Если команда действительно состоялась, то для нее характерна коллективная ответственность за достижение общих целей. И, как пишет, Т.Демарко [3], «менеджер проекта должен занимать очередь, чтобы покритиковать сотрудника, не выполняющего свои обещания», поскольку в правильной команде для этого всегда найдется масса желающих.

Завершение проекта

Главная цель этой фазы – проверить и передать заказчику результат проекта. Для этого необходимо выполнить приемо-сдаточные работы в соответствии с процедурой приемки, которая должна быть определена заранее на самой ранней стадии проекта.

Результаты проекта должны быть переданы во внедрение или сопровождение, или должным образом законсервированы для дальнейшего использования. Не должно оставаться «зависших» работ по проекту. Все линейные руководители всех участников должны быть извещены о завершении работ по проекту, и освобождении сотрудников.

Важная задача, которая должна быть решена на данной фазе, это реализация обратной связи по проекту. Цель – сохранить результаты, знания и опыт, полученные в проекте, для более эффективного и качественного выполнения аналогичных проектов в будущем. Необходимо архивировать все результаты, документировать опыт, уроки по проекту и предложения по улучшению технологии выполнения работ и управления проектами.

Все проекты и в особенности провальные проекты должны завершаться итоговым отчетом, если компания не хочет «наступать на одни и те же грабли». Помним о том, что «вчерашние проблемы, это сегодняшние риски».

Итоговый отчет должен содержать следующую информацию:

- Итоги проекта:
    - Достижение целей проекта
    - Дополнительные полезные результаты
    - Фактические сроки
    - Фактические расходы
    - Обоснование отклонения от целей
    - Отклонения результатов от требований

- Уроки проекта
    - Проблемы проекта и способы их решения
    - Материалы программные компоненты для последующего использования
    - Предложения по изменению технологий или стандартов компании

На фазе завершения желательно реализовать и план мотивации участников проектной команды, поскольку отложенное вознаграждение мотивирует существенно слабее.

Выводы

Для оперативного управления проектом используется рабочий план. Элементарная работа, как правило, представляет собой отдельное функциональное требование к программному продукту или запрос на изменение, над которым последовательно работают: бизнес-аналитик, проектировщик, разработчик, тестировщик и документалист.

Измерения по проекту необходимо выполнять регулярно, не реже одного раза в 1-2 недели. Для каждого измеримого показателя должны быть определены его плановые значения и допустимые отклонения.

В состав измеряемых показателей должны входить следующие характеристики проекта:

- Освоенный и плановый объемы работ и фактические затраты по проекту.
- Показатели прогресса и стабильности проекта.
- Размер продукта.
- Производительность.
- Показатели качества программного продукта.

По результатам проекта обязательно должна быть реализована обратная связь. Цель – сохранить результаты, знания и опыт, полученные в проекте, для более эффективного и качественного выполнения аналогичных проектов в будущем.

Дополнительная литература и источники

1. «PMBOK. Руководство к Своду знаний по управлению проектами», 3-е изд., PMI, 2004.
2. С. Макконнелл, «Сколько стоит программный проект», «Питер», 2007.
3. Том Демарко, Тимоти Листер, «Человеческий фактор: успешные проекты и команды», Спб. Символ-Плюс, 2005.

Заключение. Растите профессионалов

Работа менеджера проекта подобна труду садовника.

Подобно тому, как садовник любовно отбирает наиболее подходящие растения для своего будущего сада, менеджер набирает людей, наиболее соответствующих целям проекта.

Подобно тому, как садовник ищет лучшую почву для каждого растения с учетом его особенностей, менеджер для каждого участника проектной команды ищет наиболее подходящую для него задачу.

Подобно тому, как садовник тщательно лелеет своих питомцев, оберегает их от вредных воздействий, следит за тем, чтобы ни одно растение не затеняло другое, а только дополняло его и способствовало его росту, менеджер проекта терпеливо работает с каждым участником проектной команды, способствуя его правильному развитию, охраняя от внешних и внутренних потрясений, для того, чтобы максимально раскрыть его индивидуальные способности и увеличить отдачу от них, с удовлетворением отмечает каждое новое достижение.

«Что посеешь, то и пожнешь» - этот закон одинаково применим как к труду садовода, так и к труду менеджера проекта. Пренебрежительное отношение к людям породит лишь ответное пренебрежение. Вложите в людей часть своей души, и вам воздастся сторицей.