

PMsoop

Contents

Глава №0: Сначала была Идея!	2
Глава №1: А вот потом было слово	3
Глава №2: Поиск партнёра	4
Глава №3: Договорённости	6
Глава №3. Часть 1. Схемы работы	7
Глава №3. Часть 2. Milestones, Deliverables или кто кому что когда должен	11
Глава №3. Часть 3. Наконец, Контракт!	15
Глава №4: Явление Менеджера Проекта	16
Глава №5: МП знакомится с Заказчиком	18
Глава №5. Часть 1. Про общение.	19
Глава №5. Часть 2. Про выяснение требований, или "Необходимо"	20
Глава №5. Часть 3. Про сроки, или "Достаточно"	21
Глава №6: Собираем Команду Проекта	22
Глава №6. Часть 1. Формирование Команды	23
Глава №6. Часть 2. Первое собрание	25
Глава №7: Снова вопросы и ответы	26
Глава №8: Функциональная декомпозиция	30
Глава №9: Что и как будем тестировать	33
Глава №10: Сверка сроков	35
Глава №11: Определяем риски	38
Глава №12: Plan is nothing, Planning is everything!	41
Глава №13: Про среды	44
Глава №14: Старт разработки	51
Глава №15: Детальное планирование разработки итерации	53
Глава №16: Про тест-планы	54
Глава №16. Часть 1.	55
Глава №16. Часть 2.	58
Глава №17: Про ежедневный учёт и контроль	58
Глава №18: Про еженедельные собрания	61
Глава №19: Про информирование Заказчика	63

Глава №20: Подготовка к сдаче версии в тестирование	65
Глава №21: Тестирование и исправление дефектов	66
Глава №22: Подготовка к сдаче версии Заказчику	68
Глава №23: Обработка комментариев Заказчика	69
Глава №24: Changes менеджмент	71
Глава №25: Приёмка версии	75
Глава №26: Подготовка к работе над следующей версией	77
Глава №27: Про джедаев	79
Глава №28: Продолжаем в том же духе	82
Глава №29: Сдача финальной версии	83
Глава №30: Поддержка приложения	85
Глава №31: Завершение проекта – взгляд изнутри	87

Глава №0: Сначала была Идея!

В ролях - потенциальный Заказчик

Артефакты - нет

Суть - потенциальный Заказчик рождает Идею проекта

Идея - первична. На этом можно было бы и закончить, но...

А все начинается с того, что в некоторый неизвестный (нам) момент времени кому-то в голову приходит некоторая идея: "А что, если ...". Дальнейшие варианты различны:

- Сделать фирме сайт-визитку и увеличить продажи в X раз
- Оптимизировать надоевшую выписку справок посредством...
- Новый поисковик сделать лучше Гугла!
- Внедрить хитрую систему документооборота и поувольнять лишних сотрудников
- Сделать сайт для любимой собачки
- Завести себе вычурный (или не очень) блог
- Открыть порносайт и зарабатывать на рекламе
- ...

Сколько есть людей - столько есть идей, в общем-то.

Суть любой идеи в том, что её воплощение подразумевает получение некоторой выгоды, ясной автору. Это очень важный момент, который окажет огромное влияние на всю судьбу будущего проекта. Впрочем, не будем забегать вперед.

Глава №1: А вот потом было слово

В ролях - потенциальный Заказчик (возможно в нескольких лицах), который постепенно превращается в Заказчика

Артефакты - первичное описание задачи проекта

Суть - автор идеи (возможно с соавторами) формирует первичное описание задачи проекта

После того, как на свет появилась (пока только в голове у автора) новая идея, произойти может разное. 90% (на правах ИМХО) идей вскорости будут забыты, но нас они не интересуют (хотя сколько там, наверное, замечательных, блестящих идей которые могли бы сделать мир лучше... или угробить его окончательно).

В случае если идея автором не забыта - идея начинает обретать форму в словах и образах. В некоторых случаях автор обсуждает идею с коллегами/партнерами/друзьями/женой/итп. В некоторых случаях автор исследует альтернативы и конкурентные продукты. В некоторых случаях мечтает и придумывает все в одиночестве. По результату этого, безусловно, творческого процесса еще 90% (ИМХО снова) оставшихся идей так и не получают шанса на воплощение. Но займемся все-таки теми идеями, перспективность которых прошла эту тяжелую проверку.

Итак, по результату всех обсуждений и исследований автор формулирует свою идею обычно в виде некоторого текста, который далее будем называть первичным описанием задачи проекта.

Далее, в результате разнообразных процессов, которые мы не будем тут затрагивать, автор идеи (сам либо с соавторами/коллегами/инвесторами) определяет бюджет, который он (они) готовы потратить на воплощение идеи в жизнь. В процессе обсуждения идеи, особенно с инвесторами, в некоторых случаях первичное описание идеи обрастает различными подробностями, как функциональными, так и техническими, сроками, условиями платежей и т.п. Тем не менее, мы его называть будем все также - первичное описание задачи проекта (ПОЗП) - ибо это и есть первая информация о проекте, которая попадет в руки к Разработчику.

С этого момента лицо (совокупность лиц), заинтересованное в реализации описанной идеи в рамках имеющегося бюджета, будем смело называть Заказчиком.

Важным моментом является то, что обычно с точки зрения Заказчика первичное описание задачи проекта делает его идею понятной и очевидной любому читателю, а уж тем более потенциальному разработчику, буквально с первого прочтения. Типичная реальность будет раскрыта в следующих главах.

Глава №2: Поиск партнёра

В ролях - Заказчик, потенциальные разработчики (обычно персонализированные в виде менеджера по продажам компаний-разработчиков + специалиста отвечающего за оценку новых проектов)

Артефакты - первичное описание задачи проекта, первичные предложения от разработчиков

Суть - Заказчик представляет ПОЗП потенциальным разработчикам, которые анализируют и оценивают задачу, и формируют предложение по разработке. Заказчик выбирает конкретного разработчика для выполнения задачи

Итак, у Заказчика есть ПОЗП и некоторый бюджет. Следующий шаг - найти разработчика, который выполнит поставленную задачу в указанных условиях. Шаг этот очень важный, и очень не простой. Впрочем, обо всем по порядку.

Существуют множество разных способов, которыми Заказчики и Разработчики находят друг друга. Не все они просты и очевидны, некоторые вообще с трудом объяснимы современной наукой. Вот только некоторые наиболее типичные из них:

- Заказчик уже реализовал несколько проектов и обращается к проверенным разработчикам
- Заказчик обращается к разработчикам, которые ему посоветовали друзья/партнеры/конкуренты/доброжелатели/итп
- Заказчик объявляет тендер на разработку ПО
- Заказчик размещает информацию о проекте на сайтах-аукционах проектов типа elance.com, guru.com, freelancer и т.п.
- Заказчик использует Гул (или другой источник информации) и рассылает предложение о сотрудничестве тем разработчикам, которые ему понравились

Впрочем, с точки зрения управления проектами этот вопрос не суть важен, так что перейдем к более интересным нам темам.

Потенциальный разработчик получает от Заказчика первичное описание задачи проекта и представление о его бюджете. В исключительно редких случаях происходит следующее - разработчик сообщает Заказчику свою готовность реализовать поставленную задачу в описанных условиях и обозначает готовность начать работы через ХХ времени. Вероятность такого события равна вероятности того, что Заказчик напишет идеальное ТЗ и предложит к нему реальный бюджет и сроки, либо вероятности найти очень голодного/неопытного разработчика.

В 99% (имхо) остальных случаев задача проекта нуждается в разъяснениях и уточнениях, а бюджет (вместе со сроками) нуждается в длительном и аккуратном согласовании с выясненной задачей. Некоторые мысли о том, почему так случается:

- ПОЗП является очевидным и понятным с первого прочтения только для его автора, который "варится" в этой "теме" уже некоторое время. Посторонний человек, которым на данном этапе является представитель разработчика, ответственный за установление контактов с новыми Заказчиками, не воспринимает Идею проекта так же как её автор.
- Вообще, редкий человек в состоянии ясно излагать свои мысли хотя бы устно, и еще более редкий - письменно.
- Заказчик и Разработчик "говорят на разных языках". Я не имею в виду вариант, например англоязычного заказчика и русского разработчика (про это будет дальше), речь идет о профессиональной специфике. Часто проекты бывают связаны с профессиональной деятельностью заказчика, и редко бывает, чтобы разработчик хорошо знал область проекта.
- Разные культуры мышления и общения накладывают отпечаток на восприятие одних и тех же слов в тексте. Лингвисты меня поймут...
- Разные языки общения = трудности перевода.

Теперь рассмотрим, что и как обычно происходит на стороне Разработчика после получения первичной информации от Заказчика. По шагам:

1. Представитель разработчика, ответственный за установление контактов с новыми Заказчиками (далее - sales-manager, сейлз), изучает присланную документацию
2. В случае если задача проекта неочевидна - задаёт Заказчику уточняющие вопросы. Это продолжается до тех пор, пока сейлз не сформирует своё понимание задачи проекта, которое как кажется (сразу уточню - сейлзу и заказчику) соответствует пониманию заказчика
3. Далее сейлз отправляет уточнённое ПОЗП со своими комментариями на оценку ответственным лицам компании-разработчика (в некоторых случаях сейлз может оценить затраты на разработку проекта сам - все зависит от структуры компании и т.п.)
4. В случае необходимости оценщики задают дополнительные вопросы, которые сейлз выясняет у Заказчика
5. Результатом труда оценщиков является первичное предложение по реализации проекта (язык разработки, компоненты приложения, используемые готовые решения) и суммарная приблизительная оценка необходимых трудозатрат
6. На базе полученной информации сейлз готовит предложение по разработке проекта, в котором описывает предлагаемый вариант решения и обосновывает предлагаемую стоимость работ (например, указывает приблизительную оценку по компонентам и задачам в часах и приводит стоимость часа работ)
7. Готовое предложение отправляется Заказчику

В некоторых случаях история на этом не заканчивается, например:

8. Заказчик присылает вопросы - а почему так дорого выходит? а нельзя сделать не на ПХП на Java, он слышал, что так будет лучше?
9. Сейлз консультируется с оценщиками и отвечает
10. Заказчик предлагает - а давайте будем делать не все, что я сначала хотел, а вот тот кусок делать пока не будем?
11. Сейлз консультируется с оценщиками, корректирует оценку и отвечает

12. Заказчик спрашивает -

ну и так далее

По результату Заказчик получает некоторое количество предложений по разработке, из которых ему теперь предстоит выбрать того самого, единственного Разработчика, который воплотит его Идею в суровую материальную (информационную) форму. Эту сложную и иногда печальную (по статистике около 70% проектов не выходят дальше обсуждения предложений) фазу жизни будущего проекта описывать не буду.

Важным моментом на этапе "выбора партнёра" является понимание обеими сторонами что как ПОЗП, так и предложение по разработке не являются точными документами, им предстоит множество изменений, уточнений и других таинственных превращений, которые приведут и к изменениям бюджета, и к изменениям сроков, до того как проект выйдет в жизнь. Не стоит думать, что выбрав разработчика и согласившись заплатить сумму, указанную в предложении по разработке, Заказчик "уже почти купил" свою идею в материальной форме. Разработка ПО - это не товарное производство, а скорее опытно-конструкторское, поэтому обе стороны должны быть готовы к конструктивному сотрудничеству и принятию изменений, про которые будет рассказано в следующих главах.

Для дальнейшего рассмотрения примем, что Заказчик таки выбрал себе Разработчика и они вместе, веселые и полные оптимизма, готовятся к старту нового проекта.

Глава №3: Договорённости

В ролях - Заказчик, Разработчик (representative person)

Артефакты - ПОЗП, предложение на разработку, Контракт

Суть - Заказчик и Разработчик заключают Контракт на выполнение работ по проекту

Предварительные договорённости достигнуты, теперь самое время закрепить их в виде некоторого Договора либо Контракта.

Что такое Контракт? Если кратко - можно сказать, что это документ, описывающий ответственность сторон при выполнении некоторой работы, в нашем случае - разработке заказанного ПО.

В идеале Контракт должен четко формулировать поставленную задачу, предполагаемый процесс её решения и предполагаемые действия каждой стороны в процессе этого решения, как в случае успешного развития событий, так и в случае возможных неприятностей, задержек и форс-мажоров. Фактически Контракт должен являться руководством к действию и регламентов этих действий для всех участников процесса.

В реальности детальные контракты фактически не встречаются. Цель большинства реальных более-менее серьёзных контрактов - "прикрыть задницу в случае чего", т.е.

предусмотрительно подстраховаться от наиболее неприятных рисков. Это значит, что мало внимания уделяется стандартному регламенту (что делать, когда все хорошо), и много внимания уделяется ситуациям форс-мажорам и плохому поведению сторон ("что делать, когда все плохо").

Более того, в как минимум 50% (на правах имхо) случаев заказной разработки Контракт как серьёзный документ с подписями и печатями вообще отсутствует. Что с моей точки зрения вполне логично для небольших проектов и случаев когда обе стороны кажутся друг другу достаточно адекватными для решения возможных споров в процессе работ без предварительных договорённостей и юридической ответственности. В таких случаях роль Контракта выполняют договорённости о порядке оплаты (например, зафиксированные на сайте-аукционе типа elance.com) и документы, написанные в ходе проекта.

Учитывая реальное положение дел, попробую сформулировать тот необходимый минимум вещей, о которых все-таки следует договориться до начала работ даже для случая маленьких проектов и фрилансерской разработки. Важные аспекты предварительных договорённостей будут описаны по частям.

Глава №3. Часть 1. Схемы работы

В первую очередь это порядок и условия оплаты выполненных работ. Т.е. за что, сколько и когда платит Заказчик. Рассмотрим две наиболее типичные схемы:

- **fixed scope & fixed price.** Эта схема взаимодействия предполагает фиксированную плату за выполнение фиксированного объема работ. Объем работ оценивается разработчиком на основании представленного заказчиком ТЗ. Оплата происходит по факту сдачи готового продукта целиком или по частям по мере сдачи продукта "частями".
- **time & material.** Эта схема взаимодействия предполагает оплату времени и материалов, потраченных Разработчиком на реализацию проекта, на регулярной основе (раз в неделю, раз в месяц и т.п.) вплоть до полного окончания работ. Объем работ жестко не зафиксирован (хотя представление об объеме работ всегда есть) и может меняться в ходе проекта по взаимному согласию сторон.

Основное отличие описанных схем, как нетрудно заметить, заключается в фиксации объема работ. Рассмотрим их более детально.

Fixed scope & fixed price

Эта схема предполагает фиксацию определенного объема работ. Объем работ рассчитывается разработчиком на основании задачи сформулированной заказчиком. Задача обычно формулируется в виде одного или нескольких документов различной степени детализации.

Однако, учитывая сложности понимания между заказчиком и разработчиком (которые были описаны в предыдущих главах, и будут раскрываться глубже в последующих) следует понимать, что рассчитать точный объем работ на этапе старта проекта можно только в случае

наличия идеального и полного ТЗ и только в случае весьма высокой квалификации Разработчика. В реальности идеальное ТЗ не встречается, и квалификация людей, которые работают с проектной документацией на этапе старта, редко позволяет дать 100% точные оценки. Редкое ТЗ будет во всех случаях одинаково понято и Заказчиком и Разработчиком, а как известно разночтения и уточнения ведут в основном к увеличению объема работ и срыву сроков. Редкая оценка на этапе старта проекта будет отличаться от финального результата меньше чем на 20%.

Для наглядности можно рассмотреть проекты в других областях - например, ремонт квартиры. Кто-нибудь знает хоть одного человека, который сделал ремонт 100% уложившись в первоначальную смету и выдержав поставленные сроки? При этом учтем, что ремонты в своих жилищах люди делают уже не первую тысячу лет, а разработке ПО от силы лет 30.

Поэтому следует признать, что в случае такой схемы работы Разработчик вынужден обещать то, в чем не может быть уверен. Отсюда следует нелюбовь разработчиков к жестким фиксированным контрактам. Для компенсации рисков неточной оценки и разночтения документации при фиксированных схемах разработчик старается давать максимальные оценки трудозатрат и делать "закладки" в оценках, которые могут составлять от +10% до +50% к суммарно оцененным трудозатратам (в зависимости от опыта реализации подобных проектов и "жесткости" контракта). Причем, как показывает опыт законченных проектов, и этого может оказаться мало. Также при фиксированной схеме работы Разработчик склонен завышать стоимость часа (или другой единицы) работ в сравнении со схемой time & material.

В то же время, заказчикам такая схема работы обычно нравится - так как она даёт (мнимую) гарантию "вписаться в бюджет" и наиболее похожа на привычную в обычной жизни схему "деньги - товар".

При реализации проекта по фиксированной схеме оплаты Разработчик работает преимущественно с требованиями, и обращается непосредственно к Заказчику только в случае необходимости их уточнения. То есть, Заказчик мало вовлечен в процесс разработки, его участие предполагается в основном на этапах приёмки выполненного продукта. Замечу, что это даёт Разработчику, в общем-то, большую свободу действий по сравнению с time & material схемами.

Отдельную проблему для фиксированных схем представляют изменения, которые вносит Заказчик в процессе разработки по мере ознакомления с готовыми частями продукта. Изменения к требованиям - вполне нормальное явление, которое присутствует в подавляющем большинстве проектов. Проблемой они становятся тогда, когда Заказчик говорит что "это же написано в первоначальном ТЗ, просто вы не так поняли!". В фиксированной схеме работы у Заказчика часто изначально присутствует уверенность, что ВСЬ проект должен быть выполнен за оговоренную сумму, что вызывает внутренний дискомфорт в ответ на аргументированные просьбы Разработчика о дополнительной оплате изменений. Этот внутренний дискомфорт является дополнительным мотивом для сентенций в духе "это же написано в первоначальном ТЗ, просто вы не так поняли!". По моему опыту конфликтные ситуации, связанные с изменениями требований (а значит и изменением бюджета) в проектах, ведущихся по фиксированной схеме, возникают фактически всегда.

После сделанного краткого обзора некоторых важных моментов попробуем просто перечислить плюсы и минусы схемы fixed price & fixed score для Заказчика и Разработчика:

- плюсы для Заказчика: максимально точная оценка бюджета и сроков до начала работ, малая вовлеченность в проект (мало личной ответственности), чувство уверенности на этапе старта проекта
- минусы для Заказчика: бюджет со скрытыми "перезакладами" (шанс заплатить больше чем надо), малая вовлеченность в проект (меньше знаешь что происходит, меньше рычагов влияния), сложность согласования изменений
- плюсы для Разработчика: большая свобода в принятии решений (ограничена только требованиями, нет непосредственного контроля Заказчика), возможность "перезаложиться" в бюджете и "сыграть в плюс", большая цена часа работ
- минусы для Разработчика: высокие риски "попасть" на неоплаченные трудозатраты вследствие неверной интерпретации требований и неточных оценок, необходимость "учесть все" на этапе предварительных договорённостей, до начала работ, которая ведет к большим трудозатратам на оценки и анализ

Исходя из данного понимания, можно рекомендовать схему fixed price & fixed score для следующих случаев и/или их совокупностей:

- Заказчик некомпетентен в сфере ИТ, формулирует требования со стороны бизнеса и оставляет высокую свободу реализации интерфейса (дизайн, usability) Разработчику, а также мало вмешивается в проект во время разработки
- Заказчику крайне важно вложиться в заданный бюджет при этом он готов ограничить функционал, следуя рекомендациям Разработчика
- Требования Заказчика сформулированы максимально чётко, Заказчик не заинтересован в каких-либо изменениях по ходу проекта
- Разработчик имеет большой опыт реализации аналогичных (функционал, технологии) проектов и уверен в точности оценок
- Разработчик привлекает к проекту исключительно высококлассных специалистов (что понятным образом отражается на цене проекта) и "прописывает" в контракте чёткую и детальную схему обработки запросов на изменения требований

Фиксированных схем работы следует категорически избегать в следующих случаях:

- Заказчик понимает, что требования еще не до конца сформулированы и в процессе реализации проекта могут появляться многочисленные изменения
- Заказчик является специалистом в сфере разработки ПО и хочет иметь полный контроль над разработкой и участвовать в принятии технических решений
- Разработчик привлекает к проекту специалистов среднего и начального уровня (что может понизить цену проекта)
- Требования написаны откровенно плохо, и ни у Заказчика, ни у Разработчика нет ресурсов для их "приведения в порядок"

При работе по фиксированной схеме максимум внимания на этапе подготовки к старту проекта надо уделить детальному анализу требований и проработке архитектуры проекта для уточнения оценок.

Time & Material

Эта схема предполагает оплату Заказчиком трудозатрат Разработчика на регулярной основе и непосредственное вовлечение Заказчика в процесс разработки. При этом на этапе подготовки к старту проекта все равно делаются приблизительные оценки бюджета проекта и сроков его реализации, но обе стороны признают приблизительность этих оценок и собственную ответственность за их изменения.

Одним из крайних вариантов работы по такой схеме является вариант "body-shop", в котором Заказчик арендует персонал у фирмы-Разработчика и самостоятельно управляет всем ходом разработки, при этом фирма-Разработчик несет ответственность за уровень квалификации персонала и организацию рабочих мест. Более типичным вариантом является разделение обязанностей таким образом, что Заказчик отвечает за функциональные и интерфейсные решения (что надо сделать и как оно должно быть организовано), а Разработчик фокусируется на технических аспектах (как именно мы это реализуем и какие компоненты будут использованы).

Типичными проблемами, возникающими при такой схеме работы, являются:

- "Размытие" зон ответственности - стороны увлекаются, забывают зоны своей ответственности, Разработчик начинает придумывать функционал, а Заказчик даёт советы программистам как писать код - в результате обычно происходят странные и непонятные конфликты
- "Размытие" бюджета - Заказчик увлекается изменениями и "ударяется в перфекционизм", в результате бюджет может значительно превысить первоначальную оценку, а в некоторых случаях даже "неожиданно закончиться"
- Постоянные личные контакты между Заказчиком и Разработчиком персонализируют конфликты - в рабочих моментах могут появляться элементы личных отношений, которые мешают нормальной работе

Рассмотрим общие плюсы и минусы этой схемы для Заказчика и Разработчика:

- плюсы для Заказчика: возможность лично контролировать ход разработки, большая информированность о ходе работ и возможность увидеть промежуточные результаты, легкость в изменении требований и вводе дополнительного функционала, обычно стоимость часа ниже, чем в fixed схеме
- минусы для Заказчика: слабая определенность общего бюджета и сроков проекта, личная ответственность за результаты (может рассматриваться как +), возможность "увлечься"
- плюсы для Разработчика: часть ответственности за результат несёт Заказчик (в случае проблем "всё свалить на Разработчика" не получится), регулярность платежей, возможность привлечь к проекту специалистов среднего и начального уровня
- минусы для Разработчика: меньше свободы в принятии решений, часто необходимость обучать Заказчика в процессе разработки, слабый уровень документации - большая часть общения происходит в письмах и устно

Схема Time & Material рекомендуется в следующих случаях:

- Заказчик является специалистом в области разработки ПО и хочет контролировать разработку
- Заказчик понимает "сырость" требований и возможность множества изменений и готов вовлекаться в процесс разработки для уточнения требований по мере необходимости

Не рекомендуется применять эту схему, в случае если Заказчик не может уделять достаточно времени проекту - в таком случае лучше послать к заказчику аналитика, написать качественные требования и идти по fixed схеме.

При подготовке старта проекта по схеме time & material особое внимание нужно уделить разделению ответственности между Заказчиком и Разработчиком.

Глава №3. Часть 2. Milestones, Deliverables или кто кому что когда должен

Вторым важным аспектом, о котором нужно договариваться до старта работ по проекту является последовательность, порядок и сроки передачи артефактов между Заказчиком и Разработчиком.

Во всех проектах в процессе или по окончании работ Разработчик передаёт Заказчику результаты своего труда. Также во всех проектах Заказчик отвечает за своевременное предоставление ТЗ и другой необходимой информации. Кроме того, в некоторых проектах Заказчик должен передать Разработчику важные для разработки артефакты, такие как, например дизайн сайта, код предыдущего проекта, инструментарию разработки, спецификации доступа к рабочей среде проекта и т.п. О таких артефактах и пойдет речь.

Сразу замечу, что задержки в предоставлении артефактов являются типичной причиной срыва установленных сроков. Собственно, сроки сдачи обычно привязаны к передаче определенных артефактов (результатов работ) от Разработчика к Заказчику. При этом не всегда и не все понимают, что несвоевременное предоставление ТЗ и уточнений от Заказчика, а тем более таких вещей как дизайн, компоненты разработки и спецификации среды на этих самых сроках отражается самым непосредственным образом. Тут важно понимать, что изучение информации и артефактов, предоставленных Заказчиком, обычно требует некоторого времени на стороне Разработчика. В процессе изучения могут возникнуть вопросы, на выяснение которых требуется дополнительное время. Грамотные Разработчики при планировании сроков сдачи проекта выделяют на это определенные "закладки" времени. В случае срыва сроков передачи информации/артефактов Заказчиком времени на детальное изучение и выяснение вопросов может не хватить. Некоторые Разработчики ради сохранения общих сроков сдачи "урежают" это время. В результате информация изучается недостаточным образом, что ведет к росту количества дефектов на этапе сдачи проекта. Таким образом, срыв сроков поставки Заказчиком на пару дней элементарно может привести к срыву сроков сдачи проекта на неделю.

Поэтому важно:

- ещё на этапе подготовки к старту проекта определить списков всех (важных) артефактов которые подлежат передаче между сторонами в процессе разработки

- для каждого артефакта определить планируемые сроки передачи и, желательно, способ передачи (почта, заливка на указанный ФТП, и т.п.)
- чётко определить зависимости между артефактами и прописать в Контракте (либо заменяющих его документах) последствия сдвига сроков (например, задержка в предоставлении дизайна ведет к аналогичной задержке при сдаче проекта)
- - при планировании сроков предусмотреть время необходимое на изучение представленной информации/артефактов и выяснение возможных вопросов

Все это вполне логично может быть увязано с планированием ключевых "вех" проекта или milestones.

Типичной частью Контракта (или заменяющих его документов) является сводный план проекта в виде списка ключевых "вех" проекта - milestones. Для схем типа fixed price & fixed scope в этом же плане к "вехам" "привязываются платежи. Для схем типа time & material "вехи" проекта служат для определения сроков завершения всего проекта и расчета приблизительного бюджета проекта. Приведу примеры такого сводного плана:

Пример 1

Milestone	Milestone Description	Milestone Acceptance	Payment €
Development start	All the conditions are discussed, the set of functions is described, and Proposal for Development (PFD) is written and approved. Prepayment is completed.	Milestone is accepted as soon as payment is made. After this development is started	
Stage1 Version End	Features from 4. Project Cost and Duration that are marked as Stage1 () are ready and work on Contractor's side. Payment for this stage is completed.	Customer has 3 working days after the delivery to test Stage1 version. After this period payment has to be made	
Stage2 Version End	Features from 4. Project Cost and Duration that are marked as Stage2 () are ready and work on Contractor's side. Payment for this stage is completed.	Customer has 3 working days after the delivery to test Stage2 version. After this period payment has to be made	
Stage3 Version End	Features from 4. Project Cost and Duration that are marked as Stage3 () are ready and work on Contractor's side. Payment for this stage is completed.	Customer has 3 working days after the delivery to test Stage2 version. After this period payment has to be made	
Acceptance finish	All functionality is deployed and tested on Customer's side. Customer has approved and accepted functionality.	Customer has 5 working days to test Final version. Contractor fixed all the bugs which were found during 5 working days since Final version delivery. After this period	

		payment has to be made.	
--	--	-------------------------	--

Пример 2:

Milestone	Patch Delivery Date	Payment Ammount	Payment Date	Description
Kick-off		\$3075 (20%)	Week 0	Prepayment
Cross-cutting functionality – logging	Week 3	\$2306 (15%)	Week 4	Code review passed, QA passed with acceptable defect ratio
Single-Sign on (SSO) Login	Week 4	\$2306 (15%)	Week 5	Code review passed, QA passed with acceptable defect ratio
Configuration Page	Week 7	\$2306 (15%)	Week 8	Code review passed, QA passed with acceptable defect ratio
Just-in-time account creation and updates	Week 7	\$2307 (15%)	Week 8	Code review passed, QA passed with acceptable defect ratio
Code Complete	Week 9	\$3075 (20%)	Week 10	Overall project scope accepted, functional and QA passed, all revealed defects are either fixed or agreed as NTBF

Как можно заметить, в приведенных примерах присутствуют:

- название "вехи"/milestone
- краткое описание, которое позволяет понять, что собственно является критерием завершения "вехи"
- планируемый срок сдачи "вехи"
- сумма платежа по сдаче "вехи" (оба примера из проектов по fixed схемам)
- максимальный срок проведения платежа

Создание сводного плана "вех" проекта - milestones planning - и есть суть всего этапа подготовки к заключению Контракта (либо заменяющих его...) и подготовки к старту проекта.

Отдельно хочу остановиться на таком понятии как "критерии сдачи". Чёткое и ясное определение критериев сдачи/приёмки для каждой "вехи" (milestone) проекта является одним из очень важных моментов для успешного завершения проекта. Пожалуй, даже не менее важным, чем полнота, ясность и непротиворечивость требований.

В общем случае, критерии сдачи/приёмки обычно определяются, как ссылки на соответствующие разделы ТЗ и указание что этот раздел должен быть выполнен. Однако сразу замечу, что этого не достаточно. Кроме определения "что должно быть сделано" большой важностью обладают определения "как это будет представлено" и "процедура проверки". И еще один момент - типичной ситуацией является "забывчивость" сторон на момент подготовки к старту проекта про возможные дефекты. Разберем все четыре аспекта подробнее:

1. "что должно быть сделано". В общем случае может быть ссылкой на определенные разделы ТЗ. Однако вспомним простую истину - "ТЗ не бывает идеальным!". Ошибки в интерпретации и неоднозначности ТЗ ведут "размытию" критериев приёмки, конфликтам при сдаче и срыву сроков. Кроме этого, в ТЗ в большинстве случаев описывают преимущественно функциональные требования, а нефункциональные требования (например, производительность) приводят отдельным пунктом, на который критерий приёмки уже не ссылается.
2. "как это будет представлено". Предположим ситуацию, когда на этапе подготовки к старту проекта вопрос "в какой среде будем показывать" вообще не обсуждается. В процессе работы выясняется, что Заказчик хочет все проверять в своей среде. При этом среда Заказчика может весьма отличаться от среды разработки и "подгонка" приложения к среде Заказчика потребует дополнительных незапланированных трудозатрат. Другие типичные ситуации - Заказчику может быть неудобно проверять проект в среде представленной Разработчиком; предполагаемая среда показа не позволяет проверить некоторые требования (например нефункциональные); среда в которой происходит "показ" этапов проекта значительно отличается от production environment, т.е. среды, в которой будет "жить" проект после выхода в live, поэтому то, что все время работало, вдруг перестаёт работать.
3. "процедура проверки". В некоторых случаях на этапе подготовки к старту проекта вообще не рассматриваются, например такие вопросы как "сколько времени нужно заказчику на проверку полученной версии?" и "а как Заказчик будет представлять Разработчику свой feedback?". В результате такой невнимательности Заказчик может всё еще принимать версию 1, когда уже будет готова версия 2, а свои замечания писать Разработчику в ICQ требуя срочно все исправить.
4. "забыли про дефекты". Часто бывает, что при приёмке определенного этапа проекта Заказчик требует устранить все, в том числе косметические дефекты. С точки зрения разработки это не всегда логично, иногда удобнее фиксировать мелкие дефекты уже на следующем этапе разработки.

После такого печального обзора "типичных ляпов" сформулирую важные рекомендации касательно критериев сдачи/приёмки:

- всегда планируйте "вехи" с учетом реализации нефункциональных требований
- если ТЗ не позволяет чётко определить, что именно должно быть готово - либо доработайте ТЗ, либо приводите списки готового функционала к каждой "вехе", либо запланируете уточнение требований и критериев приёмки в процессе разработки (и "пропишите" это в Контракте рядом с планированием "вех"!), либо, наконец, отдельно впишите в Контракт что "если интерпретация требований позволяет двусмысленное толкование - то соответствие любому из возможных толкований является достаточным условием для приёмки"
- обязательно соберите и зафиксируйте в Контракте либо заменяющих его документах планируемые среды разработки, показа и "жизни" (live) приложения, а также какая сторона (Заказчик или Разработчик) отвечает за готовность конкретной среды обеспечить сдачу/приёмку в нужный момент времени, и, соответственно, ответственность указанной стороны за возможный срыв сроков по причине неготовности среды.

- для каждой "вехи" проекта и версии приложения, которая подлежит сдаче/приёмке, определите сроки, в течение которых должны быть завершены все приёмочные/сдаточные активности. Кроме того, желательно перечислить такие активности в явном виде.
- обязательно определите, в какой форме принимаются замечания и пожелания по доработке/устранению дефектов. Рекомендуется использование специализированных issues tracking систем.
- наиболее благоприятным вариантом обработки дефектов при сдаче промежуточных версий является следующий: все обнаруженные дефекты делятся на критические (которые мешают завершить проверку либо влияют на корректную работу основного функционала) и прочие; критические исправляются сразу по факту обнаружения, прочие дефекты исправляются во время следующего этапа; критичность дефекта определяется по взаимному согласию Заказчика и Разработчика; замечания Заказчика, которые ведут к изменению требований, обрабатываются вообще отдельным от дефектов образом, для чего выделяется дополнительное время на оценку изменений и что может повлечь изменение бюджета проекта.
- в любом случае достигнутую договорённость про обработку замечаний Заказчика при приёмке версии надо фиксировать в максимально чётком виде в Контракте либо заменяющих его документах.

Пока всё. Многие из описанных в этой части моментов будут раскрываться более подробно в следующих главах.

Глава №3. Часть 3. Наконец, Контракт!

Теперь попробую сделать то, что обещал еще в начале главы - свести воедино основные рекомендации на тему "что должно быть договорено и записано" вне зависимости от размера проекта и наличия собственно Контракта как юридического документа, либо работы через сайт-аукцион. Итак:

- краткая суть проекта и ссылка на ТЗ
- контакты обеих сторон и желаемые сроки ответа на запрос с каждой стороны
- степень вовлеченности Заказчика в процесс разработки - если Заказчик отвечает не только за ТЗ и приёмку, то это лучше указать
- выбранная схема работы - fixed или time & material - и условия оплаты (когда, каким образом)
- сводный план проекта в виде "вех"/milestones и сроков
- перечень артефактов передаваемых в процессе разработки с указанием кто за что и когда отвечает, желательно со сроками и привязкой к "вехам"
- перечень сред в проекте с указанием кто за что отвечает
- критерии приёмки для каждой "вехи" и версии приложения, желательно также указать событие после которого версия считается приятной (хотя бы email от Заказчика, где написано "версию принял", для юридических контрактов - акты приёмки сдачи с подписями сторон)
- порядок обработки дефектов и запросов на изменения

- действия в случае форс-мажоров (как минимум предусмотреть порядок остановки проекта в случае экстренных событий)
- порядок арбитража спорных моментов (для юридических контрактов - это дело юристов, для сайтов-аукционов - часто автоматически подразумевается использование собственной системы арбитража, для других случаев - лучше договориться заранее)

Если вы ничего не забыли и все договорено и прописано - может быть в 500-страничном Контракте с подписями Президентов Компаний, а может быть и email'ах - главное чтобы обе стороны принимали действительность этих договорённостей - можно стартовать проект.

Помните, что излишняя торопливость на этапе договоренностей и досадная забывчивость может привести к срывам сроков на месяцы и годы, финансовых потерях в размере больше первоначального бюджета проекта и многим другим неприятностям.

Также хочу обратить внимание на одинаковую важность описанных аспектов и моментов предварительных договоренностей, как для Заказчика, так и для Разработчика. Если вы Заказчик, и рассчитываете "нагибать" Разработчика в процессе разработки благодаря неточностям в контракте и при этом получить то, что заказываете - вряд ли у вас что-то выйдет...

Глава №4: Явление Менеджера Проекта

В ролях - Заказчик, Менеджер Проекта

Артефакты - ПОЗП, предложение на разработку, Контракт

Суть - Менеджер Проекта вникает в суть Проекта

Заказчик и Разработчик уже понравились друг другу, договорились о начале работ и заключили определенные договорённости (Контракт). Именно в этот момент в большинстве проектов происходит назначение, пожалуй, ключевой фигуры проекта - Менеджера Проекта.

Сразу замечу, что в некоторых случаях Менеджер Проекта начинает работу с Заказчиком еще на этапе pre-sale вместе с сейлзом и отвечать за оценки трудозатрат; в некоторых случаях Менеджер Проекта (МП) может исполнять роль сейлза; в некоторых случаях Менеджер Проекта может быть назначен на более поздних этапах - например, в случаях, когда с Заказчиком работает Бизнес Аналитик, который отвечает за написание качественных требований, МП может быть назначен на этапе сбора команды проекта (следующая глава).

В любом проекте, не только в области разработки ПО, можно выделить минимум три уровня работы с требованиями:

- уровень идеи, ответственный - Заказчик, признаки - на этом уровне думают преимущественно про бизнес и выгоду, задача уровня - получать ответ на вопрос "что надо сделать?", типичный артефакт - Vision

- уровень логики и декомпозиции, ответственный - Менеджер Проектов либо Бизнес Аналитик, признаки - на этом уровне выясняют и уточняют логику/принцип воплощения идеи, задача уровня - ответить на вопрос "как оно будет работать?", типичные артефакты - Use Case(s), Use Scenarios, SRS (спорно - в некоторых случаях может принадлежать и к следующему уровню)
- технический уровень, ответственный - архитектор, Lead Developer, признаки - описание технических инструментов и платформ, классов и интерфейсов, ну и т.п., задача уровня - ответ на вопрос "как именно мы это сделаем", типичные артефакты - описание архитектуры проекта, прототипы, диаграммы классов

Более подробно про уровни можно посмотреть например [здесь](#).

Так как я пытаюсь рассмотреть "случай типичного заказного проекта", и описать процесс максимально простым и понятным для непосвященных образом, примем, что именно Менеджер Проекта отвечает за разбор, анализ и уточнение требований при старте проекта - т.е. за второй уровень.

Итак, Менеджер Проектов получил назначение на новый проект. Основные задачи МП в проекте:

- Координация обмена информацией и артефактами между Заказчиком и командой проекта

- Определение, расстановка приоритетов и постановка задач команде проекта

- "Настройка" работы команды проекта, как на профессиональном, так и на личном уровне

Конкретные обязанности и действия следуют из данных определений и будут раскрыты в следующих главах более подробно.

Получив назначение, Менеджер Проекта в первую очередь должен изучить всю доступную информацию о требованиях проекта и Заказчике проекта - рекомендуется не только прочитать текущий документ с ТЗ и Контракт, но и организовать интервью с сейлзом проекта, почитать (если сохранилась - а сохранять её стоит) переписку сейлза с заказчиком, прочитать первые версии описания проекта и ТЗ если они менялись в процессе подготовки предложения по разработке и заключению Контракта.

Все это следует сделать для того, чтобы максимально полно понять бизнес-идею проекта и образ мыслей Заказчика.

Понимание бизнес-идеи проекта Менеджером Проекта является одним из ключевых факторов успеха проекта. Чем ближе понимание бизнес-идеи МП к пониманию Заказчика - тем выше шансы проекта на успех, и наоборот. Малое внимание к бизнес-идее есть типичная ошибка многих начинающих МП, особенно "выросших" из "технарей" - начинающие МП любят концентрироваться на вопросах "как", оставляя в стороне вопросы "зачем". В результате может получиться что-то работающее, но не нужное...

Понимание образа мыслей Заказчика сильно облегчает взаимопонимание между МП и Заказчиком. Учитывая, что МП фактически является "транслятором" информации между

заказчиком и командой проекта - важность этого аспекта не стоит недооценивать. И, кстати, можно вспомнить, что хороший МП должен быть "слегка психологом" - это тоже к области взаимопонимания.

Далее. Внимательно и дотошно изучив всю информацию о проекте и достигнув определенного понимания главной бизнес-идеи проекта, а также других потребностей Заказчика, которые предстоит реализовать в будущем проекте, Менеджер Проекта формирует первичное понимание декомпозиции проекта на отдельные задачи и последовательности оных задач.

Написал сложно, а по сути - когда все прочитаешь и в голове оно более-менее уляжется - на уровне размышлений уже появляется некоторое понимание что в этом проекте главное, а что не очень, какие требования можно реализовать отдельно от прочих, а какие связаны между собой.

Это первичное понимание я рекомендую зафиксировать, хотя бы в виде набросков на листке - для закрепления оно в сознании и для сохранения в анналах проекта - после окончания проекта будет полезно вспомнить с чего все начиналось и проанализировать ход событий. Также следует понимать, что это понимание будет являться базой для первичного планирования проекта.

Все вопросы, которые возникли (и не пропали) у Менеджера Проекта в процессе изучения информации по проекту следует записать для дальнейшего выяснения у Заказчика (об этом речь в следующей главе).

Все, теперь Менеджер Проекта готов знакомиться с Заказчиком лично и собирать Команду Проекта для старта работ, о чём читайте далее.

И еще раз повторю самый важный момент этой главы: Менеджер Проекта должен понимать бизнес-идею проекта максимально близко к пониманию Заказчика. Рекомендую обратить внимание на этот момент как Разработчикам (на уровне супервизора проекта), так и Заказчикам (приложить максимум усилий для объяснения и задать контрольные вопросы).

Глава №5: МП знакомится с Заказчиком

в 3х частях

В ролях - Заказчик, Менеджер Проекта, может быть Бизнес Аналитик

Артефакты - ПОЗП, предложение на разработку, Контракт

Суть - Менеджер Проекта начинает общение с Заказчиком проекта

После изучения всей доступной информации по проекту и осознания первичного понимания задач проекта, их приоритетов и возможных шагов по реализации оно - самое время Менеджеру Проекта познакомиться с Заказчиком и в процессе общения проверить соответствие своего понимания потребностям Заказчика.

Собственно, познакомиться с Заказчиком МП мог и ранее - в зависимости от принятого процесса и сложившихся обстоятельств. Суть данной главы в том, что Менеджер Проекта должен, просто обязан, до начала планирования работ и до объяснения задач проекта команде проекта синхронизировать своё понимание/видение данного проекта с Заказчиком. Ибо, как я уже отмечал ранее, синхронность/одинаковость понимания сути проекта, его задач и приоритетов этих задач между Менеджером Проекта и Заказчиком является фактически ключевым аспектом успешности проекта. Неправильное понимание МП задач проекта или их приоритетов приводит к ошибкам при постановке конкретных задач по реализации функционала разработчикам проекта, и в результате элементарным образом получается не то, что было нужно, не тогда, когда нужно, не так как нужно или что-то вообще ненужное. Теперь переходим к рассмотрению действий по существу.

Глава №5. Часть 1. Про общение.

Как именно знакомиться с Заказчиком? Можно начать, например таким письмом:

"Здравствуйте Иван Иванович,

Меня зовут Вася Пупкин, я менеджер проектов в компании ХХХ и мне поручено руководство вашим проектом "создание новой мега-системы". Сообщите пожалуйста, когда вам будет удобно ответить на несколько вопросов.

С надеждой на взаимовыгодное сотрудничество,

Вася Пупкин"

Можно позвонить, можно написать Заказчику в скайп или аську - всё зависит от способов общения, предпочитаемых Заказчиком (которые можно выяснить у сейлза, например), возможностей (в аутсорсинге личные встречи не всегда получаются) и фантазии Менеджера Проектов. Общая рекомендация - чем ближе общение к "живому", тем легче и быстрее оно происходит. Способы общения по приоритетам:

1. Встреча вживую с использованием наглядных материалов (доска + маркеры = гениальное изобретение)
2. Встреча вживую
3. Видеоконференция
4. Аудиоконференция (звонок по телефону или скайпу, скайп лучше т.к. простым образом записывает разговоры)
5. Чат
6. Переписка по email

Важная рекомендация по общению с Заказчиком вообще: по возможности следует записывать само общение и обязательно нужно записывать результаты, достигнутые в процессе общения. Данная рекомендация действительна и для Заказчиков. Логи общения - мощный инструмент против "забывчивости" сторон, которая часто является очень неприятным фактором на финальных этапах проекта и при обсуждении изменений требований. Кстати, про логи общения и сохранённую переписку я уже упоминал ранее в

контексте общения сейлз - Заказчик, это очень полезная информация. Так что, инструменты для записи общения и его результатов следует готовить заранее. Для наглядности представлю основные инструменты:

1. При личных встречах - если обе стороны не против, желательно использовать диктофон.
2. Использовали ли вы диктофон или нет - по завершению встречи крайне рекомендуется написать "meeting minutes" и разослать всем участникам. Что такое meeting minutes хорошо знает гугл.
3. При видеоконференциях - есть масса ПО для захвата и записи видеопотока. Meeting minutes тоже никто не отменял.
4. Аудиоконференции - аналогично. И снова - meeting minutes наше все!
5. Чаты - храним историю переписки!
6. Email - достаточно не удалять письма до окончания проекта (а лучше и после - сложить в архив и пусть живет, а то вдруг продолжение или переделки)

Все таки раскрою краткую суть meeting minutes: это некоторый текст, обычно электрическое письмо типа email, в котором перечислены участники встречи, основные темы которые обсуждались, принятые решения (ключевая часть!) и намеченные задачи на будущее. Meeting minutes следует писать "по горячим следам", и отправлять решительно всем участникам встречи. Часто ответственным за написание meeting minutes в ходе проекта является Менеджер Проекта, и этой ответственность ни в коем случае не стоит пренебрегать.

Теперь предположим, что Менеджер Проекта качественно подготовился, представился Заказчику и готов задавать упомянутые в примере вопросы. Какие же вопросы следует задать Заказчику в первую очередь? В первую очередь стоит задать наиболее важные вопросы из тех, что возникли у МП в процессе изучения информации по проекту. Важность вопроса можно определить по его отношению к тому функционалу, который кажется наиболее значимым в проекте.

Глава №5. Часть 2. Про выяснение требований, или "Необходимо"

Отступление: В идеальном проекте всю работу по выяснению и уточнению требований выполняет отдельная роль - Бизнес Аналитик, задачей которого является формирование полного и ясного ТЗ. В реальности идеальных полных и ясных ТЗ не встречается (если у кого есть - присылайте :-)), и часто Менеджер Проекта, даже при наличии в проекте бизнес аналитика, все равно выполняет некоторые задачи анализа, будучи вынужденным уточнять требования по ходу проекта. Кроме того, в заказной (а тем более аутсорсинговой) разработке для проектов небольшого (определяется в разных компания по-разному) масштаба выделение бизнес аналитика на проект может являться нежелательной тратой ограниченных ресурсов, так что все задачи анализа выполняются МП.

Идем дальше. Сначала задаём самые важные вопросы. По мере получения ответов - заодно понимаем насколько понимание наиболее значимого функционала соответствует пониманию Заказчика. В некоторых случаях Менеджера Проекта могут ожидать удивительные открытия.

Вопросы задаём до того момента, пока не возникнет ощущения что значимые задачи обоими сторонами выделены и оцениваются одинаковым образом. После чего остальные, более "мелкие" вопросы можно даже оставить на потом - для этапа обсуждения планирования итераций и технического воплощения. Вопрос, например, точного расположения кнопки "отправить" на странице обратной связи обычно попадает именно в такую категорию, при условии, что страница обратной связи не является ключевым функционалом проекта.

Чёткие рекомендации по определению важности предоставить трудно, главное о чем следует помнить - Заказчик обычно склонен мыслить на уровне бизнес-идей, и "спуск" на технические уровни Заказчика может весьма утомить, либо наоборот, вызвать прилив не очень желательного энтузиазма. Многие начинающие Заказчики любят лично расставлять кнопочки и определять их цвета, обращая на это чуть ли не больше внимания, чем на основной функционал. Задача Менеджера Проектов - "удержать" Заказчика на уровне его компетентности и при этом выяснить все действительно важные аспекты проекта.

При выяснении вопросов у Заказчика рекомендуется группировать вопросы сообразно тематике и задачам проекта, которых они касаются. Последовательное "забрасывание" заказчика разными вопросами, касающимися разных задач проекта, быстро утомляет как Заказчика, так и вопросителя, а также усложняет разбор полученных ответов. Лучше заранее подготовить несколько серий вопросов, и задавать их отдельными блоками (тем более часто бывает, что несколько вопросов могут быть взаимосвязаны). Между сериями вопросов следует делать перерывы (размер перерыва зависит от времени, которое можно потратить на обсуждение вопросов вообще, и времени которое занимает обсуждение одной серии вопросов) - во время таких перерывов информация "укладывается в головы", и могут возникать новые мысли касательно уже состоявшегося обсуждения.

Пожалуй, на этом краткий обзор начала общения Менеджера Проекта с Заказчиком и рекомендации "что делать" пока закончу. Любопытный читатель для получения более полной информации может обратиться к источникам по Бизнес Анализу и Requirements Facilitation.

По результату всех описанных действий (кто бы их не выполнял - МП или бизнес-аналитик) Менеджер Проекта должен обрести уверенность в том, что его понимание задач проекта и их приоритетов аналогично пониманию Заказчика. Еще раз повторюсь - "одинаковость" понимания задач (бизнес-идеи) проекта между МП и Заказчиком это ключевой фактор успешности проекта. Наличие в проекте бизнес аналитика обычно помогает достижению "одинаковости", но его не гарантирует. Менеджер Проекта должен уделить особое внимание этому вопросу самостоятельно, используя все доступные средства, в том числе и бизнес аналитика (если он есть).

Глава №5. Часть 3. Про сроки, или "Достаточно"

Бывает, что время на обсуждение и выяснение вопросов ограничено. Например, есть установленный срок сдачи первой версии проекта, и руководство "давит" чтобы начать разработку как можно раньше. Бывает, что Заказчик устал от вопросов, или через 2 дня срочно уезжает в командировку. Вообще, часто бывает, что выяснить все важные вопросы до старта собственно разработки не представляется возможным. Это, в общем, нормально.

В таких случаях необходимо до старта разработки выяснить всего два основных момента.

Первым моментом является понимание бизнес-идеи проекта. Без чёткого понимания, каким образом Заказчик получит выгоду от реализации данного проекта, стартовать разработку категорически не рекомендуется.

Вторым моментом является понимание наиболее важного функционала, который можно реализовать в первой фазе проекта. Этот момент непосредственно связан с планированием проекта, поэтому в этой главе будет освещен кратко, но в последующих главах раскрыт в подробностях.

Как известно, в подавляющем большинстве случаев функционал проекта реализуется не весь и сразу, а в некоторой последовательности, определенной планом проекта. С точки зрения требований (ТЗ), отдельные элементы функционала определяются отдельными задачами проекта, которые в свою очередь определяются общей бизнес-идеей проекта. При планировании реализации функционала логично и рационально в первую очередь реализовать функционал, воплощающий наиболее важную задачу проекта. Потому что чем раньше Заказчик сможет "потрогать" основной "механизм" своей идеи - тем раньше Разработчик получит либо утверждение, либо изменения основного функционала. Переделки основного функционала в конце проекта штука очень неприятная, поэтому лучше основной функционал реализовать и утвердить в первую очередь.

Таким образом, для реализации второго момента нужно понять, что именно (какой ключевой функционал) возможно воплотить в первую очередь и затем проверить, и договориться об этом с Заказчиком. Говоря другими словами - определить и прояснить, а затем утвердить скоуп первой итерации проекта. При этом принимается, что вопросы касательно прочего функционала можно будет выяснить во время разработки первой итерации.

Подробнее про итерации, итерационную модель разработки, распределение функционала по итерациям и т.п. читайте в следующих главах.

Глава №6: Собираем Команду Проекта

в 2х частях

В ролях - Заказчик, Менеджер Проекта, Команда Проекта

Артефакты - ПОЗП, предложение на разработку, Контракт

Суть - Менеджер Проекта собирает Команду Проекта

Пришло время познакомиться с командой проекта - простыми (и не очень) разработчиками, тестировщиками, дизайнерами, админами и прочими причастными.

Глава №6. Часть 1. Формирование Команды

Команда формируется обычно Менеджером Проекта по согласованию с начальниками отдела, отделом кадров, менеджерами других проектов и руководством компании. Происходит это в несколько этапов.

В первую очередь стоит определить роли, необходимые в данном проекте, и планируемую нагрузку по каждой роли. При этом следует понимать, что роли и люди - это разные понятия. В любом проекте, в котором необходимо хотя бы минимальное тестирование, необходима роль тестировщика. При этом в реальности роль тестировщика может временно выполнять менеджер проекта или один из разработчиков. В любом проекте по разработке ПО всегда присутствует роль "разработчик", и человек, выполняющий эту роль, часто попутно выполняет другие роли, не предполагающие постоянной существенной нагрузки в данном проекте.

Попробую представить краткий список ролей часто встречающихся при разработке заказного ПО:

1. менеджер проекта. уже рассматривалась
2. бизнес аналитик. основные задачи - выяснение и уточнение требований, поддержание рабочих документов по требованиям в актуальном состоянии. может совмещаться с МП
3. системный архитектор. основная задача - разработка архитектуры данного проекта и определение основных используемых компонентов. может совмещаться с ролями "начальник группы разработки", "ведущий разработчик" и "разработчик"
4. начальник группы разработки. основная задача - руководство группой разработчиков (обычно группы формируются либо по технологии, либо по ответственности за определенный функционал), самостоятельное принятие важных технических решений в рамках своей группы (в соответствии с архитектурой проекта). может совмещаться с ролями "разработчик" и "ведущий разработчик"
5. ведущий разработчик. основные задачи - написание кода, помощь в решении технических проблем другим разработчикам, анализ кода других разработчиков.
6. разработчик. основная задача - написание кода.
7. начальник группы тестирования. Основные задачи - руководство группой тестирования, поддержание в актуальном состоянии сценариев тестирования и их согласование с другими участниками проекта. Может совмещаться с ролями "ведущий тестировщик", иногда "бизнес аналитик"
8. ведущий тестировщик. основные задачи - выполнение тестов, написание тестов, помощь другим тестировщикам
9. тестировщик. выполняет тесты
10. системный администратор. Основные задачи - настройка и поддержка всех необходимых сред проекта, обеспечение "заливки" кода из одной среды в другую. Эта роль может совмещаться с ролью "ведущий разработчик"

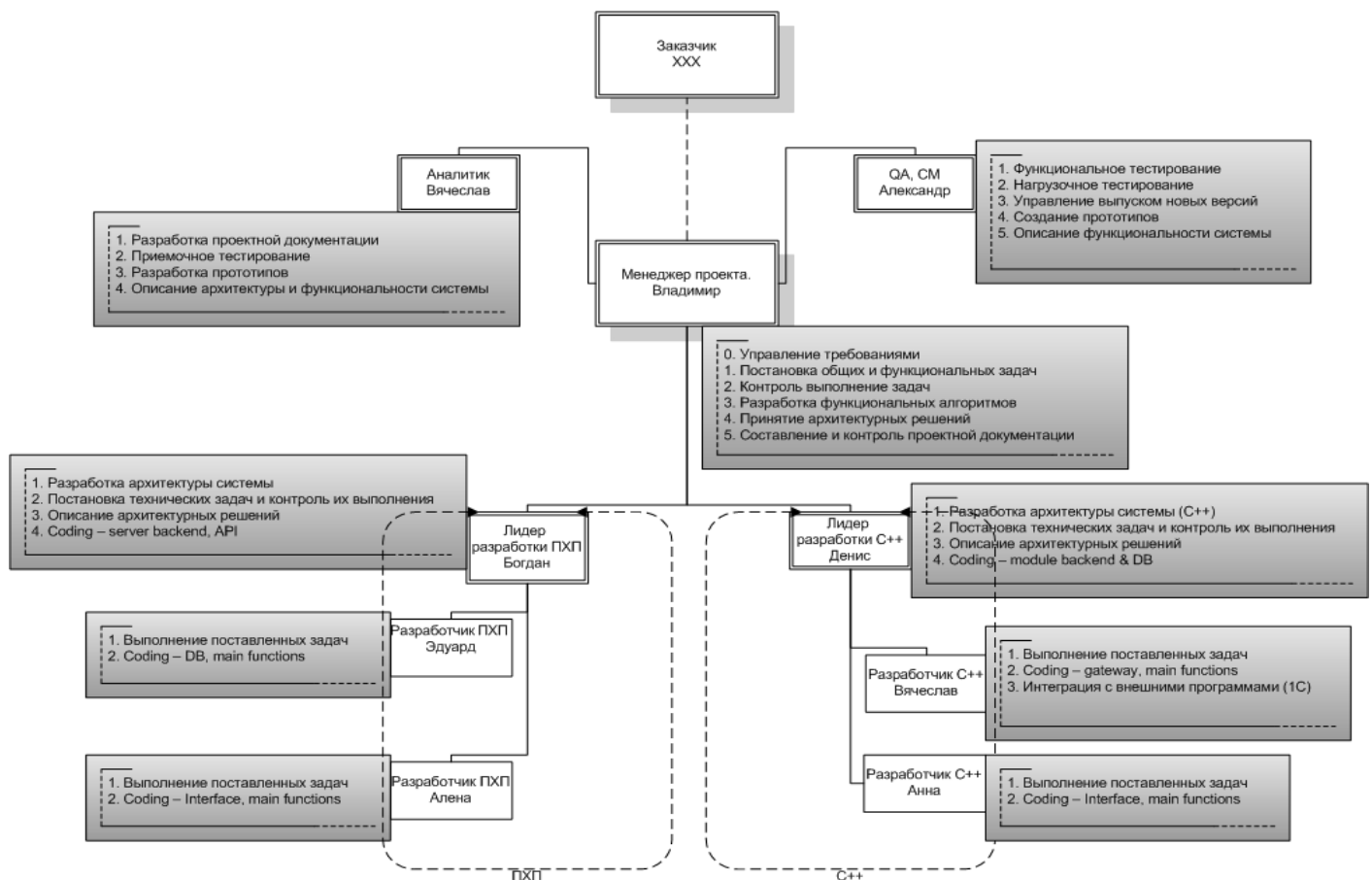
Это только краткий список, и рассмотрены далеко не все варианты совмещения ролей. В малых проектах (фриланс) все роли может выполнять один и тот же человек, включая сюда и сейлза тоже. Однако следует помнить, что делать качественно всё и сразу умеют только

гении. Чем больше масштаб проекта - тем большая нужна квалификация для каждой роли и тем больше негативные последствия совмещения нескольких ролей.

Определение, какие роли могут быть совмещены, а какие совмещать не стоит, и определение необходимой квалификации персонала для выполнения определённой роли в проекте обычно выполняются руководством компании - Разработчика при участии МП в соответствии с бюджетом проекта, ценностью Заказчика для компании, свободными ресурсами компании и множеством других факторов. Подбор конкретных разработчиков под проект также относится к этой таинственной области. Следует понимать, что в разных компаниях это происходит по-разному. Также следует понимать, что часто 3 человека с большим опытом и высокой квалификацией могут заменить полноценную команду из 10 "новичков" без потери качества разработки. Рекомендаций в этой области могу дать ровно две:

- Рекомендация Заказчику: вы либо доверяете компании-разработчику, и значит принимаете предложенный состав команды, либо не доверяете - но тогда разработку лучше сразу прекращать.
- Рекомендация Разработчику: если вам важны хорошие отношения с Заказчиком - не стоит экономить на команде; "студентов" лучше натаскивать на доделках старых проектов или повторных проектов проверенных заказчиков, а не на новых заказчиках и технически сложных проектах.

За сим считаем, что размер команды, и кто какие роли будет выполнять, уже определили. А раз определили - рекомендую Менеджеру Проекта определённую схему сразу зарисовать, можно на бумажке, а можно, например в Visio, и ознакомить с ней всех участников проекта. Вид зарисовки может быть разным, начиная от табличного в виде "Вася = разработчик + иногда тестировщик; Петя = архитектор + разработчик", до схем с определением полномочий, например как в следующем примере:



Нарисовав, распечатав и повесив в доступном всем участникам проекта месте организационную схему, Менеджер Проекта может облегчить себе жизнь за счёт улучшения осознания своих обязанностей каждым членом команды проекта :-). Естественно следует понимать, что если вы создали такую оргсхему - то в случае изменений состава команды в течение проекта её надо будет своевременно обновлять.

Глава №6. Часть 2. Первое собрание

После того, как Менеджер Проекта "утряс" вопрос состава команды проекта с руководством, и приблизительно распределил, кто какие роли и обязанности выполняет - самое время собрать всех участников вместе и рассказать им про проект, его задачи, и что кому надо будет сделать для их выполнения.

Первую встречу команды можно (и, по моему мнению, нужно) проводить до ознакомления участников с документацией. Встречу организует и проводит Менеджер Проекта. Основными задачами первой встречи являются:

1. Объявить участникам о новом проекте, в котором им предстоит работать
2. Кратко объяснить бизнес-идею и задачи проекта на необходимом и доступном уровне
3. Объявить и обсудить предлагаемый вариант распределения ролей и обязанностей в проекте (по результатам задуманная менеджером оргсхема может измениться)
4. Раздать документацию по проекту для внимательного изучения

5. Назначить время проведения следующего собрания для обсуждения возникших при прочтении документации вопросов и предварительного планирования проекта

На всякий случай перечислю основные рекомендации по проведению собраний вообще:

1. Участники собрания должны узнать об оном заранее, лучше хотя бы за день - чтобы все успели прочитать агенду и спланировать своё время. Предупреждать о собрании за 10 минут до начала - дурной тон. Использование "напоминалок" типа outlook appointment - хороший тон
2. Рассылая участникам собрания приглашение обязательно стоит включить в него агенду собрания - перечень основных тем, которые планируется обсудить. Это помогает людям правильно настроиться и, возможно, подготовить какие-то вопросы
3. В процессе проведения собрания следует соблюдать правило "один оратор в эфире!" - т.е. не допускать "галдежа"
4. Собрание должен вести от начала и до конца один человек. Он же следит за порядком высказываний и порядком вообще
5. Желательно дать высказаться каждому, кому есть что сказать по существу
6. Все принятые решения (в т.ч. о следующей встрече) нужно зафиксировать. Meeting minutes - очень полезная штука. Если "все свои" - хотя бы еще раз огласите все решения в конце собрания
7. Важно проконтролировать, что если кому-то что-то поручено, то он это запомнил. Опять же о пользе meeting minutes

В целом, Менеджеру Проекта стоит обладать определенными навыками проведения собраний и контроля аудитории. Навыки эти зависят от характера и развиваются весьма разнообразными способами - так что про это писать более подробно не буду. Интересующимся - в Гул, потом на тренинги :-)

По результатам первого собрания все участники должны понять, что надо сделать вообще в целом, и какую роль в этом предстоит играть лично каждому, а также озадачиться внимательным изучением проектной документации за отведенный и утвержденный срок.

Кроме этого важным моментом является задание положительного и конструктивного настроения в команде проекта начиная с первого собрания. Как именно это сделать - пока выходит за рамки данного текста т.к. пока с трудом поддается классификации и алгоритмизации. Могу только сказать, что вера команды в проект и его нужность/важность, а также определенные профессиональные/статусные/материальные стимулы для определенных членов команды за участие в проекте могут быть весьма полезны. Интересующимся - снова в Гул по ключевому слову "мотивация".

Глава №7: Снова вопросы и ответы

В ролях - Заказчик, Менеджер Проекта, Команда Проекта

Артефакты - ПОЗП->ТЗ

Суть - Команда проекта уточняет требования проекта, по результату ПОЗП превращается в ТЗ

Примем, что всем участникам проекта объявлено об их участии в проекте, приблизительно распределены роли и роздана на изучение документация по проекту, а также объявлен срок, до которого эту самую документацию надо изучить. С этого момента можем считать, что подготовительный этап проекта стартован; суть подготовительного этапа будет раскрыта в этой и трёх последующих главах.

Во время изучения проектной документации с большой вероятностью у разработчиков (тестировщиков и прочих участников проекта) возникнут новые вопросы. Вопросы, возникшие при прочтении документации у Менеджера Проекта, обычно отличаются от вопросов, возникающих у разработчиков и тестировщиков. Каждый участник проекта "смотрит со своей колокольни" - это нормально и естественно. Поэтому при раздаче проектной документации на изучение Менеджеру Проекта стоит попросить всех участников записывать все возникшие при изучении вопросы, и представить свой список вопросов на следующем собрании, которое назовём "проверка требований".

Кроме вопросов при изучении проектной документации у разработчиков сформируется некоторое первичное представление о последовательности воплощения функционала проекта и основных задачах разработки, а также возможном использовании различных готовых компонентов и средств разработки. Это понимание также стоит зафиксировать в виде некоторых "набросков", хотя бы и на бумаге - Менеджеру Проекта стоит обратить внимание разработчиков на этот момент. Всякие полезные мысли могут возникать не только у разработчиков, но и у тестировщиков, и других участников проекта - поэтому попросить записывать всякие полезные мысли стоит всех.

В общем, при постановке задачи "изучить проектную документацию" следует:

- определить сроки на изучение и назначить дату и место следующего собрания
- обязательно попросить записать все возникшие вопросы и подготовиться к их представлению на следующем собрании
- желательно попросить подготовить и представить на следующем собрании свои предложения касательно разработки/тестирования/... проекта, если таковые появятся

Теперь переходим собственно к следующему (за изучением проектной документации) собранию. Его основные цели:

1. Собрать вопросы, возникшие у команды проекта в процессе изучения документации, и найти ответы на них
2. Выяснить готовность документации для начала проектирования архитектуры и планирования работ по проекту
3. Проверить, что все участники команды проекта одинаковым и правильным образом поняли требования

Дополнительными целями собрания могут являться:

4. Обсудить возможные варианты реализации проекта. Выбрать оптимальный вариант реализации с учётом всех условий и "набросать" концепцию архитектуры
5. Выделить основные задачи по реализации функционала и определить их последовательность
6. Наметить основные этапы и типы тестирования
7. Определить необходимые трудозатраты для выделенных задач и сверить сроки сдачи версий
8. Определить основные риски проекта
9. Зафиксировать все принятые решения и сделанные оценки и перейти к составлению плана проекта

Однако реализовать все обозначенные цели на одном собрании можно только для весьма небольших проектов с небольшими командами разработки, так что реализацию целей 4 - 9 я последовательно рассмотрю в главах 8, 9 и 10.

Перейдём к более подробному рассмотрению целей 1 - 3:

Цель 1. Собрать вопросы, возникшие у команды проекта в процессе изучения документации, и найти ответы на них

Реализуется во время подготовки к собранию, собственно на самом собрании и после него. На "установочном" первом собрании или при раздаче требований по проекту для их изучения рекомендуется попросить всех участников возникшие у них вопросы записывать и за некоторое время (одного дня обычно хватает) до собрания по "проверке требований" прислать свой список вопросов Менеджеру Проекта.

Менеджер Проекта с помощью бизнес-аналитика (если таковой присутствует в проекте) должен до собрания отсортировать присланные вопросы на следующие три группы:

- вопросы, требующие срочного выяснения ответов у Заказчика. Такие вопросы обычно связаны с неоднозначностью и/или противоречивостью ответов, без получения ясных ответов на такие вопросы перейти к функциональной декомпозиции и планированию проекта невозможно.
- вопросы, требующие ответов Заказчика, но не сильно срочно. Например, уточнение реализации определённой функции или расположение элементов интерфейса.
- вопросы, ответы на которые можно дать самостоятельно исходя из уже имеющейся информации по проекту.

Понятно, что при анализе и сортировке вопросов дубликаты следует удалять, а вопросы, вызванные неверным пониманием требований - выделить отдельно с целью последующего разъяснения на собрании (вообще они относятся к третьей группе). В случае, если некоторые вопросы из первой группы сильно влияют на ясность понимания требований - собрание по проверке требований лучше отложить и немедленно перейти к выяснению таких вопросов. Процедура выяснения вопросов описана чуть ниже.

На самом собрании по проверке требований стоит разобрать с командой в первую очередь вопросы из первой группы. Нужно определить критичность этих вопросов и их потенциальное влияние на следующие действия подготовительного этапа, выяснить насколько задержка в получении ответов может отразиться на установленных в контакте сроках сдачи проекта. Затем следует разобрать вопросы третьей группы и провести необходимые разъяснения до достижения полного понимания всему участникам команды. После чего "пройтись" по вопросам третьей группы. В процессе обсуждения вопросов некоторые из них могут сменить группу, это нормально.

После собрания (или до него, если вопросы особо критичные) Менеджер Проекта должен выяснить у Заказчика ответы на вопросы первой группы. Так как задержки получения ответов часто сильно влияют на сроки старта и, соответственно, окончания разработки - выяснять их лучше вживую или на аудиоконференции (возможно с участием членов команды проекта ответственных за область вопроса). Влияние ответов (и сроков их получения) на сроки проекта стоит объяснить Заказчику. Хорошим вариантом может являться участие (вживую либо видео/аудиоконференция) Заказчика на собрании по проверке требований - тогда все вопросы можно будет выяснить прямо на собрании.

Проверка достижения цели - наличие вопросов, разбитых на обозначенные группы; критически важные вопросы заданы Заказчику

Цель 2. Выяснить готовность документации для начала проектирования архитектуры и планирования работ по проекту

Когда получены ответы на критические вопросы (первая группа) - можно озадачить Заказчика разбором вопросов второй группы, которые можно задать уже в виде email, но обязательно с указанием желательного срока ответа. Затем нужно обновить требования проекта (ПОЗП) с учётом полученных ответов на критические вопросы. Наличие требований проекта, в которых нет явных и критичных для планирования разработки "дыр" является важным фактором для успешности проекта. Информация должна быть собрана в одном месте, доступном всем участникам - для этого и документируют требования в виде определённого документа (Proposal For Development, SRS, etc), и именно этот документ нужно будет обновить. Делает это бизнес-аналитик, либо Менеджер Проекта (если нет бизнес-аналитика). Полученный документ, в котором собраны требования по проекту и не обнаружено (на данном этапе) критических недостатков отныне будем называть просто ТЗ.

Отдельно замечу, что ТЗ крайне редко является "неизменным" документом - обычно его еще предстоит обновлять, уточнять и изменять в ходе всего проекта и, особенно на этапах между итерациями, после сдачи очередной версии при детальном планировании работ по следующей версии. Это - нормально, так и должно быть, т.к. "видение" финального приложения у Заказчика обычно меняется по ходу визуализации/реализации продукта. На подготовительном этапе при реализации цели №1 задачей Менеджера Проекта и всей команды является получение некоторой "отправной точки", в которой информации достаточно для перехода к планированию реализации проекта. Критерий достаточной ясности требований - субъективное дело Менеджера Проекта и участников команды, и зависит от их профессионализма и опыта в разработке проектов подобной направленности.

Проверка достижения цели - наличие ТЗ, достаточного для перехода к планированию реализации с точки зрения всех участников команды проекта.

Цель 3. Проверить, что все участники команды проекта одинаковым и правильным образом поняли требования

Реализуется на собрании по проверке требований. Ответственный за реализацию - Менеджер Проекта. Анализируя вопросы, заданные участниками команды проекта и задавая свои дополнительные вопросы, а также разъясняя спорные и неясные моменты Менеджер Проекта должен добиться одинакового и ясного понимания бизнес-идеи проекта и его основных задач (основного функционала, как это должно работать) всеми участниками команды.

Проверка достижения цели - субъективное ощущение Менеджера Проектов + отсутствие спорных моментов при обсуждении требований в команде + утверждение командой достаточности ТЗ.

Глава №8: Функциональная декомпозиция

В ролях - Менеджер Проекта, Команда Проекта (разработчики, старшие разработчики, руководители разработки)

Артефакты - ТЗ, feature-list, task-list

Суть - Команда проекта выбирает оптимальный вариант технической реализации и определяет основные задачи разработки

В этой главе продолжается описание подготовительного (к старту) этапа проекта и рассматриваются следующие цели:

Цель 4. Обсудить возможные варианты реализации проекта. Выбрать оптимальный вариант реализации с учётом всех условий и "набросать" концепцию архитектуры

Цель 5. Выделить основные задачи по реализации функционала и определить их последовательность

Для достижения этих целей необходимо иметь достаточно ясное ТЗ (критерий ясности объяснён в предыдущей главе) и провести одно или несколько собраний с разработчиками, посвящённых обсуждению вариантов технической реализации проекта. Начнём по порядку:

Цель 4. Обсудить возможные варианты реализации проекта. Выбрать оптимальный вариант реализации с учётом всех условий и "набросать" концепцию архитектуры

Сначала приведу список основных моментов, которые должны быть определены для определения конкретного варианта реализации проекта:

- Основные функциональные элементы. Для проекта типа "нужно чтобы пользователи удобным образом могли из данных, хранящихся на их ПК, формировать контент представленный на некотором сайте" элементами могут быть приложение, исполняемое на ПК пользователя + сайт + API взаимодействия между ними. Для проекта типа "video-chat" элементами могут быть сайт (PHP or Java) + "движок" передачи видео интегрированный в сайт (Flash/Flex + FMS). Обычно элементы выделяют по принадлежности к разным технологиям и разным средам исполнения.
- Язык разработки для каждого компонента (C++, PHP, Java, Flash/Flex, etc)
- Платформа разработки для каждого компонента (Windows, Linux etc.)
- Используемые готовые решения - библиотеки, сторонние приложения, фреймворки и т.п.
- Компоненты среды разработки - что должно стоять на ПК разработчика, чтобы вести разработку
- Компоненты среды работы готового продукта - что должно стоять на сервере/ПК, на котором будет работать готовый продукт

Список можно расширить и дополнить - в этом вам помогут разработчики и эксперты. В разных проектах некоторые или даже все (для случаев доработки чужих проектов) описанные моменты могут быть определены Заказчиком. Те моменты, которые не определены - необходимо определить исходя из ТЗ проекта и прочей имеющейся информации.

Обычно определение этих моментов происходит простым образом – разработчики, исходя из ТЗ, предлагают возможные варианты, далее на собрании Менеджер Проекта обсуждает предложенные варианты и выбирает вариант с минимальными трудозатратами и/или рисками. Таким образом, для реализации цели МП необходимо:

- озадачить разработчиков предложением возможных вариантов реализации проекта
- провести собрание с разработчиками и выбрать оптимальный вариант
- выбранный вариант следует некоторым образом зафиксировать, достаточно зарисовки общей схемы на листике с перечислением используемых элементов

Проверка достижения цели - некоторый артефакт (листик с рисунками, схема в Visio, документ) который представляет выбранную схему реализации проекта.

Цель 5. Выделить основные задачи по реализации функционала и определить их последовательность

Непосредственно после определения варианта реализации проекта можно начинать функциональную декомпозицию проекта на высоком уровне - выделить основные элементарные задачи по реализации функционала проекта и определить связи между такими задачами, определяющие последовательность их выполнения. По результату необходимо получить список технических задач, которые можно поручить отдельным разработчикам (либо группам разработчиков), записанных в порядке их желаемого выполнения; при этом подразумевается, что выполнение всех задач по списку приведет к реализации всего функционала проекта согласно ТЗ. Как это сделать?

Для начала вспомним про такой артефакт проекта как feature-list (фиче-лист). Данный артефакт обычно создаётся на этапе приблизительной оценки трудозатрат по проекту, предшествующем подготовке предложения по разработке. Фактически речь идет о списке выделенных из ПОЗП основных функций будущего приложения. Если фиче-лист есть - для начала стоит его актуализировать в соответствии с текущим ТЗ (обычно ТЗ весьма отличается от первичного описания). Если фиче-листа нет - Менеджер Проекта может легко его получить из ТЗ, возможно с помощью бизнес-аналитика.

Далее, имея актуальный фиче-лист, Менеджер Проекта на собрании с разработчиками проводит его анализ, по результату которого получается необходимый список задач. Делается это также простым образом. В "идеале", реализация отдельной функции приложения подразумевает отдельную задачу для разработчика(ов). В реальности нужно учесть выбранный вариант реализации проекта. Для реализации некоторых функций может потребоваться выполнение нескольких технических задач, в некоторых случаях даже связанных с разными технологиями. Некоторые другие функции могут быть реализованы совместно в рамках выполнения одной технической задачи. Некоторые технические задачи для своего решения требуют предварительной реализации других задач - что и определяет последовательность разработки. Все эти моменты Менеджеру Проекта любезно покажут разработчики, если задавать им правильные вопросы :-)

Кроме того, на последовательность выполнения технических задач непосредственно влияет утверждённый в Контракте план "вех" проекта и сдачи готовых версий. Если по какой-то причине этого плана ещё нет - у Менеджера Проекта появляется большая свобода в выборе удобного и наименее рискованного пути реализации проекта. Но чаще всего такой план уже есть, так что последовательность реализации функций из ТЗ определяется не только удобством разработки, но и этим планом.

Также последовательность выполнения технических задач может определяться необходимостью "как можно раньше реализовать и проверить самые "узкие" и рискованные технические решения", например задачи обеспечивающие быстроедействие проекта или связанные с использованием нестабильных сторонних компонент (библиотек и фреймворков в бета-версиях, например).

Таким образом, необходимые действия выглядят так:

1. берем актуальный фиче-лист
2. вместе с разработчиками выделяем технические задачи и их взаимосвязи
3. определяем "узкие места" и стараемся вынести соответствующие задачи на ранние этапы разработки
4. "подгоняем" полученное под план сдачи версий проекта, утвержденный в контракте
5. составляем список задач, учитывая последовательность их выполнения
6. для каждой задачи в списке подписываем технологию реализации / кто из разработчиков за неё будет отвечать / какая роль за неё ответственна из орг. схемы проекта

При выполнении этих действий не рекомендуется увлекаться детализацией функциональных и технических задач - время для этого придет немного позже, при составлении плана проекта.

Отдельную функцию при составлении фиче-листа можно определить как отдельную операцию, выполнение которой не связано с выполнением других операций. Например, форма обратной связи на сайте - это обычно одна функция в фиче-листе, потому что заполнение формы и каждого её элемента не связано с другим функционалом сайта (сложные формы - отдельный разговор). Отдельную техническую задачу можно определить таким же образом - как задачу, выполняемую одним и тем же разработчиком(ками), не требующую в процессе выполнения реализации других технических задач.

Готовый список технических задач будем называть task-list (таск-лист). Впрочем, это еще не окончательный вид таск-листа, он будет дополняться в следующей главе.

Проверка достижения цели №5 - наличие таск-листа, одобренного разработчиками и Менеджером Проекта.

Глава №9: Что и как будем тестировать

В ролях - Менеджер Проекта, тестировщики, разработчики

Артефакты - ТЗ, task-list

Суть – Менеджер Проекта вместе с тестировщиками и разработчиками добавляет в таск-лист задачи тестирования

В этой главе рассмотрим следующую цель подготовительного этапа:

Цель 6. Наметить основные этапы и типы тестирования

Для реализации цели понадобится таск-лист, согласованный с разработчиками. Менеджер Проекта собирает тестировщиков, и они вместе определяют на каких этапах проекта понадобится тестирование, и какое именно тестирование понадобится на этих этапах. После чего выделенные задачи тестирования добавляются в таск-лист.

Для начала кратко рассмотрим основные типы тестирования, применяемые для большинства проектов:

- функциональное тестирование (functional testing) - сверка реализованных функций с их поведением, описанным в ТЗ
- тестирование безопасности (security testing) - поиск способов "сломать" приложение, вводя нестандартные данные, используя скрипты и т.п.
- тестирование интерфейса (usability testing) - проверка удобства работы с графическим интерфейсом, поиск оптимизаций
- нагрузочное тестирование (performance testing) - проверка работы приложения под нагрузкой, определённой в нефункциональных требованиях ТЗ
- регрессионное тестирование (regression testing) - проверка работы функционала, реализованного в версиях 1 - N при тестировании версии N+1 (иногда новый код может "поломать" работу старого)

Вообще, типов тестирования выделяют более 20. Подробнее о них можно узнать в специальной литературе, Гул в помощь. Какие именно типы тестирования могут понадобиться в конкретном проекте, Менеджер Проекта может определить с экспертом в области тестирования, если таковые есть в компании-Разработчике, до проведения собрания с тестировщиками.

Теперь перейдём к собранию с тестировщиками. Начать следует с определения, какие типы тестирования понадобятся для каждой версии, которая будет сдаваться Заказчику. Как минимум, в каждой версии надо проверять новый функционал, реализованный в данной версии, а также работу функционала реализованного ранее. Если дизайн интерфейса разрабатывается самостоятельно - стоит провести usability testing.

Затем следует определить возможность тестирования промежуточных версий, получаемых в процессе разработки, которые не показываются Заказчику. В целом, чем раньше можно протестировать реализованную функцию - тем проще исправить возможные дефекты. Если реализация некоторой функции является необходимой для реализации следующих связанных функций - стоит протестировать её до перехода к реализации связанных функций. Возможность тестирования промежуточных версий стоит согласовать с разработчиками.

Особое внимание надо обратить на нагрузочное тестирование - его следует проводить как можно раньше. Низкое быстродействие приложения может потребовать серьёзных изменений в архитектуре, которые проще провести вначале проекта.

Также особое внимание стоит обратить на тестирование всех "узких" мест с точки зрения разработки, таких как использование нестабильных версий сторонних приложений и библиотек, работа с технологиями, неизвестными ранее разработчикам и т.п. Устранение возможных дефектов, связанных с использованием сторонних компонент и новых технологий, также может потребовать изменений в архитектуре приложения, которые проще проводить на начальных этапах проекта.

По результату обсуждений необходимо выделить и добавить в таск-лист задачи тестирования, такие как, например "функциональное тестирование 1й версии", "нагрузочное тестирование 2й версии" и т.п. Для каждой задачи, добавленной в таск-лист, нужно указать тип тестирования и роль, ответственную за данную задачу.

На собрание с тестировщиками, посвященное выделению задач тестирования, в некоторых случаях имеет смысл приглашать и разработчиков проекта. Обсуждение возможности тестирования промежуточных версий, а также возможных сроков проведения нагрузочного тестирования и тестирования "узких мест", требует согласования с задачами разработки.

При выделении задач тестирования не стоит углубляться в детали реализации – как именно будет выполняться тестирование можно и нужно будет определить потом. Условие элементарности задачи можно сформулировать так: в рамках задачи выполняется тестирование одного типа, при этом выполнение задачи может быть проведено одним и тем же тестировщиком (или группой тестировщиков) без перехода к другим задачам.

Проверка достижения цели - задачи тестирования добавлены в таск-лист, порядок выполнения задач разработки и тестирования согласован и с разработчиками и с тестировщиками.

Глава №10: Сверка сроков

В ролях - Заказчик, Менеджер Проекта, Команда Проекта

Артефакты - ТЗ, план сдачи версий

Суть - Команда переоценивает трудозатраты по таск-листу, Менеджер Проекта проверяет сроки и согласовывает их с Заказчиком

В рамках этой главы рассмотрим реализацию следующей цели:

Цель 7. Определить необходимые трудозатраты для выделенных задач и сверить сроки сдачи версий

Теперь у нас есть таск-лист, в котором содержатся все определённые задачи разработки и тестирования проекта, записанные в последовательности их реализации, и согласованный с планом сдачи версий проекта. Следующим шагом подготовительного этапа будет переоценка трудозатрат по задачам таск-листа, перерасчёт сроков сдачи версий и согласование уточнённого плана сдачи версий с Заказчиком.

Менеджер Проекта раздаёт/рассылает таск-лист всем участникам проекта, с просьбой провести оценку трудозатрат на выполнение каждой задачи, и назначает следующее собрание для обсуждения и согласования сделанных оценок.

С одной стороны, оценку трудозатрат для выполнения конкретной задачи логично поручить тому(тем) членам команды, которые обладают наибольшим опытом и квалификацией в технологической области самой задачи. С другой стороны, желательно чтобы оценку трудозатрат проводил тот разработчик или тестировщик, который будем выполнять данную задачу. Рассмотрим ситуацию на примере:

1. в проекте участвуют три разработчика РНР, один из которых обладает квалификацией старшего разработчика и в проекте играет роль ведущего разработчика
2. в проекте есть некоторое количество задач разработки на РНР, которые некоторым образом распределены между всеми разработчиками
3. каждый разработчик оценивает те задачи, которые ему предстоит выполнять
4. когда все задачи РНР оценены – ведущий разработчик сверяет сделанные оценки и подтверждает их корректность

В случае, когда разработчики или тестировщики не уверены в своей оценке – рекомендуется обратиться к эксперту в данной области, если таковой есть в компании-Разработчике. Получасовая консультация на этом этапе может «спасти» несколько дней разработки после страта проекта.

Оценки рекомендуется ставить в виде диапазона, например 8 – 12 часов, где 8 – это ожидаемый срок выполнения задачи в оптимальном варианте, а 12 – в случае «если что-то пойдет не так». Оценки в таком виде дают Менеджеру Проекта лучшее понимание уверенности участников проектной команды в своих силах и возможных крайних сроках при переоценке сроков сдачи версий.

Касательно уверенности – оценка типа 8 – 24 часа является типичным признаком возможного технологического риска или слабой квалификации разработчика в данной технологии. Такие оценки нужно пробовать уточнять, прибегая к консультациям с экспертами, либо выделять дополнительное время на проведение исследований технологии или повышение квалификации. Время на исследования и изучения по возможности следует выделить на этом этапе, отложив собрание по сверке оценок.

На собрании Менеджер Проекта должен проверить, что:

- все оценки сделаны, прошли согласование с ведущими разработчиками и тестировщиками (если они есть в проекте)
- все «места неуверенности» по возможности устранены (проведены необходимые консультации либо исследования), либо возможные риски адекватно оцениваются всеми кого они затрагивают
- сделанные оценки не вызывают споров у участников команды

Затем на этом же собрании следует перейти к расчёту сроков сдачи версий.

В таск-листе задачи записаны в предполагаемой последовательности их выполнения. Однако при этом часто есть задачи, которые могут выполняться «параллельно» друг с другом. Первой задачей, которую надо решить для расчёта срока сдачи версии есть задача выделения «критического пути» этой версии.

Критический пусть есть наиболее длинная (по суммарным оценкам) последовательность взаимосвязанных задач (каждая следующая может выполняться только после завершения предыдущей) среди всех задач данной версии. Определять критический путь нужно с обязательным участием всех членов команды, так как в сопутствующих обсуждениях могут «всплыть» новые неучтённые ранее связи между задачами, новые задачи, неясности понимания требований и т.п.

После определения критического пути для версии следует просуммировать сначала оптимальные, затем пессимистические оценки трудозатрат, и полученные суммы привести к принятым единицам измерения сроков сдачи проекта (дни, недели, месяцы) или возможным датам (не забудьте про выходные и праздники!). В результате получаем оптимистичный и пессимистичный срок сдачи версии. Определение критического пути и расчет оптимального и пессимистичного сроков сдачи повторяем для всех версий проекта.

Затем, важный момент, для каждой версии к полученным срокам добавляем 10 – 20% (в зависимости от сложности проекта и т.п.) «сверху». Это время понадобится на сдачу версии Заказчику и исправление критических дефектов, а также на детальное планирование

следующей версии. Также это ваш «буфер» времени для компенсации возможных неожиданностей.

Получив готовые оптимистические и пессимистические сроки для каждой версии, Менеджер Проекта может закончить собрание, достать Контракт, и сравнить полученные данные с планом сдачи версий проекта записанным в Контракте.

Итак, для каждой версии у нас есть минимальный и максимальный расчетный срок сдачи. При этом мы понимаем, что минимальный срок сдачи рассчитан из предположения, что все задачи критического пути будут выполнены в оптимальные сроки, что на практике фактически невозможно. Для добавления «трезвости» в оценках рекомендую взять разницу между минимальным и максимальным сроком сдачи, поделить на 2 и полученное значение прибавить к минимальному сроку. Полученные цифры назовём средним и пессимистичным сроками. И только после этого можно переходить к сверке сроков с Контрактом.

Если установленный в Контракте срок сдачи попадает между средним и пессимистичным сроком, и так для всех версий – вам крупно повезло! Можно переходить к следующей главе.

В случае если для некоторой версии срок сдачи, установленный в Контракте, меньше чем расчётный средний срок, но больше чем оптимистичный срок – рекомендуется пересогласование срока сдачи версии при наличии такой возможности. Если возможности такой нет – при детальном планировании разработки данной версии нужно будет изыскивать дополнительные возможности и ресурсы, об этом речь позже. По своему опыту скажу, что выдержать такие сроки удаётся только при полном напряжении всех сил команды разработки.

В случае если для некоторой версии срок сдачи, установленный в Контракте, меньше чем оптимистичный срок – пересогласование сроков сдачи с Заказчиком является срочной необходимостью. Либо Менеджеру Проекта следует отдавать себе отчет, что сроки сдачи будут сорваны, и готовиться к возможным последствиям.

Исходя из личного опыта и наблюдений за «чужими» проектами замечу, что для 60% проектов сроки сдачи версий весьма близки к пессимистическим, ещё для 30% сроки сдачи превышают пессимистические и только для 10% проектов сроки сдачи близки к средним расчётным. Причина очевидна – даже на этапе подготовки проекта к старту у команды проекта обычно нет всей необходимой информации в ТЗ, сложные задачи воспринимаются командой оптимистично, а «подводные камни» пока не видны. Для сравнения – в области строительства ситуация аналогична, больше 50% новостроев сдаются с задержками и просрочками, и это при том что строительство как отрасль существует уже несколько тысячелетий

Если отношения с Заказчиком и «буква» Контракта это позволяют – я настоятельно рекомендую Менеджеру Проекта любым способом «уломать» Заказчика на утверждение пессимистичных сроков для всех версий и проекта в целом. Это избавит вас от несбывшихся ожиданий и лишних надежд и упростит общение в процессе разработки проекта. Если в Контракте сроки сдачи до сих пор не были записаны – это сильно облегчает дело.

Если изменение сроков невозможно – предложите Заказчику урезать функционал выпускаемых версий. Ищите любые компромиссы, но утверждённые сроки сдачи не должны быть ниже расчётных средних сроков. Всегда лучше иметь шанс на приятный сюрприз, чем надежду на чудо.

Если вы «подписываетесь» на невыполнимые сроки в надежде на чудо – лучше застрелиться и не мучить почём зря команду проекта. Кстати, есть такое замечание, что шансы разработчика уйти из компании растут с каждым провалившимся проектом, в котором он участвовал.

Пересогласование сроков сдачи версий лучше проводить в формате личной встречи или видео/аудио конференции между Менеджером Проектов и Заказчиком, возможно с привлечением некоторых разработчиков для усиления аргументации. В любом случае, Менеджер Проектов должен наглядно показать Заказчику, откуда и каким образом появились новые сроки и почему они отличаются от старых. Не следует стесняться объяснять, что на этапе продажи понимание проекта было одно, а теперь, после выяснения всех вопросов – оно изменилось. Вполне нормально показывать Заказчику таск-лист с оценками и выделенным критическим путём проекта, только в таких случаях следует предупреждать, что оценки делались для расчёта сроков, а не для расчёта общих трудозатрат, особенно в случае если расчёт оплаты идёт почасово. Можно намекнуть Заказчику, что если он заинтересован в успешности проекта, а не «нагнуть Разработчика согласно Контракту» - то лучше согласиться, и рассказать, почему разработка ПО больше похожа на НИОКР чем на штамповку готовых деталей на заводе.

Если вы Менеджер Проекта, и не уверены в своей аргументации – до встречи с Заказчиком проконсультируйтесь с более опытными товарищами и своим руководством.

Если вы Заказчик, и к вам пришёл Менеджер вашего Проекта с объяснениями «почему сроки надо сдвинуть» - будьте к нему благосклонны

Проверка достижения цели – рассчитаны сроки сдачи для каждой версии по таск-листу, результаты сверены с Контрактом и согласованы с Заказчиком.

Отдельно замечу, что случай работы с невыполнимыми сроками я пока рассматривать не буду.

Глава №11: Определяем риски

В ролях - Менеджер Проекта, Команда Проекта

Артефакты – список рисков

Суть - Менеджер Проекта определяет основные риски проекта

Переходим к следующей цели:

Цель 8. Определить основные риски проекта

Я не буду пытаться в рамках данной главы охватить всю область управления рисками – она слишком обширна. Попробую дать только некоторые общие понятия и рекомендации на тему «что стоит сделать даже для маленьких проектов». Всем желающим изучить область рисков и управления оными подробнее – в Гул, читать статьи и книжки, и ходить на тренинги. Впрочем, настоящее понимание приходит только с практикой :-)

Начнём с попытки определения, что есть риск. В разной литературе можно встретить разные формулировки, более или менее правильные в зависимости от контекста. Я бы сказал так:

Риск – это некоторый момент неопределённости, который не позволяет со 100% вероятностью предсказать результат управляемого процесса.

Пример риска – неопределённость оценки трудозатрат для некоторой задачи проекта. Задача может быть выполнена как за 8 часов, так и за 12. Поэтому мы не можем уверенно рассчитать срок окончания задачи.

Риск обычно определяют как возможность некоторого негативного события, например неточной оценки, происшествие которого может вызвать негативные последствия, например сдвиг сроков. Реализацией риска обычно называют собственно происшествие данного события. Есть ещё и положительные риски, но про них писать не буду – редкая это вещь :-)

Риски есть всегда в любых проектах, это нормально. Всегда есть, например, риск необходимости замены разработчика в процессе разработки, которая может вызвать задержки в сроках по причине необходимости ввода нового разработчика в курс дел. Точность оценок – всегда риск. Почти есть всегда риск при планировании пропустить некоторые задачи, такие как например подготовка среды для заливки кода (об этом позже).

Риски условно можно разделить на критичные и слабо критичные для общего развития проекта. Критичными можно назвать риски, при реализации которых потребуются сдвигать сроки сдачи версии(й) и пересогласовывать новые сроки с Заказчиком, чего Заказчики обычно не любят. Слабо критичным можно назвать риск, при реализации которого особых «телодвижений» делать не придётся. Например, выполнение таска за 10 часов, а не за 8 всё равно позволит уложиться в назначенный срок сдачи версии, так как при планировании были сделаны соответствующие «закладки» по трудозатратам.

Управлением рисками можно назвать выделение рисков и предусмотрение мер их предотвращения, либо ликвидации последствий. Как говорится, «знал бы, где упал – ...», вот и пробуем догадаться.

Любимым инструментом управления рисками является создание «закладок» при планировании сроков сдачи версий и всего проекта в целом. Делается это простым образом – для всех задач ставятся максимальные или даже завышенные оценки, плюс иногда добавляются «мифические» задачи которые, по сути, есть «буфер» дополнительного времени. Нормальным размером «закладки» является порядка 10 – 20% от рассчитанного по точным оценкам срока сдачи версии. Такие закладки позволяют справиться с реализацией слабо критичных рисков, а иногда и одиночных критичных рисков, без принятия дополнительных мер.

На практике, закладки в 15 -20% «съедаются» почти всегда за счёт неточности оценок, непредвиденного отсутствия на работе участников проектной команды (болезни и т.п.), ошибок в разработке и планировании. За счёт наличия «закладки» у Разработчика не возникает необходимости просить Заказчика сдвинуть сроки или увеличить бюджет при появлении каждой мелкой проблемы, которые есть фактически всегда. В случае если закладка не была «съедена» - это время можно потратить на более детальное тестирование и оптимизацию кода версии.

При работе по схеме fixed scope & fixed cost «закладки» обычно больше чем при работе по схеме time & material, так как согласование изменений сроков и бюджета в фиксированной схеме обычно несколько сложнее, а возможностей «поймать» Заказчика для выяснения требований меньше, из-за чего неидеальность требований становится дополнительным риском. Кроме того, размер «закладок» обычно растёт обратно пропорционально качеству требований на момент старта проекта – для компенсации этого же риска.

Ещё раз повторяюсь, что «закладки» являются инструментом управления преимущественно слабо критичными рисками. Теперь посмотрим что можно (и нужно) сделать с критичными рисками.

В первую очередь, их нужно определить. То есть, нужно представить возможные ситуации, реализация которых может привести к серьёзным изменениям сроков и бюджета проекта, либо к невозможности проекта. Таких ситуаций может быть очень, очень много – начиная от падения астероида и вплоть до... Поэтому ограничиваем рассмотрение теми ситуациями, на которые можно повлиять, и выписываем их в порядке убывания критичности.

Определение критичных рисков дело сложное, сильно зависящее от нюансов проекта, отношения с Заказчиком, знания команды проекта, адекватной оценки возможностей компании-Разработчика и многого другого. Для наглядности приведу несколько примеров критичных рисков:

- необходимость замены ведущего специалиста проекта во время разработки
- невозможность продолжения разработки согласно выбранной архитектуре по причине дефектов в используемых сторонних компонентах, необходимость изменения архитектуры
- кардинальное изменение в требованиях проекта со стороны Заказчика во время разработки, ведущее к перепланированию всего и, возможно, переписке уже сделанного

Для каждого определённого риска нужно рассмотреть возможные инструменты его предотвращения, минимизации или ликвидации последствий. Для примеров описанных выше это могут быть:

- в фирме есть другие специалисты аналогичного профиля и квалификации, значит нужно уделить внимание комментированию кода и поддержанию в актуальном состоянии технической документации во время разработки, таким образом, при замене специалиста будет минимизировано время на обучение; возможно также с самого начала привлечение возможной замены как консультанта проекта.

- до старта разработки по возможности провести максимально глубокое исследование сторонних компонент на предмет наличия/отсутствия блокирующих разработку дефектов.
- предусмотреть и прописать в Контракте, либо просто согласовать с Заказчиком, процедуру изменения требований – чтобы для Заказчика изменение сроков по причине изменения требований не было сюрпризом.

Помощь Менеджеру Проекта в определении и поиске инструментов минимизации критических рисков может оказать команда проекта, эксперты и другие сотрудники компании-Разработчика. Если вы в чём-то не уверены - можно и нужно задавать вопросы и искать ответы. Кроме того, большинство технических рисков определяются на этапе определения концепции архитектуры и составления таск-листа - вспомните про спорные моменты и оценки с большим разбросом, выясните подробности у специалистов - и оценивайте критичность.

Выделив время на определение, оценку и описание рисков Менеджер Проекта самостоятельно получает набор очень важных рекомендаций для дальнейшего построения плана проекта. Пользу этих рекомендаций трудно переоценить.

Вообще, вся суть управления рисками может быть сведена к старой поговорке «надейся на лучшее, но готовься к худшему». Управление рисками есть в повседневной жизни каждого человека, только не каждый об этом задумывается и называет свои действия умными словами. Прослушать прогноз погоды и взять с собой на улицу зонтик – это тоже пример управления рисками. Риск – намокнуть, зонтик – инструмент минимизации риска.

Многие Менеджеры Проектов управляют рисками неосознанно, просто совершая определённые действия, основанные на предыдущем опыте. Я же, вместе с авторами книжек по Управлению Рисками, рекомендую делать это осознанно – тогда опыт будет лучше усваиваться, и ошибок будет меньше.

Проверка достижения цели – наличие списка определённых критичных рисков и инструментов их минимизации.

Глава №12: Plan is nothing, Planning is everything!

В ролях - Менеджер Проекта, Команда Проекта

Артефакты – ТЗ, таск-лист, список рисков, План Проекта

Суть - Менеджер Проекта создаёт первую версию Плана Проекта

Наконец, у нас есть ТЗ, таск-лист, и список рисков. Переходим к финальной цели подготовительного этапа:

9. Зафиксировать все принятые решения и сделанные оценки и перейти к составлению плана проект

Что такое план проекта (ПП)? Это основной инструмент, как планирования будущих работ, так и отображения работ уже завершённых и текущих. Это очень важный момент – ПП нужен не только чтобы «набросать» что и в какой последовательности делать будем, но и чтобы достоверно отображать ситуацию в проекте в каждый контрольный момент времени.

План проекта – это «живой» документ, который постоянно (регулярно) актуализируется и изменяется в процессе разработки. Актуальный на текущую дату план проекта должен быть своего рода «приборной панелью», по которой можно увидеть, что уже сделано, что в процессе, насколько актуальны сроки сдачи и т.п. Это конечно в идеале...

В реальности на план и поддержание его в актуальном состоянии часто «забывают» сразу после его создания. Тем самым теряя важный инструмент «диагностики» проекта. И увеличивая шансы получения «внезапных» проблем. Впрочем, про то, как правильно обновлять план – будет позже. Для контекста этой главы просто имеем ввиду, что план надо будет неоднократно актуализировать, и учитываем этот момент при его построении.

Базой для создания ПП является таск-лист, так как в нём уже определены основные задачи разработки и тестирования. Оптимальной формой для представления ПП является диаграмма Ганта. Она позволяет наглядным (что важно) образом показать все задачи проекта в последовательности их выполнения, привязанные к календарной сетке, с указанием ответственных за выполнение каждой задачи, с демонстрацией точек появления необходимых артефактов и других «вех», с возможностью указания процентов завершения либо выполненных трудозатрат по каждой задаче, и многое другое. В общем, я думаю что все читатели уже знают что такое диаграмма Ганта, а если кто не знает – Гул в помощь.

Инструменты для создания плана проекта в виде диаграммы Ганта есть разные, из них самый, наверное, распространённый – MS Project, сам им пользуюсь. Но, в дальнейшем изложении постараюсь к инструментам не привязываться.

Итак, задача №1 – берём таск-лист, и все задачи разработки из таск-листа переносим в план проекта. Для каждой задачи назначаем исполнителя (можно несколько). Для каждой задачи не забываем указать планируемые трудозатраты, причем здесь, это важно, в отличие от таск-листа, ставим наиболее вероятную (реальную) оценку (которая обычно где-то посередине между оптимальной и пессимистичной). Учитываем риски – для сложных задач планируемые трудозатраты близки к пессимистичной оценке, для простых – к оптимальной. В первую очередь вносим задачи критического пути, не забывая про последовательность. Потом добавляем задачи, которые могут выполняться параллельно с задачами критического пути, и расставляем их на календарном плане исходя из возможностей участников команды проекта.

Если вдруг оказалось что из-за «параллельных» задач сроки сдачи сдвигаются – значит, критический путь был определён неверно. В таком случае – собираем разработчиков и переопределяем критический путь.

Для каждой версии, которая сдаётся Заказчику или подлежит внутреннему тестированию, рекомендую добавить задачу-milestone после завершения финальной для этой версии задачи разработки, для того чтобы все контрольные точки были наглядно представлены в ПП.

Задача №2 – добавляем в план проекта все задачи тестирования, согласно таск-листу. По завершению финальной задачи тестирования для версий, которые сдаются Заказчику, рекомендуем опять же добавить milestone.

Теперь нужно добавить в план проекта дополнительные задачи, такие как например:

- подготовка сред разработки – обычно делается на подготовительном этапе параллельно с созданием плана проекта и т.п., но иногда нужно время и после старта
- подготовка сред тестирования, «показа» и рабочей среды продукта (где он будет жить по окончании разработки) – всё это надо сделать до того как такая среда будет востребована, например параллельно с разработкой
- заливка кода в среду «показа» или рабочую среду и контроль целостности и работоспособности, для крупных проектов этом может занимать целый рабочий день или даже больше
- подготовка сред контроля версий, дефектов, отчётности и т.п. – обычно делается на подготовительном этапе, но бывает всякое
- обновление технической документации – обычно делается по завершению версии
- подготовка сценариев тестирования – следует делать заранее, до начала тестирования, обычно эта задача выполняется параллельно с разработкой
- после начала «показа» каждой версии иногда стоит «зарезервировать» один или несколько дней (в зависимости от размеров проекта) для ответов на вопросы Заказчика и срочные исправления критических дефектов. Такие задачи вставляются между версиями и растягивают сроки проекта.
- детальное планирование следующей версии – обычно выполняется после подтверждения приёмки предыдущей версии перед началом разработки или параллельно с первыми задачами следующей версии, смысл в том, чтобы учесть всю новую информацию и реальное положение дел и детализировать задачи.

Все эти задачи будут описаны более детально в следующих главах. Сейчас речь идёт о том, чтобы не забыть, что это делать будет надо, на это нужно будет время и исполнитель. Какие именно дополнительные задачи актуальны для данного проекта решает Менеджер Проекта совместно с командой, исходя из своего опыта, применяемых процессов и инструментов, отношений с Заказчиком и т.п. «Думайте сами, решайте сами», моя задача – дать общие рекомендации применимые в большинстве случаев 😊

Большинство дополнительных задач обычно расположены «параллельно» критическому пути на диаграмме Ганта, а значит, не влияют на сроки проекта. Но некоторые – такие как исправление критических дефектов и детальное планирование – ставятся между версиями и, соответственно, влияют на сроки.

Если при согласовании сроков были сделаны правильные «закладки» времени – сроки сдачи версий в плане проекта должны соответствовать согласованным срокам. Если полученные в ПП сроки отличаются в меньшую сторону от согласованных сроков (как и должно получиться для нормального случая когда согласование сроков проведено по пессимистичным оценкам и предусмотрен запас времени в виде «закладок») – добавляем «буферные» задачи для «выравнивания». Если сроки в ПП отличаются от согласованных сроков в большую сторону –

придётся повторить процедуру согласования сроков, и при этом снова не забыть вставить «буферы» в размере порядка 10% от общего длительности для каждой версии.

Время, заложенное в «буферных» задачах, является «подушкой безопасности» для компенсации реализованных рисков. Кроме этого, по мере дальнейшей детализации задач разработки и тестирования, которая будет выполняться на старте каждой новой версии, планируемые трудозатраты могут увеличиваться. Подробнее об этом позже.

Теперь в ПП внесены все задачи разработки, тестирования, дополнительные задачи, и вложены «буферы». Переходим к актуализации календаря.

В календарь проекта нужно обязательно внести все нерабочие дни, попадающие в сроки проекта, а также запланированные отпуска сотрудников. Возможно (и скорее всего), сроки сдачи версий сдвинутся. Хорошо, когда сроки устанавливаются в относительных величинах. Плохо, если на конкретные даты. Если после внесения нерабочих дней и отпусков сроки сдвинулись за пределы согласованных сроков, даже с учётом расходования «буферов» - опять же, рекомендую по возможности сроки пересмотреть (либо пересмотреть отпуска). Также рекомендую пересмотреть сроки, если «буферы» израсходованы полностью ещё на этапе составления ПП.

Касательно даты старта проекта. По-хорошему, правильной датой старта проекта является дата начала подготовительного этапа. Если позволяют обстоятельства, рекомендую внести в ПП все уже выполненные задачи, пометить их как выполненные, а старт разработки поставить на следующий рабочий день после согласования ПП. В таком случае план проекта с самого начала будет честно выполнять свою функцию – показывать, что уже сделано, что делается, и что будем делать.

Теперь у нас есть почти готовый план проекта. Осталось согласовать его с командой проекта, при необходимости - утвердить с руководством и Заказчиком, и можно начинать думать о собственно разработке 😊

Глава №13: Про среды

В ролях - Менеджер Проекта, Команда Проекта

Артефакты – спецификация сред проекта

Суть – Менеджер Проекта совместно с командой определяет среды, используемые в проекте, и составляет их спецификацию

Собственно, эту главу тоже можно отнести к подготовительному этапу, т.к. описанные в ней действия желательно сделать до старта разработки. Речь в ней пойдёт об определении различных сред, используемых в проекте, установке необходимого софта и создании руководства (спецификации) по использованию этих сред в течение проекта.

Перечислю основные используемые среды:

- среда планирования и учёта задач
- среда разработки
- среда тестирования
- среда учёта дефектов и изменений
- среды запуска проекта
- среда хранения требований и других проектных документов

Все эти среды нужно определить и, желательно, задокументировать до начала разработки. Рассмотрим каждую из сред подробнее:

Среда планирования и учёта задач

Данная среда состоит из нескольких компонент, которые будут рассмотрены по отдельности:

1. Представление плана проекта

План Проекта – важный документ для всей команды проекта (наглядное представление последовательности задач), и, в некоторых случаях, для заказчика (следить за прогрессом). Хранить его надо в некотором доступном для заинтересованных лиц месте. Следует учесть, что ПП часто актуализируется и изменяется в ходе проекта – и узнавать об этих изменениях должны все участники проекта своевременно. Варианты реализации компонента могут быть разные, например:

- использовать MSP, рассылать новые версии адресно или «на всех», хранить файл ПП в shared папке
- использовать web-based систему планирования, например Jira
- вести планирование по рекомендациям Agile методики и использовать task board

Какой именно вариант подходит именно вам для данного проекта – решайте сами.

В некоторых случаях детальный план проекта команде проекта показывать может быть не принято. Причины могут быть разные, хотя мне кажется, что это всё глупости. Тем не менее, в таких случаях всё равно есть необходимость показать команде последовательность будущих задач и определение «как это делаем» является элементом данной среды.

2. Доступность требований по каждой задаче из плана проекта для разработчиков

Кроме последовательности задач команде проекта, пожалуй, в первую очередь, нужно знать актуальные требования по каждой задаче. Часто для решения данной задачи просто рассылается «на всех» последняя версия требований, и разработка ведётся по ней. Если качество требований достойное, и есть чёткая привязка задач из ПП к разделам требований – можно делать и так. Только не стоит забывать о том, что требования уточняются и изменяются в процессе разработки, поэтому новая информация должна быть сразу доступна всем участникам проекта. Уточнения и изменения обычно приходят в виде электронной почты, или истории чата с Заказчиком, или записей по результатам аудио/живого общения с Заказчиком. Если эта новая информация сразу не добавляется в общую спецификацию требований, а просто передаётся разработчикам в том же виде – ждите возможных проблем.

Если сначала работали по спецификации, а потом было еще 3 письма и пару указаний в устной форме – часто может получиться, что никто толком не знает, что именно и как надо сделать, и задачи выполняются с ошибками, требующими последующих исправлений. В общем, помните, что требования должны быть доступны в одном и том же месте в актуальной форме. Рекомендуемые варианты реализации:

- работать по спецификации и обновлять саму спецификацию по каждому изменению и уточнению требований; для этого в команде рекомендуется иметь выделенного аналитика
- работать в среде Sharepoint с привязкой требований к плану проекта в MSP – аналогично следующему пункту
- хранить требования в issue tracking системе, например Jira, в виде задач; уточнения и изменения добавлять в виде комментариев к задаче (что удобно ещё и тем, что видна история изменений); а ещё к такой системе можно дать доступ Заказчика, чтобы он сам мог видеть актуальные требования и их уточнять – улучшает качество понимания в команде проекта; а ещё такая система наиболее наглядна и удобна в использовании для разработчиков; а ещё лучше всего «стыкуется» с другими задачами этой же среды и других сред, т.к. есть плагины для диаграммы Ганта, file sharing, SVN etc.
- работать по Agile–методике, выяснять/уточнять требования на регулярных собраниях команды проекта с Заказчиком (работает только в случае отличной доступности Заказчика)

При любом варианте помните, что информация об уточнениях и изменениях требований должна быть не просто доступна разработчикам, но следует целенаправленно обращать внимание разработчиков на эту информацию. В вариант с Jira это, кстати, проще всего, т.к. открытая страница с задачей обычно висит в браузере во время выполнения этой задачи.

3. Учёт выполненных и текущих задач

Смысл, думаю, понятен, а вот варианты реализации могут быть различными:

- Менеджер Проекта «ручками» актуализирует ПП
- Команда отчитывается в MSP (через MSP сервер), прогресс виден на диаграмме Ганта
- То же можно сделать через Jira с правильными плагинами
- Task board для Agile

4. Учёт реальных трудозатрат команды проекта

Непосредственно связано с предыдущим компонентом. Типичные варианты реализации:

- команда проекта пишет ежедневные отчёты (из которых МП потом ручками собирает данные и актуализирует ПП, применимо только для небольших проектов и команд)
- трудозатраты и прогресс отчитываются в MSP/Jira/etc
- трудозатраты вообще можно не учитывать т.к. Заказчик оплачивает всё рабочее время, а прогресс по задачам отмечает МП в виде процентов

Определить, как именно будут реализованы описанные компоненты, Менеджер Проекта может в большинстве случаев самостоятельно, руководствуясь стандартами своей компании, нюансами самого проекта и т.п. Для достижения большей уверенности можно проконсультироваться с командой проекта. В некоторых случаях некоторые элементы среды могут быть определены Заказчиком.

После того, как всё определили – настоятельно рекомендую зафиксировать принятые решения – что для чего и как будем использовать - в текстовом виде. Полученный «документ» назовём спецификацией сред, далее его надо будет дополнить принятыми решениями по следующим средам.

5. Среда хранения документов проекта

В рамках данного компонента речь идёт о доступном для всех участников проектной команды (и иногда Заказчика) месте хранения публичных документов проекта. Таких, как например спецификация требований (ТЗ), спецификация сред, концепция архитектуры проекта, мокапы дизайна проекта и т.п. Варианты возможной реализации:

- shared folder (file hosting)
- использовать SharePoint либо Jira с нужными плагинами
- SVN

Среда разработки

Основными компонентами данной среды являются:

1. ПО разработчиков

Имеется ввиду набор софта, который должен быть установлен на ПК разработчика для нормального ведения разработки. Если в проекте задействовано несколько технологий – набор софта для каждой технологии может быть разным. Выяснить, что планируется использовать можно на собрании с разработчиками исходя из принятой концепции архитектуры (собственно, в концепции этот момент может быть уже определён). По результату обсуждения принятые решения дописываем в спецификацию сред. А разработчиков отправляем устанавливать нужный софт, если это ещё не сделано.

2. Хранение и контроль версий

Думаю, все знают что это такое и насколько важно использовать среду контроля и хранения версий при разработке проекта. На собрании с разработчиками исходя из принятых в компании правил и нюансов конкретного проекта нужно определить:

- какой программой пользуемся
- где она будет установлена, параметры доступа
- правила наименований, определения «веток» и фиксации версий

Результаты добавляем в спецификацию сред. Даём задачу ответственному персоналу на установку и настройку всего что нужно.

В некоторых случаях среда хранения и контроля версий может быть определена Заказчиком – тогда просто добавляем нужную информацию в спецификацию сред.

3. Среда «сборки» проекта

Необходимо определить где будет происходить «сборка» (компиляция) всего проекта и необходимое для этого ПО. В крупных проектах для этой цели может выделяться отдельный сервер. В небольших проектах где используется одна технология сборка может выполняться на ПК одного из разработчиков. В любом случае этот момент стоит определить и зафиксировать спецификации сред.

Среда тестирования

По-хорошему, среда тестирования должна эмулировать среду типичного пользователя проекта. Даже если в проекте для тестирования сайта достаточно Smoke tests & Monkey testing – нужно использовать наиболее распространённые браузеры, а не то, что нравится тестировщику. Компоненты среды тестирования:

1. Эмуляция типичного пользователя

Пример для простого сайта – браузеры, через которые тестируем сайт. Для исполняемого приложения – ПК должен соответствовать некоторому «среднему» уровню (который должен быть определён в нефункциональных требованиях) плюс на нём должно быть установлено «типичная» ОС и набор программ. В общем, речь идёт про создание условий приближённых к условиям типичных пользователей готового продукта. В некоторых случаях эти условия детально определены в требованиях проекта, но такое встречается редко. В большинстве случаев эти условия можно определить исходя из понимания проекта и здравого смысла. Для этого МП может собрать тестировщиков проекта и обсудить с ними доступные варианты. Принятые решения (или соответствующий раздел из требований) добавляем в спецификацию сред.

Заметка – часто для эмуляции среды пользователя используют виртуальные машины. В таком случае в спецификации сред не забудьте указать, где и как они хранятся и параметры доступа к ним.

2. Специальное ПО

Если в проекте применяются средства автоматического тестирования, используется ПО для нагрузочного тестирования, или используются другие специализированные средства тестирования – их тоже стоит описать в спецификации сред. Какие именно спецсредства и как планируется использовать следует обсудить с тестировщиками проекта до старта разработки.

Среда учёта дефектов и изменений

В правильном варианте должна являться компонентом среды планирования и учёта задач, так как исправление дефектов и реализация изменений требований являются типичными задачами разработки. Мне известны такие варианты совмещения планирования и учёта задач с учётом дефектов и изменений:

- Sharepoint + MSP
- Jira + плагины

Оба эти варианта позволяют удобное управление задачами разработки и разделение задач на разработку, устранение дефектов и реализацию изменений. Также оба варианта позволяют тестировщикам (или Заказчику) регистрировать дефекты и новые задачи, а разработчикам отчитываться о текущем прогрессе и завершении. И там и там есть возможности комментирования, куча статусов для отображения возможного состояния задачи (new, assigned, needs information, duplicate, fixed, closed, withdrawn). В общем – рекомендую.

Если планирование и учёт задач ведётся в среде, не рассчитанной на регистрацию дефектов и запросов на изменения – значит, для регистрации дефектов и изменений понадобится выделенная среда. Суть такой среды – с одной стороны тестировщики регистрируют дефекты, с другой стороны разработчики отчитываются про исправления. Подробнее отправлю в Гул по запросам bug tracking или issue tracking. Простым вариантом, кстати, но только для внутреннего пользования, является всё тот же task board. Минус в том, что Заказчику его показывать трудно.

Принятое решение по среде учёта дефектов и изменений следует внести в спецификацию сред.

Среды запуска проекта

Нужно определить в каких средах будет запускаться создаваемое приложение. Ниже описаны типичные варианты. При описании каждой среды важно определить версии для ОС и всего ПО используемого для запуска приложения.

1. Среда запуска для разработки

При разработке после «сборки» проекта обычно есть необходимость запустить собранную версию и проверить реализованный функционал. Делают это сами разработчики. Для проекта по разработке небольшого сайта эта среда может представлять собой Apache поднятый на ПК одного из разработчиков. Для проекта по созданию исполняемого приложения – сам ПК разработчика. Для крупных проектов это может быть несколько различных сред на разных серверах. Иногда (часто) среда запуска для разработчиков может совпадать со средой «сборки» проекта.

Определить параметры среды Менеджер Проекта может на собрании с разработчиками. В некоторых случаях среда может быть определена Заказчиком. В любом случае принятое решение описываем в спецификации сред, даже если это «запуск на ПК Васи».

2. Среда запуска для тестирования

Тестирование рекомендуется выполнять в среде запуска отличной от среды запуска для разработчиков. Причина проста – разработчики «собирают» новые версии значительно чаще, чем выпускают версию для тестировщиков, а внезапные изменения в тестируемой версии затрудняют тестирование.

Важным моментом при определении среды запуска для тестирования является необходимость учесть перспективу нагрузочного тестирования, если оно необходимо. Грубо говоря, выбранная среда запуска должна позволять нагрузить приложение согласно ТЗ.

Среду запуска для тестирования рекомендуется определять на собрании с разработчиками и тестировщиками вместе. В некоторых случаях среда может быть определена/предоставлена Заказчиком. В других случаях среда запуска для тестирования может быть описана в разделе «среды тестирования». Но в любом случае принятое решение должно быть описано в спецификации сред.

3. Среда запуска для показа Заказчику

Думаю, тут всё понятно. Речь идёт о среде, в которой будем «сдавать» готовые версии Заказчику. Часто она совпадает либо со средой запуска для тестирования, либо со средой запуска готового проекта.

4. Среда запуска готового проекта

Наконец, важно определить параметры среды, в которой будет работать готовое приложение, и сделать это надо до старта разработки – так как именно от параметров этой среды зависит подбор параметров всех остальных сред запуска приложения. В идеале, все среды должны быть одинаковы, но такое возможно достаточно редко вследствие частых ограничений по ресурсам. В идеале, требования к среде запуска должны быть определены в ТЗ. Но даже в таком случае всё равно стоит их скопировать в спецификацию сред.

Всё, теперь у нас есть законченная спецификация используемых сред проекта. Этот документ должен быть доступен всем участникам проектной команды в течение всего процесса разработки. Основные плюсы наличия такого документа:

- наглядность представления информации; меньше шансов, что кто-то что-то забудет
- доступность всей информации по средам в одном месте, удобство внесения изменений, если они вдруг появятся (такое бывает – например, Заказчик захочет увеличить пиковую нагрузку сайта со 100 до 1000 одновременных пользователей, что повлечёт за собой изменения требований к среде запуска готового проекта)
- удобство при вводе в проект новых участников

Если в проекте есть документ, описывающий архитектуру приложения – спецификацию сред можно включить в этот документ. Либо приложить к спецификации требований. Либо оставить как отдельный документ. Но, в любом случае, спецификацию сред стоит написать и опубликовать для всех участников проекта.

Глава №14: Старт разработки

В ролях - Менеджер Проекта, Команда Проекта

Артефакты – ТЗ, План Проекта, список рисков, спецификация сред проекта, концепция архитектуры, оргсхема команды проекта

Суть – Менеджер Проекта объявляет старт разработки

По окончании подготовительного этапа перед началом собственно разработки рекомендую провести собрание с командой проекта. Цели собрания:

1. Проверить готовность всех артефактов созданных за время подготовительного этапа
2. Проверить готовность команды к старту работ
3. Объявить старт работ

Напомню правила проведения собраний:

- оповещать о собрании лучше заранее, желательно за день, в письменном виде, рекомендуется использование автоматических напоминаний (outlook appointment)
- цели собрания должны быть ясны всем участникам
- собрание надо «вести», последовательно и планомерно, не допуская «галдежа», «один оратор в эфире» - хорошее правило
- результаты собрания (принятые решения) рекомендуется фиксировать в формате meeting minutes

Перейдём к детальному рассмотрению целей собрания:

Цель №1. Проверка готовности артефактов

В течение подготовительного этапа проекта должны быть созданы следующие артефакты:

- ТЗ. Должно быть достаточно полным и ясным для старта работ по первой версии (старта первой итерации). Все участники проекта должны быть ознакомлены с ТЗ, сам документ должен находиться в доступном для всех участников проекта месте на протяжении всего проекта
- План Проекта. В плане должны быть представлены все основные элементарные задачи разработки, тестирования и дополнительные задачи. ПП должен быть доступен всем участникам проекта на всём протяжении проекта. В плане крайне желательно иметь «буферы» времени для страховки от мелких рисков.
- Список рисков. Должен охватывать все найденные критические риски. Находится у Менеджера Проекта.
- Спецификация сред. Должна охватывать все основные используемые среды – разработки, тестирования, запуска, планирования и учёта, хранения артефактов. Должна быть доступна команде проекта на всём протяжении проекта.
- Концепция архитектуры. Должна кратко описывать выбранную техническую реализацию проекта. Должна быть доступна всем разработчикам проекта на всём

протяжении проекта. Рекомендуется назначить разработчика, ответственного за обновление концепции архитектуры в ходе разработки.

- Организационная схема команды проекта. Должна наглядно представлять распределение ролей и ответственности в команде проекта и быть доступна всей команде на всём протяжении проекта.

Если все артефакты созданы в том или ином виде и соответствуют описанным в этой и предыдущих главах критериям – переходим к проверке сред:

- Среды разработки должны быть уже установлены
- Среда хранения и контроля версий должна быть уже установлена, доступ к ней проверен
- Среда запуска для разработки – желательно чтобы уже была готова
- Среда планирования и учёта задач - должна быть уже готова
- Среда учёта трудозатрат – должна быть готова
- Среда хранения артефактов (документации) – желательно чтобы была готова

Среды тестирования, учёта дефектов и изменений и остальные среды запуска можно установить и настроить после начала разработки.

Проверять готовность артефактов (документов) и сред рекомендуется последовательно на собрании с командой проекта. Важный момент – вся команда должна признать готовность и достаточность артефактов и сред для начала разработки первой версии проекта.

Цель №2. Проверка готовности команды

Выполняется в процессе проверки готовности артефактов и сред. Если все участники команды проекта ознакомлены со всеми артефактами, не имеют по ним принципиальных вопросов и подтверждают готовность необходимых им сред – значит, команда готова к старту.

Цель №3. Объявить старт работ

Если все необходимые артефакты и среды в порядке, и команда готова – Менеджер Проекта может официально объявить старт разработки проекта.

Тут есть ещё один важный момент, связанный, в том числе, с предыдущей целью – мотивация команды проекта на успешное выполнение проекта. В этом материале я намеренно слабо затрагиваю вопросы мотивации и эмоциональные моменты управления командой разработки – так как данные темы пока с трудом поддаются чёткому описанию, и по моему опыту решения принимаются больше интуитивно, чем логически. Тем не менее, мотивация команды является одним из ключевых факторов успешности проекта. Поэтому обращу внимание Менеджеров Проектов на необходимость убедиться в её наличии до старта разработки, а сделать это удобно как раз на собрании, посвящённом старту разработки.

«Чувствуй силу, Люк», как говорил мастер Йода. Проверьте, что в команде проекта нет уныния и чувства безнадёжности. Удостоверьтесь, что каждый участник команды проекта понимает не только суть проекта, но и зачем выполнение данного проекта нужно именно

ему. Для кого-то это просто получение ЗП, для кого-то – возможность повысить свою квалификацию в некоторой технологической области, для кого-то – возможность проявить себя в новой роли, проявить оргспособности, доказать выросшую квалификацию и получить повышение. Каждый должен чётко понимать, что и зачем он делает, и что эти действия ему дадут – это один из залогов успеха. Для выяснения этих вопросов в течение подготовительного этапа МП может провести тет-а-тет беседы с каждым участником команды проекта, выслушать его вопросы и сформулировать совместное видение этого самого «зачем».

При объявлении старта работ – Менеджер Проекта может выступить с небольшой речью, цель которой – в том числе и закрепить положительную мотивацию команды проекта. Что и как надо говорить – решает каждый Менеджер Проекта самостоятельно с учётом нюансов проекта и знания участников команды проекта.

Глава №15: Детальное планирование разработки итерации

В ролях - Менеджер Проекта, разработчики

Артефакты – ТЗ, План Проекта -> детализированный ПП для первой версии

Суть – Менеджер Проекта совместно с разработчиками уточняет план разработки первой версии

Детальное планирование разработки выполняется перед стартом каждой новой итерации проекта. Задача – разделить длительные задачи на более «мелкие». Основной смысл – увеличить количество «точек контроля» в проекте за счёт проверки выполнения выполненных задач.

Рассмотрим на примере. В проекте есть выделенная задача разработки под названием «форма регистрации пользователя», оцененная в 4 рабочих дня. Эту задачу Менеджер Проекта совместно с ответственными разработчиками разбивает на следующие подзадачи:

- блок login & password + captcha, с проверкой на уникальность имени
- блок ввода контактной информации
- блок подгрузки аватары
- блок дополнительных данных

Каждая из выделенных подзадач может быть выполнена за 1 рабочий день. Все задачи делает один и тот же разработчик – тот, кто был ответственный за общую задачу «форма регистрации пользователя». По выполнению всех подзадач автоматически выполняется общая задача. При этом выполнение каждой подзадачи можно проверить отдельно. В случае если с выполнением подзадачи возникнут трудности – информация поступит Менеджеру Проекта с большей вероятностью, чем при выполнении общей задачи целиком, без дополнительных контрольных точек.

Напомним, что критериями для определения задач разработки на подготовительном этапе являются:

- «элементарность» - задача может быть выполнена одним и тем же разработчиком (имеется ввиду роль, а не человек, т.е. задача «привязана» к одной технологии)
- «несвязанность» - в процессе выполнения этой задачи не требуются результаты других задач

Задачи, определённые таким образом, могут быть весьма и весьма длительными. Для небольших проектов разработка целой версии может быть представлена одной задачей. Контролировать ход выполнения длительных задач зачастую бывает неудобно – так как основным инструментом контроля является «опрос» разработчика, ответственного за задачу. В случае если в процессе разработки возникли проблемы – информация о них почти гарантировано поступит Менеджеру Проекта с задержкой (так как разработчики часто сообщают о проблеме только после нескольких неудачных попыток её решить),

Хорошей практикой является детализация задач до такого уровня, когда большинство подзадач могут быть выполнены за один рабочий день. В таком случае Менеджер Проекта может получать актуальную информацию о ходе разработки ежедневно и оперативно реагировать на проблемы в случае их возникновения. Как реагировать на проблемы – про это будет написано в следующих главах. Что касается проверки выполненных подзадач – она должна выполняться самими разработчиками, а также (по возможности) избирательно самим Менеджером Проекта.

Итак, перед стартом разработки новой версии Менеджер Проекта собирает разработчиков и вместе с ними детализирует задачи разработки для этой версии. По результатам необходимо обновить План Проекта. В процессе обсуждения могут быть обнаружены новые «непонятки» в требованиях, также могут измениться оценки трудозатрат – это нормально.

Дополнительной (а иногда и основной) целью детализации задач разработки является «последняя проверка» полноты и ясности требований, а также правильности оценок / верности сроков. Актуальность такой проверки растёт с каждой следующей версией проекта – так как в процессе разработки и сдачи предыдущих версий часто появляется новая информация, которая приводит к изменению оценок и планов. Фактически, детализация задач – это «маленький подготовительный» этап для каждой версии, в чём-то повторяющий общий подготовительный этап для всего проекта в целом.

Когда все задачи разработки детализированы до оптимального уровня (один рабочий день на подзадачу это хорошо), и всё ОК со сроками и требованиями Менеджер Проекта даёт команду на начало разработки версии (старт итерации).

Глава №16: Про тест-планы

В ролях - Менеджер Проекта, тестировщики

Артефакты – ТЗ, План Проекта

Суть – Менеджер Проекта ставит тестировщикам задачу разработки планов/сценариев тестирования

Теперь, когда разработчики занялись своими прямыми обязанностями по разработке версии – самое время вспомнить про тестирование. Фактически, с задачами тестирования стоит сделать то же самое, что с задачами разработки – нужно их детализировать.

На подготовительном этапе при составлении таск-листа и плана проекта были определены виды тестирования для выпуска каждой версии, а также была проведена оценка планируемых трудозатрат. Соответственно, предполагаем наличие некоторого понимания, как именно и в каком порядке будет производиться тестирование каждой версии. Теперь это понимание нужно перевести в вид некоторого плана тестирования.

Глава №16. Часть 1.

В которой опишу некоторые нюансы детализации для разных видов тестирования:

1. Функциональное тестирование

Предполагает проверку реализованного функционала на работоспособность и соответствие ТЗ. Часто задача может быть переформулирована так: проверить, что пользователь сможет выполнить все необходимые действия и получить нужный ему результат. То есть, для выполнения функционального тестирования полезно/необходимо знать сценарий работы пользователя приложения. В редких случаях сценарий работы пользователя может быть предоставлен Заказчиком, чаще его составление является задачей тестировщиков проекта.

Сценарий работы пользователя представляет собой чёткую последовательность действий, которые необходимо выполнить пользователю в приложении для получения некоторого результата. Сценариев может быть несколько. Последовательность действий должна быть прямой, без ветвления. Если один и тот же результат можно получить разными способами – это будут разные сценарии. При функциональном тестировании приложения каждое действие сценария проверяется по отдельности – это нужно учитывать при разработке сценария. «Нажать на кнопку» - это не действие, а «ввести данные в форму А, нажать на кнопку X – автоматический переход на страницу Б» это действие.

Действие из сценария работы пользователя, подлежащее тестированию, обычно определяют как Test Case. Набор test cases составленный согласно сценарию работы пользователя – сценарий тестирования. Если все test cases согласно сценарию пройдены успешно – можно сказать, что функционал реализован правильно. Подробнее про сценарии тестирования и test cases можно узнать в специальной литературе – Гул в помощь.

Таким образом, при детализации функционального тестирования обычно выделяют такие задачи:

- составление сценария работы пользователя;
- составления сценария тестирования согласно предыдущему пункту;
- описание test cases;

- выполнение test cases.

Все эти задачи кроме собственно выполнения test cases можно выполнять во время разработки версии, тестировщики работают параллельно с разработчиками. И тут есть следующий важный для fixed scope & fixed cost проектов момент: обычно при оценке планируемых трудозатрат оценивают только собственно выполнение test cases. Про необходимость разработки сценариев тестирования и test cases на этапе оценок часто забывают. Если так и произошло – придется либо выполнять эти задачи «за свой счет», либо опять пересогласовывать бюджет. А вот отказываться от написания как минимум сценария тестирования до начала тестирования – не стоит, так как качество тестирования «вслепую» всегда оставляет желать лучшего, а «препирательства» с Заказчиком по поводу исправления дефектов всегда затягивают сроки и портят настроение всем участникам проекта.

Впрочем, надо заметить, что для некоторых совсем небольших проектов, либо проектов, в выполнении которых у Разработчика накоплен большой опыт – бывает достаточно smoke testing, то есть тестирования «просто так», «вслепую», без сценариев и test cases, просто по интуиции тестировщиков.

Для некоторых проектов написание test cases может быть необязательным, достаточно сценария тестирования или сценария работы пользователя, которые можно написать относительно быстро.

Но, для большинства проектов test cases писать нужно, просто необходимо, и чем детальнее – тем лучше. По трудозатратам – на написание качественных test cases обычно уходит немногим меньше времени, чем на разработку функционала, который они покрывают. Чем качественнее test cases – тем качественнее продумано тестирование, и тем меньше дефектов «дойдут» до Заказчика, а значит, тем больше шансов сдать проект во время.

Необходимость написания сценариев тестирования и test cases Менеджер Проекта определяет самостоятельно, следуя принятым в компании-Разработчике стандартам, рекомендациям тестировщиков и руководства, своему опыту и бюджету проекта.

2. Тестирование безопасности

Часто (в относительно простых проектах) совмещается с функциональным тестированием следующим образом. Как уже упоминалось, test case описывает одно действие из сценария работы пользователя. Для выполнения сценария это действие должно совершаться успешно, то есть с определённым результатом. Однако логично предположить, что это действие может совершаться и с другими результатами – в большинстве случаев негативными с точки зрения выбранного сценария тестирования. Действия, которые направлены на проверку возможности получения негативных результатов в test case, могут являться элементами тестирования безопасности приложения. В общем, рекомендация – при разработке test cases следует по возможности запланировать попытки «сломать» функционал, заставить приложение выдавать негативные результаты.

Далее, требования по безопасности приложения могут содержаться в ТЗ, либо выводиться из ТЗ косвенно посредством здравого смысла. Примером таких требований могут являться

защита веб-форм от кросс-скриптинга, или защита от ввода спецсимволов в текстовые поля (если поле определено в ТЗ как текстовое – в большинстве случаев логично ограничить ввод спецсимволов, даже если это явно не указано в ТЗ). С точки зрения качества тестирования такие требования к безопасности приложения всегда стоит выписать в явном виде (особенно если их нет в ТЗ) и заранее запланировать средства проверки этих требований при тестировании приложения.

В общем, с точки зрения детализации задач тестирования, обычно стоит сделать следующее:

- учесть дополнительные трудозатраты при написании test cases на планирование проверки «нестандартных» результатов;
- запланировать трудозатраты на выведение основных требований безопасности (ТЗ + здравый смысл), либо их уточнение (если они уже есть);
- запланировать трудозатраты на переформулировку требований безопасности в сценарии проверки их выполнения;
- явно определить задачи по проверке выполнения требований безопасности при тестировании тех элементов приложения, где эта самая безопасность присутствует.

Как можно заметить, и здесь все задачи, кроме последней, могут быть легко пропущенными при оценке трудозатрат на тестирование. Что именно и в какой форме потребуется в конкретном проекте предстоит решить Менеджеру этого конкретного Проекта.

3. Нагрузочное тестирование

С точки зрения детализации нагрузочного тестирования можно выделить следующие типичные задачи:

- разработка сценария(ев) нагрузочного тестирования – сценарий описывает как именно будет выполняться повышение нагрузки, какие средства контроля поведения приложения будут применяться и т.п.
- подготовка инструментария – для проведения нагрузочного тестирования часто применяется специализированное ПО как для создания нагрузки, так и для контроля состояния приложения под нагрузкой.
- подготовка среды – задача должна быть определена в рамках спецификации сред и внесена в план проекта, но не лишним будет это проверить
- выполнение тестов и обработка результатов

4. Регрессионное тестирование

Обычно детализируется в соответствии с уже реализованным функционалом, на задачи вида «проверить функцию А». Так как для проведения регрессионного тестирования часто используются те же test cases что и при функциональном тестировании – можно просто скопировать задачи.

5. Тестирование интерфейсов

По этому виду тестирования замечу только один момент: при тестировании веб-приложения в разных браузерах – стоит выбрать основной браузер, в котором будут выполняться все test cases, и выделить отдельные задачи на проверку вида и поведения интерфейсов в остальных используемых браузерах. В случае если все test cases надо проверять во всех браузерах – время на выполнение test case можно смело умножать на количество браузеров. Кстати, аналогичная ситуация с тестированием исполняемых приложений в разных ОС.

Глава №16. Часть 2.

В которой речь пойдёт про действия Менеджера Проекта по детализации тестирования.

Фактически, в части 1 я кратко прошелся по основным понятиям, используемым при планировании тестирования. Теперь о том, что стоит, и что не стоит делать Менеджеру Проекта.

Для начала стоит поговорить с начальником группы тестирования (если он есть), либо ведущим тестировщиком (если он есть), либо с единственным или обоими (если тестировщиков трое – один должен быть либо ведущий либо начальник) тестировщиками. Поговорить следует о том, как он видит процесс тестирования версии, как оценивает необходимость создания тест-плана, разработки сценариев тестирования и написания test cases. По результатам данного разговора необходимо определить, какие средства из перечисленных в части 1, и, возможно, дополнительных средств, стоит использовать в данном конкретном проекте. После чего Менеджер Проекта может поставить своему собеседнику задачу: совместно с группой тестирования разработать и предложить вариант детализации плана тестирования, соответствующий выбранным средствам. В описании варианта должны быть представлены детализированные задачи тестирования + оценки трудозатрат.

Когда предложение от тестировщиков готово – Менеджеру Проекта стоит собраться вместе с ними и обсудить это самое предложение. Задача МП – посредством внимательного выслушивания участников и задавания наводящих вопросов удостовериться, что предложенный вариант достаточно полный и всех устраивает. Именно для этого и написана часть 1 – чтобы МП понимал о чём идёт речь и в каких случаях какие средства применяются. После чего можно смело обновлять план проекта и отпускать тестировщиков работать над своими задачами.

Чего Менеджеру Проекта делать не стоит – так это пытаться определить, как и что делать тестировщикам, за самих тестировщиков. В большинстве случаев у тестировщиков опыта и квалификации больше, чем у МП, так что к их рекомендациям следует прислушаться. Тут важно понимать ещё и такой момент, что достаточно опытный тестировщик может выполнить тестирование без написанных test cases и сценариев тестирования, причём более качественно, чем новичок вооруженный всеми средствами, сценариями и планами.

Глава №17: Про ежедневный учёт и контроль

В ролях - Менеджер Проекта, команда проекта

Артефакты – План Проекта

Суть – Менеджер Проекта следит за ходом работ в проекте

Теперь все участники проекта заняты своими прямыми обязанностями – разработчики пишут код, тестировщики готовят test cases, бизнес-аналитик, если он есть, помогает тестировщикам. Что же остаётся делать Менеджеру Проекта? А остаётся ему ежедневно следить за выполнением работ, своевременно замечать все потенциальные проблемы и, при необходимости, перераспределять усилия команды нужным образом. Как именно это стоит делать – рассмотрим на примерах:

«Один день из жизни Менеджера Проекта во время разработки версии – светлая сторона»

Утро

Первым делом с утра можно проверить входящую корреспонденцию. А вдруг Заказчик чего написал интересного? Например, ответил на заданные ему ранее уточняющие требования вопросы. Такую информацию далее надо будет передать разработчикам.

Затем, следует лично пообщаться с командой проекта. Хорошо, когда все сидят в одной комнате, иначе – нужно всех обойти и поговорить с каждым. В процессе (неформального) общения нужно выяснить следующие моменты:

- каждый знает, что ему нужно делать сегодня;
- наличие новых вопросов по текущим задачам;
- успеваемость согласно Плана Проекта.

Если есть новые вопросы – нужно разобраться, в чём суть, и по возможности найти ответ самостоятельно. Если на вопрос нельзя ответить самостоятельно – оценить степень критичности и отправить вопрос заказчику. В общем, с вопросами в процессе разработки поступаем так же, как и на подготовительном этапе. С той разницей, что задержка с получением ответа на критичные вопросы в процессе разработки может сильно влиять на сроки – о чём обязательно сообщаем Заказчику.

Если выполнение задач идёт с опозданием – нужно, опять же, разобраться в чём причина. Чаще всего причиной являются технические проблемы и недостаток квалификации разработчиков. Для компенсации таких рисков у нас есть «буферы» времени, но также можно применять и другие инструменты – перераспределять задачи между разработчиками, привлекать для консультаций сторонних экспертов и т.п. Если причиной отставания является критический риск – требуется принимать срочные меры, и хорошо, если они предусмотрены в списке рисков проекта, определённом на подготовительном этапе. Кстати, своевременное обнаружение проблем является важнейшей частью ежедневной работы Менеджера Проекта. А про действия при обнаружении критичных рисков речь пойдёт далее, в отдельной главе.

Если все знают что им делать, задачи выполняются во время и вопросы все решаются без обращений к Заказчику – есть подозрения что проект может стать успешным 😊

День

Далее, в течение рабочего дня, Менеджер Проекта может заниматься:

- ответами на вопросы команды, возникающие по ходу разработки
- общением с Заказчиком (про это будет глава 19), в том числе выяснением ответов на вопросы у Заказчика
- всеми остальными задачами, такими как обновление документации проекта, участие в тестировании, кадровые вопросы, и многое другое

Вечер

Вечером рекомендуется провести ещё один, контрольный, «обход» команды проекта с целью выяснения статуса текущих задач – все ли всё успели, какие есть проблемы и вопросы, все ли знают задачи на завтра и т.п.

В общем, в течение рабочего дня я настоятельно рекомендую Менеджеру Проекта провести как минимум один, а лучше два раунда общения с командой проекта для выяснения знания своих задач и наличия вопросов/проблем.

«Один день из жизни Менеджера Проекта во время разработки версии – тёмная сторона»

Ещё одним важным нюансом ежедневной работы Менеджера Проекта является сбор информации про личные планы участников команды проекта. Если говорить проще – МП должен первым (или вторым после начальника отдела) узнавать о том, что разработчик:

- себя плохо чувствует и может завтра остаться дома
- планирует взять отгул
- планирует взять отпуск
- может вскорости уволиться

Отгулы, больничные, отпуска, а тем более увольнения непосредственно влияют на план проекта и отражаются на сроках сдачи версий. Чем раньше Менеджер Проекта узнает о предстоящем событии такого рода – тем проще и лучше можно подготовиться к минимизации связанных с ним рисков. Собственно, вариантов минимизации рисков тут только два – перераспределение задач между оставшимися членами команды или привлечение сторонних разработчиков. Оба варианта требуют дополнительного времени – на переключение между задачами и изучение проектной информации. Поэтому, ещё раз напишу, чем раньше МП узнает про планируемую недоступность разработчика – тем лучше.

Бывает и так, что разработчик «выпадает» из проекта неожиданно. Типичные причины – болезнь, или непредвиденные семейные проблемы. Такие ситуации Менеджер Проекта вынужден решать «по ходу пьесы». Но, замечу, что вовремя заданный вопрос про самочувствие или «печальный вид» половину таких неожиданностей может снять. И вот тут переходим к следующему моменту.

Менеджер Проекта в команде проекта должен являться Тимлидом, и в его задачи попадает контроль мотивации и настроения членов проектной команды. А проще говоря, МП должен знать «чем живут» члены его команды. Какое у кого настроение, с чем это связано, кто доволен своей работой, а кто нет и почему, у кого какая ситуация в семье, у кого когда ДР (и не забыть поздравить), у кого бабушка давно и тяжело болеет, кто жениться собрался, а кто разводиться - и т.п. Многие скажут, что в работу Менеджера Проекта описанные «вещи» не входят – но я не соглашусь. Во-первых, от мотивации сотрудников напрямую зависит успешность выполнения рабочих задач. А во-вторых, как уже говорилось выше – своевременная информация про возможное отсутствие весьма важна для планирования.

Как и обещал, подробнее про мотивацию писать не буду – так что пока это «тёмная сторона». Полагайтесь на свою интуицию и опыт. Замечу только, что хороший МП строит отношения с командой таким образом, что разработчики предупреждают его о своих отсутствиях либо личных факторах, затрудняющих рабочий процесс, заранее и самостоятельно

Summary: ежедневно Менеджер Проекта должен проверять:

- наличие задач у каждого участника команды
- выполнение задач согласно ПП
- наличие вопросов
- наличие проблем
- настроения коллектива в целом и каждого его участника в частности
- планы участников команды касательно отсутствия на работе

PS: А ещё, Менеджеру Проекта стоит следить за собственным моральным состоянием и мотивацией. И помнить, что эмоциональное состояние начальника легко передаётся подчинённым. Иногда лучше перепоручить часть обязанностей кому-то другому, или взять пару дней отгулов, чем лечить депрессию не только у себя но у всей команды проекта. Впрочем, это вообще «изнанка тёмной стороны»...

Глава №18: Про еженедельные собрания

В ролях - Менеджер Проекта, команда проекта

Артефакты – План Проекта

Суть – Менеджер Проекта проводит периодические собрания с командой проекта с целью актуализации Плана проекта

Ежедневная работа Менеджера Проекта является, по сути, тактическим руководством проектов. В то же время не стоит забывать о стратегическом руководстве и планировании. А это значит, что реальное положение дел должно отображаться на плане проекта.

В идеале, обновление Плана Проекта нужно выполнять ежедневно. Однако, в большинстве реальных проектов это почему-то не получается – «текучка» отнимает немало времени и сил, так что

обновление ПП постоянно откладывают. Поэтому хорошим, и вполне реализуемым вариантом является регулярный еженедельный обзор успеваемости и соответствующее обновление ПП.

Для этого, с самого начала проекта нужно договориться с командой о том, в какой день будет проводиться собрание посвященное обновлению плана. После чего, если вы пользуетесь аутлуком (или другим ПО в котором есть напоминки) – можно сразу завести повторяющийся аппойнтмент на выбранный день и время, в описании которого перечислить стандартные задачи собрания. Таковыми являются:

- проверка выполнения запланированных задач
- отражение в Плане Проекта реального статуса по всем задачам
- сверка сроков сдачи версий в соответствии с реальным положением дел
- возможное перепланирование будущих задач

В общем, как и для любых других собраний – все участники должны заранее знать когда оно состоится и чему будет посвящено. Хорошими днями для еженедельных собраний являются четверг (вечер) и пятница (первая половина дня). Пятница вечер – плохое время для любых собраний кроме пятничных (не говоря уже про сборку версий и т.п.).

В указанный день и время команда проекта собирается вместе и под руководством своего менеджера смотрит на открытый План Проекта, обсуждает текущее состояние дел, и дальнейшие планы. По ходу обсуждения Менеджер Проекта отмечает на ПП выполненные задачи (Рекомендую по очереди «пройтись» по всем задачам которые были в работе за прошедшую неделю), при необходимости вносит изменения в план (незапланированные ранее отсутствия разработчиков, срыв сроков по выполненным задачам как в плюс так и в минус, перестановки в очередности следующих задач – всё это нужно отражать в ПП). По результатам проведенного обзора – смотрим, как изменились сроки сдачи версий и проекта в целом. Если не изменились – хорошо. Если меняются – вспоминаем рекомендации предыдущих глав... Главная из которых – при исчерпании буферов «запасного» времени и подвижках сроков нужно сразу «бить в набат» и пересогласовывать сроки с Заказчиком. Надежда «на авось» выручает крайне редко. Затем, обновлённую версию Плана Проекта нужно не забыть выложить на своё стандартное (общедоступное) место.

По результатам собрания все участники должны понимать реальное положение дел на сегодняшний день, общую успеваемость и (снова «тёмная сторона») укрепить свой настрой на успешное выполнение проекта. Кстати, знание реального положения дел и планов «что мы сделаем, чтобы все было ОК» само по себе укрепляет настрой даже в кризисных ситуациях. Врать сотрудникам «что всё хорошо» - плохо, тем более что они сами чувствуют как есть на самом деле.

Кроме этого, результаты собрания необходимы Менеджеру Проекта для составления отчетов для Заказчика, о чем речь пойдет в следующей главе.

Да, и еще, если в вашей компании принято писать еженедельные отчёты по статусам проектов для руководства – полученной на еженедельном собрании информации будет ещё оно полезное применение.

Глава №19: Про информирование Заказчика

В ролях - Менеджер Проекта, Заказчик

Артефакты – План Проекта, отчёт Заказчику

Суть – Менеджер Проекта информирует Заказчика о ходе разработки

О пользе постоянного и регулярного общения с Заказчиком на всём протяжении проекта написано немало статей и других материалов. Некоторые методики руководства проектами (agile) даже построены на основе ежедневного общения. Суть, в общем случае, сводится к тому, что Заказчик должен быть в курсе всего происходящего в проекте. Чем меньше обманов и скрытой информации – тем больше доверия и выше шансы на успех проекта. Идеальный Заказчик поймёт любые возникшие трудности, и даже поможет в их разрешении.

Однако в реальности приходится учитывать неадекватность некоторых заказчиков, возможные «некрасивые» накладки на стороне разработчика, которые решаются в рабочем порядке, и прочий «политес». Так что будем реалистами, и попробуем охватить только ту информацию, которую стоит сообщить Заказчику в любом случае.

События, которые приводят к изменению сроков сдачи версий в большую сторону

Таковыми являются:

- незапланированное отсутствие ресурсов (реализация человеческих рисков) – болезни, отгулы, увольнения разработчиков
- неожиданные технические проблемы (реализация технических рисков) – «оказалось что с помощью библиотеки X этот функционал реализовать корректно нельзя» или «компонент Y имеет собственный баг, который мешает реализации наших задач» и т.п., обычно связано с использованием сторонних компонент и фреймворков
- недооценка трудозатрат (риски планирования) – разработка функции Z потребовала не 40, а 80 человеко-часов
- срыв сроков поставки (риски взаимодействия) – например Заказчик не прислал во время дизайн, из-за чего простаивает разработка и сдвигаются сроки
- реализация всяких других рисков

В общем, речь идет о реализации рисков, которые не покрываются «буферным» временем и не могут быть минимизированы имеющимися средствами, а значит с высокой вероятностью сроки сдвинутся.

В таких случаях, Заказчик должен узнавать о переносе сроков сдачи версий чем раньше тем лучше. Один из вариантов обоснования – получение информации о переносе сроков обычно вызывает негативную реакцию, и чем раньше она «пройдет» - тем лучше, к моменту сдачи версии желательно чтобы Заказчик был «спокоен». Есть и другие варианты, например своевременное информирование о проблемах в спокойном и вежливом ключе увеличивает «вовлеченность» Заказчика в разработку и облегчает коммуникации. Кстати, кто из читателей предложит свой вариант?

Типичной ошибкой в управлении проектами является попытка «скрыть» от Заказчика реализованные риски, которые приводят к срывам (особенно незначительным) сроков сдачи, и надежда «отыгаться в овертаймах». Когда Заказчик узнаёт о том, что версия не готова, в день сдачи (или за день-два до сдачи) – он обычно расстраивается, и глубина расстройства растёт с каждым днём задержки, что приводит к сложным коммуникациям и проблемам с приёмкой готового функционала. Кроме того, «гонки» сильно утомляют и понижают мотивацию разработчиков. В целом, люди любят fair play – помните об этом.

Так что, повторюсь в очередной раз – если вы видите что сроки (почти) гарантировано «съезжают», пусть всего на 1 день – лучше сразу сообщить об этом Заказчику спокойным и вежливым образом, например так:

«Ув. Заказчик. Мы тут при разработке столкнулись с проблемой X, по факту разбирательства с которой мы с высокой вероятностью ожидаем смещение сроков сдачи версий таким образом _табличка_. Надеюсь на ваше понимание и поддержку, мы со своей стороны делаем всё возможное для минимизации проблемы и постараемся сократить отставание при дальнейшей работе».

Неясность требований

Если во время разработки вдруг выяснилось, что описание некоторого функционала является недостаточно чётким для его реализации, либо противоречиво, либо может быть интерпретировано разными способами, и выяснение дополнительной информации обязательно требует участие (представителей) Заказчика – фактически можно сказать что произошла реализация риска. Поступать рекомендую так же как написано выше – то есть, не откладывая уведомить об этом Заказчика, и проакцентировать необходимость получения разъяснений для продолжения разработки. Также можно намекнуть про возможность сдвига сроков сдачи версий симметрично задержке в получении ответа. Но тут следует понимать, что если на этапе подготовки к старту Заказчик не ограничивал время на анализ документации, то наличие таких «непоняток» является скорее виной Разработчика, поэтому давить на Заказчика особо не стоит. А вот если Заказчик форсировал старт проекта и не дал достаточно времени на анализ документации, либо проект стартован с «сырой» документацией с согласия Заказчика, либо вы работаете по agile процессу – то в таких случаях ответственность Заказчика за сроки нужно подчеркивать без лишней скромности.

Текущее состояние выполнения задач

Информировать Заказчика о (серьёзных) проблемах нужно безотлагательно, но следует понимать что такая информация носит негативный характер. Поэтому, для сохранения «баланса», важно также сообщать Заказчику о достигнутых успехах, и делать это стоит регулярно.

Хорошей практикой является еженедельная отправка Заказчику письменных отчётов о текущем прогрессе в разработке. В отчёте кратко описываем, какие задачи были выполнены за прошедшую неделю, рисуем красивые картинки (актуальная диаграмма Ганта) или таблички (тасклист с отметкой выполненных и текущих задач). Если были проблемы –

спокойно описываем что произошло и как порешали. Обязательно в отчёте должен быть представлен актуальный (на день создания отчёта) план сдачи версий (milestones).

Детальные и информативные отчёты, кроме регулярного сообщения Заказчику успехов Разработчика, играют ещё одну очень важную роль – это сводная история развития вашего проекта. Наличие такой истории весьма способствует адекватности понимания текущего состояния проекта всеми сторонами, а также весьма упрощает возможные «разборки» в случае конфликтных ситуаций.

Так что, рекомендую писать Заказчику еженедельные отчёты даже если он их не просил, даже если вы ежедневно плотно общаетесь, и нет никаких «непоняток» - потому что отчёты это ваша страховка и (часто единственный) сводный текстовый источник информации о проекте.

Резюме главы:

1. Сообщать Заказчику о проблемах которые сдвигают сроки надо чем раньше тем лучше
2. Сообщать Заказчику об успехах нужно регулярно
3. Писать еженедельные отчёты Заказчику очень полезно

Глава №20: Подготовка к сдаче версии в тестирование

В ролях - Менеджер Проекта, команда проекта

Артефакты – план проекта, требования, release notes

Суть – проверка готовности к сдаче версии в тестирование

Предположим, в рамках версии предполагалось воплотить 10 функций/фич (features). Разработчики сообщают менеджеру проекта, что все сделано. Что дальше?

В первую очередь, нужно чтобы разработчики сами проверили, что написанный код работает и заявленные требования реализованы в полном объёме. Большим подспорьем в этом деле является использование Unit Tests в разработке. Успешное прохождение всех Unit Tests обычно показывает что «всё хорошо». Если юнит-тестирование в разработке не применялось (ил применялось только частично) – рекомендую менеджеру проекта поставить задачу ведущим разработчикам лично ещё раз перепроверить написанный код и соответствие сделанного требованиям.

Следующим рекомендуемым шагом является проверка реализованного функционала самим менеджером проекта. Задача проста – открыть приложения и «пробежаться» по функционалу. Основное внимание следует обращать на соответствие сделанного требованиям версии и отсутствие критических ошибок. Нужно это для страховки от невнимательности разработчиков – а это нередкое явление, разработчики работают с приложением ежедневно и могут какие-то моменты просто не замечать или считать чужой зоной ответственности.

Если всё ок, и на первый взгляд всё работает в соответствии с требованиями – переходим к шагу «пишем Release Notes». Release Notes – это такой документ, в котором кратко перечислен функционал реализованный в рамках текущей версии (и предыдущих тоже), отмечены известные недостатки (на ранних версиях бывают баги и нерабочий функционал, который не затрагивает ключевые функции этой версии но может вызвать вопросы у тестировщиков и Заказчика), указаны разработчики ответственные за каждую функцию. Задача этого документа – дать тестировщикам (и в дальнейшем Заказчику) понимание что тестировать, а на что не обращать внимание, и к кому обращаться за пояснениями или кого ставить ответственным за исправление дефекта. Кроме этого, Release Notes может содержать массу дополнительной информации, как то:

- спецификация среды запуска
- перечень используемых сторонних компонент
- руководства по установке приложения
- help, FAQ

Формат документа Release Notes бывает разным. Примеры:

<http://www.mozilla.com/en-US/firefox/3.6/releasesnotes/>

<http://readysset.tigris.org/nonav/templates/release-notes.html>

Написание Release Notes редко занимает более часа, сделать это может сам менеджер проекта или ведущий разработчик. Использование этого документа экономит не только время на выяснение «что сделано» между тестировщиками и разработчиками, но и нервы (что немаловажно с точки зрения нормальных отношений в проектной команде). Так что, рекомендую.

После того, как новая версия проверена у разработчиков и менеджером проекта, и написаны release notes – можно «отдавать» версию в тестирование. В зависимости от того, что и как вы разрабатываете, процедура передачи может быть разной. Например:

- дать доступ к папке с готовым инсталлируемым приложением
- поставить приложение на выделенный сервер тестирования (требует привлечения либо разработчиков, либо админа проекта)
- дать доступ к серверу разработку тестировщикам (если для тестирования не используется выделенная среда)
- отправить код и инструкции по установке группе тестирования (если тестированием занимается отдельная команда и установку они производят сами, не забудьте написать инструкции...)

В любом случае, не забудьте приложить Release Notes ;-)

Глава №21: Тестирование и исправление дефектов

В ролях - Менеджер Проекта, команда проекта

Артефакты – требования, план тестирования, issues

Суть – Устранение выявленных дефектов

Как именно проводить тестирование – я писать не буду. На эту тему есть масса толковой и не очень литературы, Гул в помощь. Остановлюсь лишь на основных моментах, важных с точки зрения управления проектом.

1. Хорошо, когда есть план тестирования, в котором описаны какие виды тестирования надо применять, и как именно это делать. Про план тестирования я уже писал в предыдущих главах. Основная польза от такого плана – «прозрачность» текущего состояния тестирования для менеджера проекта, удобство в планировании сроков и трудозатрат.
2. Важной штукой является среда учёта дефектов a-la Jira. Когда все дефекты собраны в одном месте, с проставленными статусами и указанием ответственных за них тестировщиков и разработчиков – это реально удобно. Время, потраченное на установку такой среды, с лихвой компенсируется временем, сэкономленным при поиске информации по дефекту и уточнении этой информацией, кроме того нервы тоже надо экономить. Чем прозрачнее процесс – тем проще жить всем его участникам.
3. В любом случае, какую бы среду учёта дефектов вы не использовали, описание дефекта должно содержать следующую информацию:
 - a. Кто обнаружил дефект
 - b. С какой функцией связан дефект (ссылка на раздел требований)
 - c. Описание шагов воспроизведения дефекта
 - d. Скриншот ошибки - желательно
4. Даже если вы не пользуетесь никакой специализированной средой учёта дефектов – список дефектов должен быть полным. Ни в коем случае не стоит принимать и разбирать дефекты по отдельности – иначе легко потерять общую картину и поиметь противоречия между дефектами.
5. Тестирование и устранение дефектов следует вести в виде дополнительной итерации. А именно:
 - a. Некоторое время идёт тестирование, затем тестировщики представляют сводный список дефектов.
 - b. Тестирование останавливается, разработчики устраняют дефекты и готовят новую версию
 - c. К новой версии неплохо бы обновить release notes – добавить информацию о том, какие именно дефекты устранены в этой новой версии
 - d. Новая версия отдаётся снова в тестирование
 - e. См. П. а
6. При получении списка дефектов обязательным действием является приоритизация этого списка. А именно, менеджер проекта или ведущий тестировщик должны обозначить в явном виде степень критичности каждого дефекта и очередность его устранения для разработки.
7. Важно установить некоторый порог критичности дефектов – так чтобы дефекты с критичностью ниже X не подлежали обязательному устранению до показа версии Заказчику. Помним о том, что в release notes есть такой раздел как known issues, в котором можно перечислить всякие мало существенные мелочи. Порог критичности

устанавливается менеджером проекта по своему усмотрению, исходя из отношений с заказчиком и важности затронутых функций приложения. Если такого порога критичности нет – тестирование может очень сильно затянуться.

8. По завершению тестирования следует обновить документ Release Notes. Обновлённая версия понадобится вам при выпуске следующей версии, а также как источник информации при сдаче версии Заказчику. В идеале, Release Notes хранит историю разработки и тестирования функционала, а также исправления дефектов, на всём протяжении проекта. Именно поэтому пренебрегать этим документом не стоит.
9. По завершению тестирования следует провести собрание с командой проекта с целью обсудить и утвердить готовность версии к сдаче Заказчику. Только после этого, когда ни у кого нет возражений, версия готова к сдаче.

Глава №22: Подготовка к сдаче версии Заказчику

В ролях - Менеджер Проекта, команда проекта

Артефакты – план проекта, release notes, требования

Суть – Проверяем что всё готово

Предположим, что версия готова, проверена, протестирована, и признана командой проекта годной для сдачи Заказчику. Что делать дальше?

Сначала надо установить версию в среду показа Заказчику, проверить доступность версии и записать параметры доступа к ней. Если показ происходит в среде разработки – обязательно проверьте, что к ней есть доступ «извне», т.е. что заказчик сможет пройти по указанной ссылке. Иногда полезно сразу создать для заказчика тестового пользователя. Если вы сдаёте консольное приложение – убедитесь в том, что оно упаковано в общепринятом формате (известно, что формат .RAR не особо распространён за пределами СНГ, лучше использовать .ZIP). Если есть некий help – полезно не забыть его прикрепить к архиву или вывесить на тестовой версии сайта. Если вы отдаёте Заказчику только код, а сборка приложения происходит на его стороне – рекомендую сделать дополнительную проверку сборки и установки, для которой привлечь разработчика, не знакомого с проектом – так сказать, проверить «на кошечках», что с этим делом справится любой грамотный технический специалист. Да, ещё – если вы отдаёте Заказчику консольное приложение, или код, или что-то ещё что нужно скопировать на сторону Заказчика – рекомендую на всякий случай проверить доступность выбранного способа передачи данных, например, доступность FTP-сервера и наличие на нём пользователя для Заказчика, или что электронная почта пропустит файл такого объёма. По результату описанных действий – имеем «пакет» данной версии, собранный и готовый к показу/отправке.

Теперь поработаем над сопроводительной документацией. Начнём с обновления плана проекта. Следует отметить все выполненные задачи, и проследить, что сроки соответствуют реальным. План проекта полезно отправлять Заказчику вместе с показываемой версией, для усиления положительного эффекта «вот смотри, как мы классно работаем!» и наглядного отображения выполненных задач.

Далее, так как план проекта всё-таки не содержит раскрытого описания выполненных задач, а так же учитывая, что у нас есть куча интересной информации в документе Release Notes, где, как известно, должен быть перечислен весь реализованный в версии функционал с привязкой (ссылками) на соответствующие разделы требований – готовим Release Notes в версии для Заказчика. При этом не забываем что документы Заказчику нужно отправлять на том же языке, на котором написана документация ☺ В принципе, если Заказчик не участвует в тестировании – вместо release notes (где, как помним, есть ещё и список дефектов обнаруженных и исправленных в этой версии) можно подготовить некоторый облегчённый вариант документа, в котором присутствует только список выполненных задач (с названиями более понятными чем в ПП, с привязкой к требованиям) и список known issues – известных проблем версии которые либо не смогли устранить, либо считаем не существенными для её сдачи.

Наконец, готовим сопроводительное письмо к данной версии. В письме, кроме слов о том что «Ура! Версия готова можно проверять!», обязательно указываем параметры доступа к версии (URL тестового сайта, логин/пароль для FTP, etc). Иногда полезно в вежливой форме напомнить Заказчику про оговоренные в Контракте сроки на проверку версии (если такое было оговорено) и важность соблюдения этих сроков для своевременной сдачи следующих версий. К письму можно прикрепить план проекта и release notes (или его аналог).

Осталось всё ещё раз перепроверить, убедиться, что ничего не забыто, и отправить это письмо Заказчику. В случае доступности телефонной связи или других прямых каналов общения – рекомендую сообщить Заказчику о готовности версии в том числе и таким путём (письмо по электронной почте – это просто удобный способ собрать всю информацию вместе, общаться лучше вживую).

Кстати, есть ещё один хороший способ начала показа версии. Если есть возможность – устройте презентацию версии на видеоконференции или вживую. Личный показ что и как работает экономит массу времени и страхует от misunderstandings. Благо, современные средства для видео связи легко доступны и просты в использовании.

Дальше – ждём рецензий и комментариев.

Глава №23: Обработка комментариев Заказчика

В ролях - Менеджер Проекта, Заказчик

Артефакты – “wishlist”, changes

Суть – Принимаем комментарии Заказчика, обсуждаем, устраняем дефекты, готовимся к ченджам

При проверке версии Заказчик может обнаружить дефекты, придумать изменения либо просто задавать вопросы «почему так?». На вопросы следует отвечать быстро, без задержек – это помогает Заказчику быстрее закончить проверку. А вот дефекты и запросы на изменения рекомендуется принимать списками, с целью упрощения обработки и отсеивания противоречивых данных на самых ранних этапах. Поэтому, как только получаем первые

дефекты/изменения – сразу просим Заказчика формировать их в список и прислать все вместе по окончании проверки. Как это аргументировать – зависит от фантазии и опыта менеджера проектов, обычно удобство обработки является достаточно весомым аргументом. Исключение стоит делать только для критичных дефектов, наличие которых затрудняет дальнейшую проверку версии – такие дефекты надо принимать и исправлять незамедлительно (вообще их быть не должно после внутренних проверок и тестирования – но все мы не совершенны...). А вот если Заказчик настаивает на срочной реализации изменений – такие попытки следует пресекать сразу. Все изменения должны проходить согласно общей процедуре – сверка с текущими требованиями, оценка трудозатрат, планирование, согласование изменений сроков и бюджета. Даже если изменения совсем незначительные – буквально на 5 минут работы – всё равно, не стоит их делать сразу. Такие пятиминутные изменения имеют свойство накапливаться и в результате выливаться в часы и дни работы, не учтённые в плане и бюджете проекта, что совсем не есть хорошо...

Итак, для начала сразу приушаем Заказчика к хорошему тону – все дефекты и запросы на изменения присылать сводным списком, мгновенно разбираются только критичные дефекты.

Далее, предположим, Заказчик закончил проверку и прислал список своих замечаний/дефектов/изменений. Этот список нужно разобрать – дефекты отдельно, запросы на изменения отдельно. Дефекты заносим в среду учёта дефектов, расставляем приоритеты и приступаем к исправлениям. Тут всё как обычно – по завершению исправлений выпускаем новую версию, просим тестировщиков проверить исправленные дефекты, если всё ок – обновляем версию в среде показа, готовим всю «сопроводилку» заново, и просим Заказчика проверить ещё раз. Не забываем обсудить с Заказчиком и установить нижний порог критичности дефектов – возможно, некоторые мелкие исправления можно сделать уже в рамках следующей версии и не терять на них время при сдаче этой.

Теперь переходим к разбору списка запросов на изменения. Для каждого изменения – change – чендж – следует провести приблизительную оценку трудозатрат. Затем, делаем сводную таблицу ченджей с указанием для каждого ченджа оцененных трудозатрат (лучше даже сразу указывать насколько изменятся сроки, например +1 день к общей длительности) и, отдельно (для наглядности), изменение бюджета проекта, вызванное данным ченджем (особенно актуально для fixed cost проектов). Добавляем в сводную таблицу отдельную, пока пустую, колонку под названием «приоритет». И отправляем результат Заказчику с просьбой утвердить необходимость реализации данных изменений (с учётом влияния на сроки и бюджет конечно), а также проставить приоритеты – что нужно сделать в первую очередь (в данной версии), что можно реализовать в следующей версии, что может подождать до конца проекта (расшифровку значения приоритетов разумно объяснить Заказчику). После чего – ждем ответа.

Важным моментом для успешной разработки является понимание Заказчиком такой простой вещи, как влияние изменений на сроки и бюджет проекта. На моей практике был не один проект, проваленный по причине излишнего «перфекционизма» Заказчика и излишнего внимания к несущественным изменениям, которые в результате увеличивали сроки и бюджет на треть и более. Объяснять Заказчику эти простые истины нужно неустанно, чётко, ясно, и при каждом сопутствующем поводе. Получили комментарии – сразу указываем, что изменения обрабатываются отдельно и ни в коем случае не «прямо сейчас». Всегда при

обсуждении изменений «подчёркиваем» влияние каждого изменения на бюджет и сроки. Поддаваться на уговоры «ну тут же чуть-чуть, давай прямо сейчас быстренько» очень не рекомендуется – иначе такие «быстрячки» станут нормой на протяжении всего дальнейшего проекта. Ещё один аргумент «почему так делать не стоит» – потому что такие «быстрячки» обычно нигде не регистрируются в требованиях, и в конце проекта уже никто не помнит, почему было сделано это именно вот так.

Менеджер Проекта – будь бдителен!

PS: Учитывая важность грамотного Changes Management для успешности проекта в целом – ему будет посвящена отдельная следующая глава.

Глава №24: Changes менеджмент

В ролях - Менеджер Проекта, Заказчик

Артефакты – changes

Суть – Когда и как мы реализуем запросы на изменения требований в течение проекта

Итак, ещё раз про Changes management, с подробностями и разъяснениями.

Пункт 1. Ченджи – это хорошо!

Бояться запросов на изменения в течение проекта – не надо. Ченджи – на самом деле есть гуд! Потому что ченджи – признак «живого», нужного проекта. Попробую пояснить. Как было замечено в Главе №0, проект начинается с идеи. Далее, по мере обсуждений и размышлений Заказчик излагает своё видение конечного продукта в виде требований. Однако, требования нельзя «потрогать», с ними нельзя «поиграться» как с реальным продуктом. А, как известно, не у всех людей воображение развито достаточно, чтобы представить себе весь продукт целиком во всех возможных вариантах использования. В требованиях даже «картинки», если и есть – статичны. В то же время, визуальный и кинетический (потрогать, поиграться) каналы восприятия являются самыми используемыми для большинства населения нашей планеты. Короче, гарантировать, что требования на 100% отражают именно то, что хочет Заказчик – нельзя.

В качестве примера из жизни – крайне редко ваше представление о том, что вы хотите сделать, соответствует готовому продукту. Идя на пикник, человек представляет себе полянку с костром – абстрактную. Придя на место, в процессе обустройства полянки и разведения костра, эта абстрактная картинка конкретизируется и становится реальностью. Теперь сравните, что вы представляли изначально, и что есть в результате. И найдите 10 отличий ☺

Если идея «живая», продукт реально нужен Заказчику – он продолжает обдумывать её реализацию и после утверждения требований и старта разработки. Возможны ситуации, когда ещё до показа первой версии Заказчику в голову может прийти некоторое важное уточнение функциональности – ничего необычного в этом нет.

При получении первой версии (или прототипа) продукта Заказчик наконец-то может увидеть свою идею «в реале». Тут активируются визуальный и кинетический каналы восприятия и обработки информации. «Пощупав» продукт со всех возможных на данный момент сторон, и сравнив со своим представлением (в том числе подсознательным) о конечном продукте, Заказчик наверняка придумает массу различных пожеланий по его улучшению и развитию. И это хорошо.

С каждой новой версией продукта вы показываете Заказчику новую функциональность – а значит, он может увидеть и «пощупать» продукт с новых сторон. И снова придумать массу улучшений... И это тоже хорошо.

Стремление Заказчика улучшать свой продукт время разработки – это нормальное проявление нормального человеческого поведения при создании чего-то нового. Наличие ченджей – признак «живого» проекта, реально нужного и востребованного. Поэтому, ещё раз, не надо их бояться. Бояться стоит проектов, где ченджей вообще нет – скорее всего, такой проект никому не нужен, а значит, чреват неожиданными сюрпризами (отказ о продолжения разработки, проявление конфликтов на стороне Заказчика мешающих разработке и т.п.)

Пункт 2. Только в рамках бюджета!

Мы договорились, что сам факт того что Заказчик засыпает нас ченджами на всём протяжении разработки – это хорошо. Однако нужно ли все эти ченджи реализовывать, и если да – то когда и как?

В первую очередь, крайне важно объяснить Заказчику, что реализация ченджей (всех!) происходит только на платной основе. Реализация ченджей требует дополнительных трудозатрат, не заложенных в бюджет на этапе старта проекта. Эти трудозатраты должны быть оплачены. И выполнение ченджей должно происходить в соответствии с общим планом разработки, который после получения ченджей, их оценки и утверждения изменений в бюджете обновляется соответствующим образом и заново утверждается с Заказчиком. Всё это должно быть прописано в Контракте, или хотя бы оговорено в устной форме до старта проекта. Сам факт того, что за ченджи надо платить – не должен быть сюрпризом для Заказчика, это ответственность менеджера проекта и сейлзов.

Кроме получения вознаграждения за выполненную работу, утверждение бюджета на ченджи несёт в себе дополнительный смысл – ограничивает Заказчика от бесконечного перфекционизма. Некоторые Заказчики имеют свойство уделять большое внимание доведению продукта до совершенства (в их понимании). При этом часто одни и те же функциональные элементы продукта могут переделываться по несколько раз, а «перекраска» кнопочек и оформления становится ежедневной рутинной. С точки зрения бизнеса (да и вообще, создания новых продуктов) можно с уверенностью сказать, что ни один новый продукт не бывает совершенным. Момент выпуска нового продукта «в массы» следует выбирать тогда, когда его запуск будет наиболее выгоден с точки зрения маркетинга (для принципиально новых продуктов – чем раньше, тем лучше), необходимым условием является только стабильная работа ключевого (инновационного) функционала этого продукта. Занимаясь «перфекционизмом» Заказчики могут упустить время для успешного запуска

своего продукта, либо «проспать» выход аналогичного продукта конкурентов. Так как, в идеальном случае, менеджер проекта «играет на стороне Заказчика» - помочь Заказчику отделить ненужный перфекционизм от необходимых улучшений может быть и его задачей.

В таком контексте, чётко реализуемое условие оплаты за все (даже 5-минутные) ченджи является, безусловно, полезным, как для компании-Разработчика, так и для самого Заказчика. А вот попустительство со стороны менеджера проекта и реализация мелких ченджей вне бюджета такому вредному перфекционизму только способствует.

Пункт 3. О здравом смысле и учёте особенностей проектов

Из каждого правила есть исключения. Которые, впрочем, только подтверждают правило.

В некоторых случаях, например, при работе по первому проекту с важным для компании Заказчиком, или при работе с давно проверенными Заказчиками, некоторые ченджи могут быть реализованы вне бюджета. Речь идёт о небольших уступках, на которые идёт менеджер проекта для улучшения отношений с Заказчиком. Цель таких уступок вполне очевидна, но крайне важно сделать всё таким образом, чтобы в результате Заказчик не «сел на шею» и не начал воспринимать подобные уступки как норму, а также рассчитывать на расширение объёма бесплатных работ.

Правильным подходом является чёткое разъяснение Заказчику сути уступок, на которые мы идём, и акцентирование стандартных правил работы. Например:

«Учитывая важность этого пилотного проекта для нашего сотрудничества и незначительность трудозатрат по сравнению со временем, необходимым для утверждения изменений к бюджету, мы сделаем эти изменения просто так. Но, это не является стандартной практикой нашей работы, и мы не можем гарантировать, что в следующий раз сможем поступить также».

Таким образом, вы не только проясняете ситуацию, но и подчёркиваете значимость Заказчика для вашей компании, что ему обычно приятно. Заказчик должен знать и понимать стандартные правила совместной работы (для этого их полезно прописывать в контракте или обсуждать и утверждать в другой форме до начала работ по проекту). В случае уступок – Заказчик должен понимать, что это уступки.

Кстати, обычной ошибкой молодых/неопытных менеджеров проектов является предоставление Заказчику уступок без его уведомления о том, что это уступки. Заказчик, что логично, воспринимает такой подход как стандартную практику для всех проектов. И когда в продолжение проекта, либо в следующих проектах, менеджер начинает «закручивать гайки» - Заказчик воспринимает это негативно.

И ещё один момент – про различие акцентов в проектах типа fixed cost и time&material. Для fixed cost проектов при согласовании/утверждении ченджей акцент делаем на изменение бюджета проекта (мы же не хотим работать бесплатно?), затем на сроки. В проектах типа time&material Заказчик и так оплачивает все трудозатраты, поэтому акцентируем изменения в сроках в первую очередь.

Пункт 4. Заметки по очередности реализации изменений

Вопрос – в каком порядке следует реализовывать ченджи? Обычно, этот порядок определяется в соответствии с приоритетами, расставленными Заказчиком. И реализация изменений планируется «между» версиями, либо в течение следующей версии (в зависимости от удобства разработки). Но есть такие ченджи, которые надо делать незамедлительно – это ченджи, значительно влияющие на архитектуру проекта и требующие переделки уже сделанного.

Примем неизбежное – такие ченджи бывают, и не так уже и редко как хотелось бы. В некоторый момент Заказчика может осенить некоторая мега-идея, которая меняет функции продукта решительным образом. И Заказчик будет готов к соответствующим изменениям в бюджете и сроках ради воплощения этой новой идеи. Что делаем в таких случаях?

В первую очередь, хочу вспомнить правила распределения реализации функционала продукта по версиям. Правило №1 - функции, которые определяют архитектуру всего приложения, следует реализовывать чем раньше, тем лучше. Проще говоря, сначала надо заложить фундамент, а потом строим стены. Правило №2 – в первую очередь нужно реализовать те функции, которые наиболее важны Заказчику, то есть обеспечивают реализацию бизнес-идеи проекта. Чем раньше Заказчик сможет «пощупать» ключевой функционал – тем раньше он его утвердит либо придумает что-то кардинальное.

В случае, если одновременное выполнение обоих правил невозможно, я рекомендую сделать выбор исходя из оценки предрасположенности Заказчика к мега-идеям. Это, в первую очередь, дело опыта менеджера проекта, хотя есть и общие закономерности. Внесение кардинальных изменений в функционал чаще происходит в небольших инновационных проектах, связанных с разработкой новых продуктов на развивающихся рынках. Разработка приложений для оптимизации бизнес-процессов в серьёзных компаниях менее склонна к таким изменениям. Но, всё же, личность Заказчика является более весомым фактором.

В общем, правильное планирование очередности задач разработки значительно облегчает реализацию экстренных ченджей, да и ченджей вообще.

Если вы получили от Заказчика запрос на изменение, серьёзно затрагивающий уже реализованные функции и архитектуру приложения, сделайте следующее:

1. Рассмотрите чендж с разработчиками и оцените масштаб переделок (грубо)
2. Рассчитайте соответствующее изменение к бюджету и срокам проекта (умножив данные от разработчиков на 1,5 - 2 чтобы учесть дополнительные затраты на обновление требований, плана проекта и т.п.)
3. В максимально ясной и корректной форме изложите полученные данные Заказчику
4. Если Заказчик подтверждает готовность к таким жертвам – остановите разработку
5. Проведите полное перепланирование работ, как будто проект начинается заново
6. Утвердите с Заказчиком новый бюджет и сроки
7. Стартуем работы

То есть, при получении кардинальных ченджей, надо сначала объяснить Заказчику суть кардинальности, и если он согласен с дополнительными трудозатратами – надо фактически стартовать проект заново.

Кстати, в подобных случаях, иногда, можно поступить и другим образом. Необходимое изменение реализуется «на костылях», то есть способом, требующим минимальных трудозатрат (и минимума изменений в архитектуре) на данном этапе. При этом обычно страдает производительность и стабильность конечного продукта. Такой подход применим в случае, если мы хотим показать Заказчику работу этой функции для утверждения «так верно», с целью предотвращения дальнейших изменений. Короче, делаем что-то вроде прототипа. И, в дальнейшем, мы рассчитываем на оптимизацию архитектуры при рефакторинге кода в конце проекта. Если точнее, то поступать таким образом можно, если:

1. Нет уверенности, что Заказчик окончательно определился, что он хочет, есть высокие шансы, что когда он увидит результат – решит оставить всё как было
2. Рефакторинг кода в конце проекта реален и не вызовет возмущений от Заказчика. Например, первая версия может выйти в тестовое использование «на костылях», а далее будет разработка второй версии для массового использования с учётом результатов тестового периода и с оптимизацией производительности

Во всех других случаях такой подход противопоказан, если конечно для вас важны отношения с Заказчиком после формального завершения проекта.

Пункт №5. Общий цикл

Наконец, ещё раз «нарисую» общий цикл работы над ченджами:

1. Получаем информацию от Заказчика
2. Оцениваем критичность изменений
3. Оцениваем приблизительные трудозатраты и, как результат, влияние на бюджет и сроки
4. Обсуждаем изменения бюджета и сроков с Заказчиком
5. По утвержденным чендгам – просим Заказчика определить приоритеты и желательные сроки реализации
6. Утвержденные изменения вставляем в план работ
7. Обновляем схему платежей – добавляем затраты на ченджи
8. Обновлённый план работ и схему платежей утверждаем с Заказчиком
9. Приступаем к реализации согласно ПП ☺

Глава №25: Приёмка версии

В ролях - Менеджер Проекта, команда проекта, Заказчик

Артефакты – требования, план проекта, акт приёмки

Суть – Получаем подтверждение приёмки версии от Заказчика

После того, как в готовой версии устранены все критичные багги и выполнены критичные изменения, и Заказчик признал, что реализованный функционал соответствует требованиям и плану проекта, необходимо получить от Заказчика формальное подтверждение приёмки версии. В зависимости от условий Контракта (или других предстартовых договорённостей) сделать это можно по-разному: Заказчик может подписать акт приёмки версии (лично или прислать скан), либо достаточно подтверждения приёмки в email. Важен сам факт признания Заказчиком версии «годной», а значит, возможности старта работ по следующей версии.

Вне зависимости от способа получения подтверждения приёмки, есть следующие условия, которые рекомендуется соблюдать:

1. Важно указать номер сборки (билда) и дату выпуска принятой версии. То есть, не просто «версия норм», а «версия №1234 от 15 марта 2010 норм».
2. Желательно в документе (письме), подтверждающем приёмку версии, перечислить реализованный в этой версии функционал (со ссылками на требования), исправленные дефекты (со ссылками на номера дефектов в Jira) и реализованные изменения (тоже ссылками на номера или письма)
3. Желательно в этом же документе (письме) перечислить те дефекты и ченджи, исправление/реализация которых отложена на следующие версии. Тоже с привязкой к номерам дефектов и ченджей.

Выполнение этих рекомендаций является страховкой от плохой памяти или махинаций Заказчика типа «это не так, все переделывайте, в той версии, что я принял, этот чендж уже был сделан, почему его теперь нет?» (хотя такого ченджа вообще не было).

В простом случае для небольшого проекта приёмка может выглядеть следующим образом. Менеджер проекта пишет Заказчику письмо подобного содержания:

«Уважаемый Заказчик. На данный момент мы обсуждаем версию номер 1234 от 15.03.2010, в которой реализованы следующие требования: _список_

В процессе сдачи версии мы исправили следующие дефекты: _список_

А также реализовали ченджи: _список_

На данный момент согласно нашим последним обсуждениям критических (требующих срочного исправления) дефектов и ченджей не зарегистрировано. Оставшиеся дефекты и ченджи, такие как: _список_ могут быть исправлены в следующих версиях.

Поэтому, прошу вас формально подтвердить приёмку данной версии, после чего мы сможем начать подготовительные работы по следующей версии».

Если Заказчик отвечает «Да, конечно, версия принята, продолжайте работы» - сохраняем его ответ вместе с нашим письмом и переходим к следующим шагам. Если Заказчик что-то оспаривает – разбираем ситуацию вплоть до получения подтверждения.

Без наличия формального подтверждения приёмки текущей версии переходить к работам по следующим версиям можно только в том случае, если вы доверяете Заказчику на 100%.

Следующим важным моментом перед стартом следующей версии является получение платежей за выполненные работы. Если вы работаете по схеме fixed cost, то, скорее всего платежи по вашему проекту привязаны к приёмке версий. В таком случае, получение платежа за сданную версию должно являться обязательным условием для старта следующей. Если вы работаете по схеме time&material – сдача версии является удобным поводом напомнить Заказчику про задержанные платежи, если таковые были.

В целом, сдача выполненного этапа работ – всегда хороший момент для взаиморасчётов. Разработчик, со своей стороны, вовремя сдаёт версию, Заказчик, со своей стороны, во время за неё расплачивается. Нормальная практика для любого бизнеса, в котором есть частичные поставки. Задержка поставок или платежей портит отношения. Так что, если со стороны Заказчика есть проблемы с платежами – не стесняйтесь ему об этом намекать, особенно при сдаче версий ☺.

Далее, код принятой Заказчиком версии рекомендуется сохранить в SVN отдельным образом. Trunk или brunch – решайте сами. Если вы не используете SVN – хотя бы просто сохраните отдельную копию кода проекта (желательно продублировав в нескольких местах). Сданная версия – это ваша «реперная точка». Возможны ситуации (например, разборки с Заказчиком в стиле «я это говорил, а то не говорил») когда вам понадобится «поднять» эту версию для сравнения с новой версией, либо для других целей.

После выполнения всего, описанного выше, желательно официально зафиксировать дату старта подготовительных работ по следующей версии. Это важно для планирования сроков сдачи следующих версий и проекта в целом. Уведомите Заказчика о том что «с сегодня, 16.03.2010, стартуем работы по версии №Х» и внесите эту дату в план проекта.

Глава №26: Подготовка к работе над следующей версией

В ролях - Менеджер Проекта, команда проекта, Заказчик

Артефакты – требования, changes, план проекта

Суть – Актуализируем требования и план проекта

Теперь кратко рассмотрим рекомендуемые (необходимые) действия при старте разработки каждой следующей (после первой) версии.

Шаг 1. Обновляем требования. Скорее всего, в процессе разработки и сдачи предыдущей(их) версии(й), появились некоторые ченджи. Часть из них уже реализована, другая часть – запланирована к реализации в последующих версиях. Ченджи – запросы на изменения – меняют требования проекта. А значит, чтобы ничего не забыть избежать недоразумений и противоречивых требований, необходимо обновить сводный документ с требованиями с учетом всех (сделанных и запланированных) ченджей.

Обновление требований – работа для аналитика проекта, или кого-то кто выполняет эту роль. Логично предположить, что обновление требований требует определённых трудозатрат, которые нужно учесть в планировании задач и сроков сдачи проекта. Эти трудозатраты

полезно предусмотреть ещё при начальном планировании проекта, до старта первой версии, на что и указано в соответствующей главе.

Впрочем, есть интересный способ работы с требованиями, который облегчает жизнь и аналитика, и менеджера проекта, да и всей остальной команды. В начале проекта используется сводный документ с требованиями – SRS, PFD etc. Затем, при старте первой версии все задачи разработки заводятся как таски в issue tracking system (viva Jira!). В описание таска копируем соответствующий раздел требований. Далее, в процессе работы, при поступлении ченджей (утверждённых) – информацию про ченджи добавляем к соответствующим таскам как комментарий. В общем, при условии доступа Заказчика к используемой системе, там же это всё сразу и обсуждается и утверждается, соответственно у таска меняются planned effort и т.п. Далее, при старте следующей версии – снова заводим таски, копируем требования новой версии из первичного документа, и сразу добавляем ченджи. Таски на новую версию заводим только перед её стартом – т.к. к этому времени часть первичных требований может быть уже не нужна, либо изменена до неузнаваемости.

Плюсами такого способа работы с требованиями является полная прозрачность изменений для всех участников процесса, и отсутствие необходимости обновлять громоздкий документ с требованиями после начала проекта. (Тем более что в конце проекта он уже не нужен – вместо него нужен хелп).

Обновлённый документ с требованиями (либо заведенные таски на новую версию) нужно обязательно утвердить с Заказчиком.

Шаг 2. Replanning. Вне зависимости от того, как вы обновляете требования, предположим, что это сделано. Логично, что при изменении требований (ченджи), оценка необходимых для их реализации трудозатрат изменяется. Так что теперь нужно обновить план проекта – проверить все оценки для новой версии, внести необходимые изменения в задачи и их продолжительность.

Кроме ченджей, нужно учесть исправление дефектов, которые были признаны незначительными при приёмке предыдущей версии. На это тоже нужно время, что полезно отразить в ПП.

Переоценку трудозатрат и проверку корректности последовательности задач нужно делать вместе с командой проекта. Даже если после сдачи предыдущей версии нет ни ченджей, ни дефектов. Не стоит забывать про возможные изменения планов внутри команды проекта – кто-то может перенести свой отпуск, кто-то запланировать пару отгулов перед/после праздников, праздники, кстати, тоже надо учитывать. Так что, собираем команду, все вместе проверяем задачи проекта и личные планы, вносим изменения в ПП и пересчитываем сроки (milestones).

Обновлённый план проекта, естественно, нужно утвердить с Заказчиком до старта разработки следующей версии.

Шаг 3. Обновляем техническую документацию. Как я уже упоминал, для крупных проектов следует вести техническую документацию.

Под «крупными» понимаем проекты, в которых:

- участвует более 3х разработчиков,
- тянутся более полугода,
- используются сложные технические решения,
- происходит частые замены в команде проекта.

Под «технической документацией» понимаем как минимум:

- концепцию архитектуры проекта,
- комментарии в коде (стандартизированные!),
- диаграмму классов,
- описание структуры БД.

Зачем это всё нужно я уже описывал, да и, в общем, это очевидно.

Техническую документацию (техдоки) надо обновлять в течение проекта, и удобно это делать после приёмки версии X но перед стартом версии X+1. Приёмка версии X Заказчиком означает, что в ней серьёзных изменений уже не будет. Соответственно, можно (нужно) проверить и обновить техдоки таким образом, чтобы написанное соответствовало сделанное. Далее, после обновления требований и переоценки трудозатрат, логично проверить, что изменения могут быть нормально реализованы с учётом выбранной архитектуры. То есть снова проверяем, и, при необходимости, обновляем техдоки.

Обновлённую техническую документацию нужно обсудить и утвердить при участии всей команды проекта.

Шаг 4. Начинаем работы по новой версии. Ну, то есть, если всё готово – то «побежали».

Глава №27: Про джедаев

В ролях - Менеджер Проекта, команда проекта

Артефакты – нет

Суть – разбор полётов

Вспоминаем про «светлую сторону силы» - мотивацию сотрудников и поддержание духа команды проекта на высоте, достаточной для успешного продолжения разработки проекта. Истинные джедаи делают так.

«Разбор полётов». Приёмка очередной версии Заказчиком – хороший момент для проведения «разбора полётов». Задача «разбора»:

- вспомнить ход разработки и тестирования версии,
- оценить успехи и неудачи в целом,

- оценить работу каждого сотрудника в рамках выпущенной версии
- отметить успехи и неудачи каждого
- сделать общие выводы на будущее
- раздать кнута и пряника
- придумать, как сделать так, что бы дальше было только лучше, и раздать соответствующие задачи кому надо

Как видно, задачи масштабные, а значит к проведению «разбора полётов» менеджер проекта подготовиться должен тщательно. Следует «вспомнить всё», и составить план собрания. Чтобы ничего не забыть, полезно пересмотреть записи, сделанные во время работы над версией, речь о которых пойдёт ниже:

- Отчёты по проекту, как внутренние (ежедневные) так и для Заказчика. Напомнят про встреченные проблемы и сроки выполнения задач,
- Отчёты сотрудников о трудозатратах. Напомнят о сверхурочных работах, отгулах, опозданиях, и т.п.,
- Meeting minutes. При правильном ведении – напомнят о встреченных проблемах, и, в том, числе успехах,
- Переписка с Заказчиком. Напомнит про все неожиданности, спровоцированные Заказчиком (ченджи).

«Вспомнив всё» хороший менеджер проекта подготовит для себя план проведения собрания, с пометками и необходимыми комментариями.

Следующей задачей для менеджера проекта будет получение средств и возможностей для «раздачи кнута и пряников» членам команды. Сделать следующее нужно:

- Готовим аргументированное «представление к наградам» для отличившихся сотрудников.
- Идём к начальству с этим отчётом. «Выбиваем средства» - утверждаем предполагаемые меры с начальством.
- Если версия в целом является успешной – просим у начальства средства на «тимбилдинг».

Суть в том, что ощутимую (материальную или др.) оценку результатов работы надо дать как сотрудникам по отдельности, так и команде в целом. Нормальной является ситуация, когда кто-то получает премию, кто-то – выговор и штраф, и вся команда при этом идёт в бильярд отметить успешное завершение очередного этапа работ. Что именно и как делать для «тимбилдинга» - решайте сами в соответствии с правилами вашей Компании и по согласованию с руководством.

Теперь можно назначить и провести собрание под кодовым названием «разбор полётов по версии ХХХ». Правила проведения собраний – предупреждать и объявлять тему заранее, выступать по очереди, говорить по сути, и фиксировать результаты – действуют для «разбора» так же как и для любых других собраний. Ещё раз повторю что надо охватить:

1. По шагам разобрать ход работ в данной версии.

2. Указать на задачи, где реальные трудозатраты не сошлись с планом
3. Вывести основные встреченные проблемы при выполнении задач. Оценить результативность принятых мер
4. Опросить сотрудников, собрать их комментарии
5. Сделать выводы на будущее, зафиксировать
6. Дать оценку успешности данной версии «в целом», руководствуясь соблюдением сроков, количеством дефектов, удовлетворённостью заказчика
7. Дать оценку работе каждого сотрудника, с привязкой к проблемам и успехам
8. При необходимости - дать рекомендации сотрудникам «на будущее», зафиксировать
9. Объявить «кнуты и пряники» (если конкретные меры и цифры называть публично не принято – объявите сам факт поощрений/наказаний, про цифры расскажете после собрания лично)
10. Для успешной версии – объявить о «тимбилдинге», обсудить и назначить время проведения.

После собрания пишем meeting minutes, в котором фиксируем выводы и принятые решения – как обычно. Общим результатом должно являться укрепление командного духа и повышение уровня ответственности сотрудников.

«Напутственная речь». При старте новой версии, после завершения подготовительных работ (шаги 1, 2 и 3 из прошлой главы), перед стартом разработки (шаг 4) истинный джедай (менеджер проекта) может провести «напутственное собрание». Суть в том, чтобы:

- напомнить сотрудникам о выводах, сделанных по результату сдачи прошлых версий, и принятых мерах
- указать на те задачи новой версии, выполнение которых требует повышенного внимания и ответственности (задачи критического пути, потенциальные риски)
- воодушевить команду на успешную работу (можно напомнить про тимбилдинг, премии и т.п.)
- объявить старт работ

«Про Ситхов и печеньки». Тёмная сторона силы, или вопрос личной мотивации самого менеджера проекта – очень опасна, особенно для молодых джедаев. Точно так же, как команде проекта важно (для мотивации) получить своевременную обратную связь – оценку результатов работы на очередном завершённом этапе – так и для менеджера проекта важно получить оценку своего труда (в том числе и по мотивации команды).

Печально, но факт, что у руководителей менеджеров проектов не всегда есть время вникать в течение каждого проекта и давать оценки по каждой версии продукта. Особенно если в Компании одновременно ведётся разработка множества разных проектов, разной степени важности для Компании, разной степени успешности, с возможными критическими проблемами в некоторых проектах. В общем, надеяться на «разбор полётов» по отношению к себе, тем более после каждой версии (итерации) проекта, может далеко не каждый РМ.

Однако, собранные, настойчивые и целеустремлённые джедаи всегда найдут возможности для корректировки своей самооценки и улучшения качества своей работы. Например:

1. Если ваш Заказчик достаточно нормальный и адекватный товарищ – обсудите свою работу с ним. Что бы он хотел улучшить и дополнить, что он вам посоветует? Такое общение может дать вам полезную информацию, и в целом полезно для улучшения отношений с Заказчиком.
2. Если у вас есть супервизор – можно самостоятельно инициировать оценку результатов версии. Супервизор обычно читает отчёты по проекту, следит за перепиской (как минимум за проблемными обсуждениями) и обладает большим опытом управления проектами в разных ситуациях. Даже обсудив ход работы в проекте «по верхам», можно узнать что-то полезное. Не бойтесь обращаться с вопросами – супервизор вам дан именно для того чтобы вы могли их задавать.
3. Если участники вашей проектной команды принадлежат к некоторым отделам (например, по технологиям – PHP, QA, C++, .NET) – очень полезным может оказаться общения с руководителями этих отделов. Вы можете узнать о том как ваши сотрудники отзываются о работе в вашем проекте, насколько они её удовлетворены и т.д. А также получить полезные советы касательно личных особенностей определённых товарищей. Вообще, общаться с руководителями отделов (если участники проекта «арендуются» в отделах по технологиям) нужно регулярно – т.к. им тоже полезно узнавать ваши отзывы.
4. Наконец, можно собрать отзывы о своей работе от своих же сотрудников. В личном порядке (на собрании этого делать нельзя!) попросить дать советы и рекомендации по улучшению своей работы как менеджера. Применимо только к адекватным и ответственным сотрудникам, требует наличия у менеджера самокритичности.

Надеюсь, эти советы помогут вам избежать психологических травм, которые нередко случаются на «тёмной стороне» ;-) И да пребудет с вами сила.

Глава №28: Продолжаем в том же духе

В ролях - Менеджер Проекта, команда проекта, Заказчик

Артефакты – все, в зависимости от ситуации

Суть – делаем то же что в гл. X – 27 пока не сдадим финальную версию

В этой главе хочется ещё раз перечислить важные моменты, которые не стоит игнорировать при переходе от версии к версии. Итак:

1. При поступлении новой информации касательно требований – уточнений, изменений и т.п. – необходимо объединять новую информацию со «старыми» требованиями. Либо обновляем основной документ с требованиями (SRS), либо добавляем информацию к таскам. Если этого не делать – повышается риск неверной интерпретации требований и «забывчивости» как у разработчиков, (сделали по неверной версии требований) так и у Заказчика (приёмка по не той версии требований). После обновления требования надо утверждать с Заказчиком – чтобы потом не было «а я такого не просил». Удобный момент для обновления требований – после сдачи очередной версии, перед стартом следующей.

2. При поступлении новой информации по требованиям также необходимо проверять и обновлять архитектуру проекта (концепцию). Если в проекте есть отдельный архитектурный документ – не забывайте вносить обновления. Если нету – как минимум, необходимо собрать разработчиков и проверить, как изменения влияют на уже написанный код, и планы на будущую разработку.
3. По завершению версии не забываем обновлять техническую документацию. Даже если никакой отдельной документации не ведётся, а есть только комментарии в коде – следует озадачить ведущего разработчика проверкой их наличия и корректности.
4. Release Notes является кумулятивным документом. При выпуске новой версии просто добавляем новую информацию в Release Notes о прошлой версии. В результате – имеем полный лог проекта, в котором понятно когда и в каком билде были добавлены новые функции, исправлены дефекты, внесены изменения и т.п.
5. При тестировании новой версии весьма полезно проверять работоспособность функционала прошлых версий. Не забываем регрессионное тестирование – и избегаем множества проблем и «ляпов».
6. После сдачи каждой версии необходимо проверять актуальность дат (milestones) в плане проекта. Даже если не было никаких изменений. В вопросе соблюдения запланированных дат сдачи версий лучше «перебдеть, чем недобдеть» - это помогает сохранить хорошие отношения с Заказчиком.
7. По возможности, не стоит начинать новую версию до получения всех платежей за предыдущие версии. Соблюдайте правило «сдача в срок – платежи в срок», приучайте к нему Заказчика.
8. Регулярно собирайте актуальную информацию по личным планам членов команды проекта. Помните, что отсутствие сотрудника на работе обычно приводит к задержкам в выполнении задач. Все плановые отпуска и отгулы влияют на план проекта – и чем раньше вы о них узнаете, тем больше будет возможностей для минимизации потерь для проекта. Проверку личных планов удобно совмещать с проверкой плана проекта перед стартом новой версии.
9. После приёмки версии Заказчиком полезно устроить с командой проекта «разбор полётов», на котором рассмотреть и оценить вклад каждого сотрудника и подвести итоги работы + сделать выводы на будущее.
10. Успешная сдача версии – хороший повод устроить небольшой праздник для команды проекта. Тимбилдинг рулит. Также, за сверхурочную работу, трудовые подвиги и досрочное выполнение задач сотрудников полезно премировать.

Глава №29: Сдача финальной версии

В ролях - Менеджер Проекта, команда проекта, Заказчик

Артефакты – все

Суть – Готовимся и сдаём финальную версию проекта

Наконец, все промежуточные итерации проекта завершены, и подходит к завершению финальная итерация. Сдача финальной версии несколько отличается от сдачи промежуточных, и эти отличия хочется подчеркнуть отдельно.

1. Регрессионное тестирование – для финальной версии является обязательным. Даже если вы не «забивали» на него при сдаче всех промежуточных версий. Причина – в финальной версии весьма желательно исправить _все_ известные дефекты приложения. Обнаружение уже известного ранее дефекта после сдачи финальной версии может сильно испортить отношения с Заказчиком. Так что, уделяем регрессионному тестированию особое внимание
2. Нагрузочное тестирование – обязательно должно быть успешно пройдено. Даже если все предыдущие версии прошли его успешно. Выпущенное приложение должно нормально работать при запланированной нагрузке. В противном случае – снова рискуем репутацией. А потеря репутации = уменьшение шансов на повторные проекты от этого Заказчика.
3. Тестирование безопасности – если есть – тоже обязательно для финальной версии.
4. В целом, предыдущие 3 правила сводятся к максимальному «вылизыванию» финальной версии перед сдачей. В идеале, дефектов быть не должно. В реальности – допускается сдача финальной версии с наличием незначительных дефектов, которые могут быть исправлены в течение срока бесплатной поддержки приложения (согласуется с Заказчиком отдельно).
5. Необходимо выполнить все ченджи, и не забыть их протестировать. Собственно, ченджи – это часть общего скоупа проекта (выпускаемой версии), и логично, что весь скоуп должен быть реализован и протестирован. В некоторых случаях часть ченджей оставляют на следующую версию приложения. Однако, разработка следующей версии – это отдельный проект, и такие ченджи уже не входят в скоуп версии, которую сдаём сейчас.
6. Необходимо проконтролировать готовность среды запуска приложения. Для случая, когда среда запуска подготавливается силами проектной команды – подготовить и проверить готовность. Для случая, когда среда запуска предоставляется Заказчиком – проверить доступность и соответствие требованиям приложение. Сделать это нужно до сдачи версии.

Всё это касается этапа подготовки финальной версии к сдаче Заказчику. По завершению подготовки финальная версия должна соответствовать следующим критериям:

1. Реализован весь функционал согласно скоупа проекта. Нет нереализованных требований или ченджей
2. Весь реализованный функционал протестирован на соответствие требованиям и отсутствие ошибок
3. Выполнены все необходимые (запланированные) виды тестирования
4. Подготовлена среда в которую будет установлена финальная версия (среда запуска)

Когда всё готово – начинаем процедуру сдачи версии. Как обычно, «заливаем» приложение в среду показа Заказчику, ждём окончания проверки и подтверждения приёмки. После получения от Заказчика подтверждения приёмки версии в среде показа – «заливаем» приложение в среду запуска и проверяем вместе с Заказчиком как она себя там чувствует. Если всё ОК и претензий нет – просим Заказчика подтвердить данный факт в письменном виде, причём именно в такой форме – «всё работает, претензий нет». Важно зафиксировать отсутствие претензий. Форма, в которой получаем подтверждение – зависит от условий Контракта (или заменяющих его договорённостей).

Если установка приложения в среду запуска выполняется на стороне Заказчика – делаем то же самое. То есть, помогаем при необходимости (консультируем) с установкой, затем проверяем работу приложения и если всё ОК – просим подтверждение приёма и отсутствия претензий к работе версии.

Далее, после установки приложения в среде запуска, следует зафиксировать вместе с Заказчиком дату начала срока бесплатной поддержки приложения. На всякий случай напомню, что это такое – общепринятой практикой является бесплатное устранение дефектов (и консультации) в течение некоторого времени после сдачи финальной версии. Срок такой поддержки варьируется в зависимости от сложности и длительности разработки и оговаривается в Контракте (или другим образом) до начала проекта. В общем, фиксируем дату старта срока поддержки и, вместе с тем, указываем дату его окончания. Важно чтобы Заказчик понимал, что бесплатная поддержка ограничена по срокам, и, соответственно, постарался максимально интенсивно его использовать («обкатать» на тестовых пользователях).

Следующим делом является получение всех платежей по проекту. Если Заказчик платил всегда вовремя – скорее всего всё уже ОК (нормальным является проведение последнего платежа после установки финальной версии в среду запуска). Если есть «висящие платежи» - настоятельно рекомендую получить их до передачи исходного кода проекта и технической (сопроводительной) документации, которая является следующим шагом.

Наконец, после того как приложение запущено, приёмка подтверждена Заказчиком, получены все платежи и зафиксирован период поддержки – отдаём Заказчику исходный код проекта и сопроводительную техническую документацию. Finita.

Глава №30: Поддержка приложения

В ролях - Менеджер Проекта, команда проекта, Заказчик

Артефакты – по необходимости

Суть – взаимодействие с Заказчиком после сдачи проекта

Итак, после сдачи финальной версии проекта начинается (в большинстве случаев) период бесплатной поддержки. В течение этого периода Компания-разработчик обязуется бесплатно для Заказчика устранять выявленные дефекты, а также, при необходимости, консультировать Заказчика касательно технических аспектов и нюансов работы приложения.

В классическом случае после сдачи финальной версии Заказчик устраивает период тестового использования продукта. В течение этого периода работа продукта «обкатывается» на группе тестовых пользователей, которыми могут быть как сам Заказчик и члены его команды (друзья, сотрудники, коллеги и т.п.) так и посторонние лица. Часто тестовый период «синхронизируют» по продолжительности с периодом бесплатной поддержки продукта разработчиком.

По результатам тестового периода Заказчик получает отзывы пользователей. В отзывах пользователей содержатся как жалобы на неадекватное (с их точки зрения) поведение приложения, так и предложения по улучшению работы продукта. Эти отзывы (особенно жалобные) Заказчики имеют обыкновение пересылать менеджеру проекта, требуя срочного устранения описанных «дефектов».

Поступать в данном случае следует точно так же, как и при получении отзывов Заказчика при сдаче версии в процессе разработки. Спорные вопросы – обсуждаем и решаем (обычно приходится указывать на то, что ряд «жалоб» пользователей, по сути, являются запросами на изменения, а не дефектами). Далее – дефекты выделяем отдельно, расставляем приоритеты (вместе с Заказчиком) и исправляем в рабочем порядке.

Ченджи – рассматриваем отдельно, и желательно по окончании периода бесплатной поддержки. Даже критичные (для Заказчика) ченджи не стоит делать сразу. Лучше сразу договориться, что в течение периода бесплатной поддержки ченджи не рассматриваем вообще, аргументируя это переключением разработчиков на другие задачи (а они нам нужны для оценок) и желанием сосредоточиться на устранении дефектов.

Причина такого подхода (ченджи – потом) заключается в условиях планирования работ и получения платежей. В течение разработки стоимость работ по реализации ченджей может быть включена в следующий платёж, а сама задачи для разработчиков включены в общий план работ путём его модификации. Поэтому реализация ченджей легко «вставляется» в текущую либо следующую версию.

Однако, после сдачи проекта и получения всех платежей реализация ченджей требует фактически организации новой итерации проекта, или даже нового проекта. С отдельным планированием работ и утверждением отдельного бюджета и сроков. Поэтому логично собирать ченджи «в кучу» и делать все сразу – чтобы не утверждать бюджет (и получать платёж) на каждый чендж в отдельности, и не дёргать разработчиков, которые могут быть уже заняты в других проектах.

В общем и целом, здесь есть некоторое противоречие между желанием собрать максимум ченджей вместе и обработать/реализовать общим списочком в рамках одной дополнительной итерации, и условиями работы разработчиков. Нормальной ситуацией является роспуск проектной команды после сдачи финальной версии, не дожидаясь завершения срока поддержки. Это логично и правильно с точки зрения бизнеса – устранение дефектов и реализация ченджей редко требует постоянной работы всей команды. Обычно в этой работе участвуют только некоторые разработчики, и то эпизодически. Время простоя команды в течение периода поддержки Заказчиком, естественно, оплачиваться не будет. Поэтому разработчики быстро переключаются на новые проекты. Вовлечение разработчика в новый проект обычно начинается с периода изучения документации и т.п., во время которого они могут периодически отвлекаться на устранение дефектов в сданном проекте. Но дальше, спустя некоторое время, разработчики будут задействованы в новых проектах уже полностью, и тут уже их отвлечение на реализацию ченджей становится проблемным.

Собственно, это и есть противоречие. Чем больше проходим времени с момента сдачи финальной версии – тем меньше доступность разработчиков, и тем удобнее собрать все

ченджи разом. Поэтому, выбор подхода к реализации ченджей в законченно проекте часто требует предварительного обсуждения с руководством Компании важности дальнейших работ с данным Заказчиком и планов на дальнейшее использования разработчиков в других проектах. Решение принимается с точки зрения бизнеса. Иногда выгодно «придержать» команду проекта некоторое время, не давая новых задач, чтобы быстро и эффективно реализовать ченджи (в случае важности Заказчика для Компании и ожидания развития этого проекта либо поступления новых заказов). Иногда команда перебрасывается на новые проекты в течение нескольких дней после сдачи финальной версии, и сроки реализации ченджей (а иногда и устранения всех багов) сильно растягиваются (в случае если для Компании ценность новых проектов и Заказчиков выше, чем предыдущего). Nothing personal, just business – не люблю эту фразу, но тут она к месту.

По завершению срока поддержки все последующие задачи от данного Заказчика выполняются только в виде новых итераций/проектов, исключительно на платной основе. В том числе устранение дефектов. Так что, менеджер проекта (законченного) при обращении к нему Заказчика с просьбами может смело (но вежливо) направлять его к менеджеру по продажам. Тем более, что чаще всего в этот момент сам МП уже занимается другими проектами.

Глава №31: Завершение проекта – взгляд изнутри

В ролях - Менеджер Проекта, команда проекта

Артефакты – по необходимости

Суть – финальный разбор полётов и завершение проекта для разработчиков

Для завершения нашего рассказа осталось рассмотреть завершение проекта со стороны проектной команды. Как я уже писал выше, после сдачи финальной версии проекта команду проекта можно потихоньку (сохраняя необходимые ресурсы для завершения стадии поддержки и возможных ченджей) распускать – привлекать к новым проектам и т.п. Однако перед этим было бы неплохо провести несколько мероприятий, описанных ниже.

Если коротко, сделать нужно следующее:

1. Провести финальный «разбор полётов», оценить вклад каждого сотрудника в проекте, наградить не причастных и наказать невинных
2. Зафиксировать все результаты проекта – сложить вместе все артефакты и дополнительную информацию, и сохранить на всякий случай для потомков в защищённом хранилище
3. По результатам проекта составить сводное резюме Заказчика – описать его психологический портрет, сильные и слабые стороны (с точки зрения разработки ПО), другие нюансы – всю информацию, которая может оказаться полезной в случае, если этот Заказчик обратится к Разработчику с новым проектом
4. Подготовить сводный отчёт по завершённому проекту, с указанием всех важных данных по планированию и выполнению бюджета – для общей оценки качества

работы Компании и бюджетной статистике. Такой отчёт обычно представляется руководству Компании, иногда вместе с «портретом» Заказчика.

5. Приблизительно определить необходимые дополнительные трудозатраты для периода поддержки и ченджей – необходимо для планирования дальнейшего использования сотрудников в других проектах.

Сделать всё это удобно следующим образом:

1. После сдачи финальной версии менеджер проекта с помощью ведущих разработчиков и тестировщиков составляет общий список значимых артефактов проекта. Обычно таковыми являются:
 - a. Изначальные требования проекта
 - b. Реальные требования проекта + дефекты + ченджи, по которым велась работа (при использовании Jira можно сделать экспорт всех задач, дефектов и ченджей)
 - c. План проекта – исходный и финальный (лучше сохранить обе версии, для сравнения)
 - d. Исходный код проекта (SVN dump, с учётом всех brunch'ей и trunk'ов)
 - e. Техническая документация – всё что есть
 - f. Переписка по проекту и логи общения с Заказчиком
 - g. Спецификация сред проекта
 - h. Сценарии тестирования и тест-планы
2. Менеджер проекта собирает информацию и утверждает возможные меры (поощрения/наказания) для «разбора полётов»
3. Назначаем собрание с участием всей команды проекта, темы собрания:
 - a. Сверка списка артефактов, подлежащих хранению
 - b. Разбор полётов и раздача слонов
 - c. Составление резюме Заказчика
4. На собрании, после выполнения п.а. и б. проводим опрос сотрудников – просим дать свою характеристику Заказчику. Важно услышать мнение всех сотрудников, в том числе тех, кто с Заказчиком не общался. Комментарии по качеству поступающих требований, срокам получения ответа на заданные вопросы, формулировке дефектов – всё это будет полезным. Полученную информацию фиксируем.
5. Назначаем ответственного за сбор и сохранение технических артефактов.
6. В конце собрания объявляем о завершении проекта. Можно сразу сообщить, кто из сотрудников может быть задействован для исправления дефектов и реализации ченджей, а кто уже свободен (если с этим вопросом всё уже понятно).
7. После собрания менеджер проекта составляет резюме (портрет) Заказчика, основываясь на своём опыте и отзывах участника проекта.
8. Сохраняет все артефакты проекта (согласно списка) установленным (принятым в Компании) образом
9. Готовит и отправляет общий отчёт для руководства
10. Ощущает удовлетворение от проделанной работы и хорошо выполненного проекта ☺

Немного подробнее остановимся на п.б. В большинстве случаев необходимые дополнительные трудозатраты менеджер проекта может оценить самостоятельно – руководствуясь знанием данного Заказчика и особенностей проекта. Исходя из оценки

трудозатрат и знания проектной команды также можно «прикинуть» кто из сотрудников лучше справится с этими работами (исправление дефектов в течение периода поддержки, реализация ченджей потом). После чего, составленный приблизительный план рекомендуется обсудить и утвердить с руководством Компании и начальниками отделов, в которых числятся указанные товарищи. Чтобы потом не было ситуаций, когда нужный сотрудник уже на 100% задействован в других задачах, а тут срочные багфиксы пришли. Избежать таких ситуаций полностью удаётся редко, но приложить все усилия для их минимизации однозначно стоит. И делать это лучше ещё до финального собрания с командой проекта.

Ну и, напоследок, снова о «тёмной» стороне. Для менеджера проекта по завершению проекта очень важно и полезно получить отзыв о своей работе, причём от всех участвующих сторон, которыми являются:

1. Команда проекта
2. Заказчик
3. Руководство Компании

Как это сделать? Например, где-то так:

1. На финальном собрании, после составления резюме по Заказчику, можно попросить участников проекта дать характеристику своей работе. Такой вариант подходит в случае весьма «тёплых» и открытых отношений в команде. Если есть риск, что обсуждение сведется к ругани, либо народ просто не выскажет то, что думает на самом деле – можно поступить другим образом. После собрания можно разослать сотрудникам краткий «опросник», в котором попросить оценить качество работы МП по различным критериям. Можно опрос сделать анонимным. Кстати, в хороших Компаниях такой опросник и оценка работы менеджера по результатам проекта может являться стандартной практикой ;-)
2. Заказчика можно также попросить дать развёрнутый отзыв по работе с Компанией в целом, и с данным менеджером проекта в частности. Возможно, в Компании подобный приём является стандартной практикой. Если нет – я рекомендую менеджеру проекта сделать это самостоятельно. Получите полезную информацию, как для себя, так и для отчёта руководству.
3. В идеале, руководство само вызовет вас «на ковер» и сделает с вами то же, что вы сделали с сотрудниками на финальном разборе полётов 😊 Руководствуясь при этом сводным отчётом по проекту, отзывами сотрудников и Заказчика. Это в идеале, конечно. Если что-то подобное в вашей Компании не принято – не стесняйтесь обратиться за оценкой своей работы самостоятельно.

В целом, должность «менеджер проектов» обычно подразумевает высокий уровень личной ответственности и самостоятельности. Именно поэтому, кстати, менеджерам редко дают обратную связь и наставления «сверху» - руководство часто самостоятельность смешивает с самодостаточностью. Однако, если вы заинтересованы в своём профессиональном росте и адекватной самооценке – не стесняйтесь просить отзывы и фидбеки. Даже у руководства. Это нужно вам, в первую очередь.

The End.