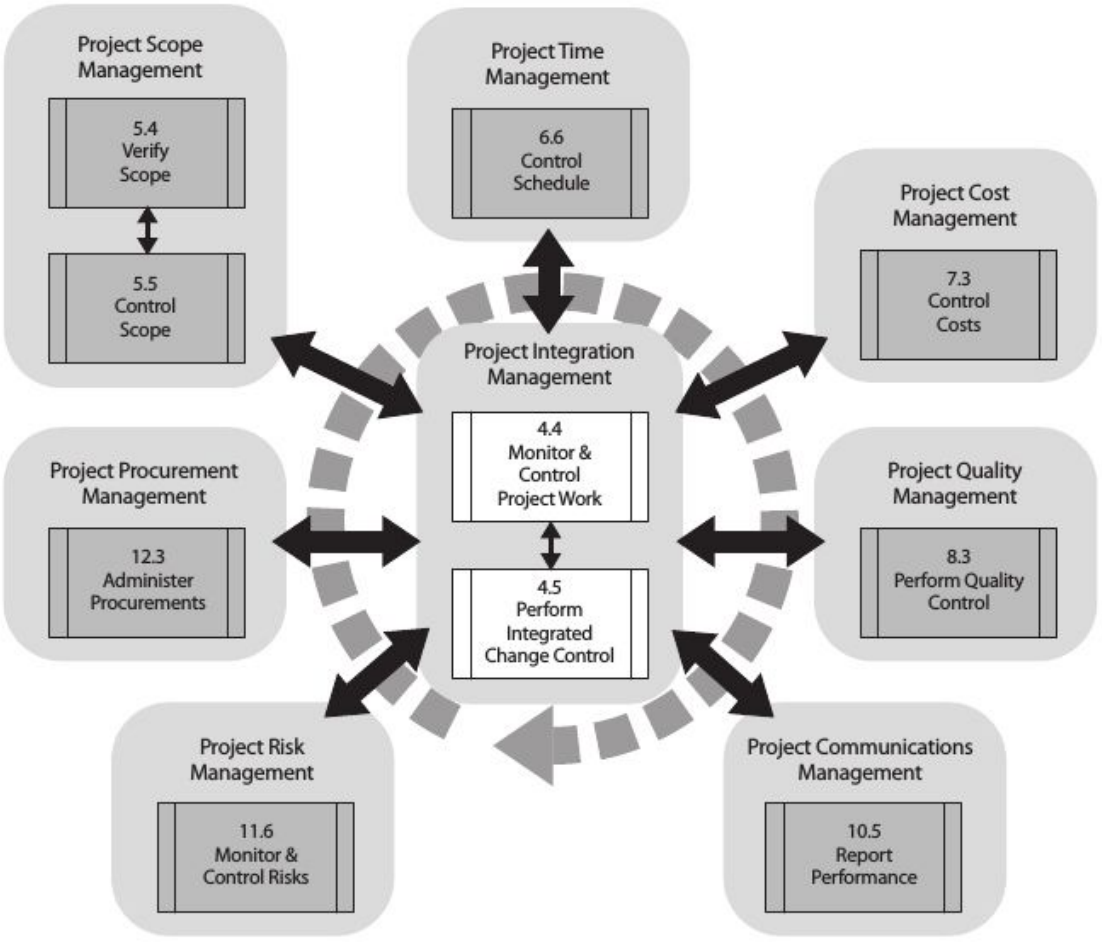
Итак, мы продолжаем говорить про управление проектами и действия, которые менеджер проекта может (и должен) совершать в ходе проекта. PMBOK определяет целый набор разных видов деятельности (см. рисунок ниже), мы же продолжим говорить про треугольник равновесия проекта и способы балансирования этого равновесия.



Балансирование равновесия проекта

Равновесие в проекте нужно сохранять постоянно (до завершения работы над проектом). Эта работа делается на разных уровнях: проекта, бизнес-целей, компании.

Балансирование на уровне проекта. Изменения, которые никак не повлияют на сроки, бюджет, количество людей и т.д., менеджер может принимать сам, они не требуют авторизаций или разрешений. Пока это остается в рамках ограничений, с проектом можно делать всё, что угодно.

Балансирование на уровне бизнес-целей. Изменения на уровне проекта возможны не всегда, иногда их просто недостаточно, и требуется сдвигать одну или несколько вершин треугольника равновесия. В таком случае нужно принимать решения на уровне бизнес-целей, а это требует согласования с заказчиком и/или руководством компании.

Балансирование на уровне компании. Изменения на уровне компании --- глобальные изменения. Подобные изменения принимаются уже руководством

компании, которое должно балансировать сразу все проекты компании. Например, у одного проекта может быть резко урезан бюджет с целью выделения дополнительных средств другому, более критичному проекту. Данный уровень в лекции мы затрагивать не будем.

Балансирование на уровне проекта

Повторная переоценка задач

Первое, что стоит попробовать сделать при балансировке на уровне проекта -- повторно оценить задачи. То есть, уже в процессе работы ещё раз пересмотреть график проекта и задачи в нем. Возможно сейчас у вас больше информации, чем на этапе изначальной оценки, и оценки получатся точнее. Вполне возможно, какие-то задачи были переоценены, а некоторые вообще уже потеряли актуальность.

Однако, при переоценке задач следует действовать очень осторожно: нельзя выдавать желаемое за действительное. Например, если хочется сократить бюджет, то возникает большой соблазн обмануться и избавиться от задач, которые на самом деле важны.

Плюсы: потенциальное сокращение сроков, бюджета, объема работ. Но если новой информации или требований у вас не появилось, то этот метод вряд сильно сработает. И тем не менее, к нему стоит время от времени возвращаться.

Перераспределение задач критического пути

При балансировании проекта всегда стоит уделять особое внимание задачам на критическом пути в сетевом графе.

Можно попробовать увеличить число людей, занятых работой над критическими задачами, задавшись целью сделать их быстрее. Обычно привлекают людей, работающих над другими задачами. Однако, такой подход может скрывать массу подводных камней. Во-первых, добавление людей к задаче не всегда повышает продуктивность работы, например, если задачи были распланированы с оптимальным количеством людей. Кроме того, следует иметь в виду, насколько хорошо задачи параллелятся. Если хорошо, то добавление новых людей к решению этой задачи, скорее всего, принесет пользу. Но чаще всего задачи нельзя распараллелить. Например: задача разработки архитектуры, запланированная на одного архитектора, не будет сделана существенно быстрее архитектором и четырьмя программистами. Добавление людей к таким задачам в худшем случае только замедлит работу, а в лучшем --- принесёт незначительное улучшение.

Следует также иметь в виду, что при привлечении новых людей к решению задачи продуктивность каждого из работников может упасть. Соответственно, расходы проекта растут. Ну и при переводе человека из одного проекта в другой может потребоваться время на его обучение и адаптацию.

Таким образом, при перераспределении людей по задачам нужно учитывать следующие моменты.

- Квалификация работников: не стоит заставлять дизайнера писать тесты или проектировать архитектуру.

- Действительно ли это поможет сократить время работы? Не будет ли человек лишним?
- Как поменялись резервы в задачах? Не изменился ли критический путь?

Добавление людей в проект

При добавлении людей в проект задачи практически никогда не начинают резко делаться быстрее, чем до этого, т.к. при добавлении человека в уже запущенный проект его нужно хотя бы как-то обучить и адаптировать, контролировать его работу. Это приводит к трате времени других людей в проекте, соответственно, сбивается график проекта: человек, который мог бы в это время делать задачи проекта, тратит своё время на нового сотрудника (а сам новый сотрудник ещё не работает). В графике продуктивности команды при добавлении новых людей почти всегда происходит сначала небольшой спад.

Исходя из этого, в краткосрочной перспективе добавление новых людей в проект не выгодно (только если добавляется не какой-нибудь супер-специалист, который хорошо знает команду, предметную область и проект). Почти всегда добавление новых людей -- это инвестиция в будущее проекта.

Резервы для добавления обычно ищутся внутри компании, но обычно из резерва можно найти не самых опытных людей. Поэтому не стоит надеяться на то, что в критической ситуации можно резко добавить ещё крутых специалистов, и всё пойдёт быстрее.

Привлечение новых людей хорошо работает в opensource проектах, поскольку в подобных проектах, как правило, обычно строго древовидная иерархия и с новым человеком работает локальная часть этой иерархии. Однако, например, в agile проектах это может работать хуже, поскольку новый человек будет обращаться за помощью ко всем (т.к. может) и тем самым затормозит прогресс проекта.

Привлечение экспертов в проект

Эксперты бывают внутренние (изнутри компании) и внешние.

Внутренний эксперт -- специалист внутри компании, временно снятый со своего основного проекта. Такой эксперт привлекается, как правило, чтобы помочь с какой-то локальной проблемой, а потом вернуться к своей основной работе. Плюсы: человек лоялен к проекту/компании, обладает требуемыми компетенциями и готов их вам предоставить в лучшем виде. Например, эксперт настраивает базу данных, делает это быстро и грамотно, время внутри команды не тратится. Но, во-первых, эксперт -- человек, он тоже может совершить ошибку при настройке, может не уложиться в сроки и т.д. Во-вторых, никаких компетенций в области конкретной задачи, для решения которой привлекли эксперта, командой проекта не приобретается, т.к. всё сделал сторонний человек.

Что же делать, если снова случится такая проблема? Если бы было потрачено время на обучение своего сотрудника, то при необходимости этот сотрудник мог бы снова решить проблему. Таким образом, в долгосрочной перспективе лучше обучать своих специалистов.

С целью избежания подобных ситуаций (а также с целью повышения компетенций команды) при привлечении эксперта со стороны, его нужно заставить

передавать знания. Вряд ли это будет встречено с охотой, но неким образом эксперт всё же должен быть вовлечён в процесс планирования. Можно заставить эксперта участвовать в статус-митингах, попросить провести семинар и т.д. В таком случае привлечение эксперта работает хорошо, и при возникновении проблемы, схожей с той, для решения которой был привлечён эксперт, команда проекта уже будет иметь хотя бы некоторые знания, связанные с решением возникшей задачи.

Внешний эксперт -- то же самое, но с компетенциями всё обстоит ещё хуже, поскольку это полностью сторонний человек, которого можно попросту не найти, если вдруг проблема возникла снова (или же при решении были допущены ошибки).

Предпочтительный вариант -- растить собственных экспертов. Это осуществляется либо с привлечением внутренних экспертов компании, либо же приглашением людей со стороны, которые будут обучать членов команды. Следует специализировать людей внутри команды в конкретных областях. Со временем они вполне сами могут стать экспертами (если не уйдут от вас в другой проект ☺).

Аутсорсинг частей проекта

Аутсорсинг -- передача части проекта на разработку сторонней компании. При подобном подходе возникают 2 проблемы: помимо неприобретённой экспертизы, вы получаете зависимость от сторонней компании. Можно даже не догадываться, что происходит в той компании, которой отдана часть проекта, и у менеджера проекта существенно меньше возможностей контролировать процесс разработки. Аутсорсинг -- довольно экстремальная вещь: или всё идёт хорошо, или всё идет плохо, а команда об этом и не подозревает. Нужно стараться контролировать процесс выполнения работы над частью проекта, отданной “на растерзание” другой компании. Из очевидных плюсов --- аутсорсинг позволяет избежать проблем с квалификациями своих разработчиков. Например, когда нужно сделать приложения под все платформы, а программистов под MacOS в команде нет, и заводить их не хочется. В таком случае аутсорсинг -- очень выгодное решение.

Сверхурочная работа

Ещё один часто приходящий на ум вариант повышения продуктивности -- заставить членов команды работать больше. Казалось бы, всё выглядит очень оптимистично: пусть есть X часов в неделю суммарно при работе программистов по 40 часов. Если заставить их работать по 60 часов в неделю, получим 1.5 * X. Кроме того, не нужно добавлять новых людей, обучать их и т.д., просто члены команды будут работать дольше. Да и дополнительное время они будут работать в нерабочее время, когда офис не так сильно заполнен людьми, так что будут меньше отвлекаться и т.п.

Однако, на самом деле всё вовсе не так хорошо. Конечно, это нормально, если перед дедлайном приходится, например, поработать в выходные, чтобы сделать релиз в понедельник, но если переработки происходят регулярно, то программисты начинают “выгорать”. Люди начинают уставать, продуктивность падает.

При решении интеллектуальных задач производительность очень сильно страдает от усталости. Заставить человека, занимающегося интеллектуальным трудом, работать больше, чем обычно --- почти всегда путь к плохим последствиям. Как писал ДеМарко, на каждый час переработки приходится примерно час

недоработки. Не бывает так, чтобы люди волшебным образом стали работать в 1.5-2 раза больше. Фанатизм в работе сказывается на здоровье, усталости. Заканчивается это часто возросшей апатией человека и нежеланием делать вообще что-либо. Таким образом очень легко теряются программисты. В долгосрочной перспективе переработки крайне невыгодны.

Грамотный менеджер должен не заставлять команду программистов работать “на часик подольше”, а как раз наоборот, следить за тем, чтобы не было систематических переработок. Сознательные переработки --- признак непрофессионализма. Если человек каждый день засиживается на работе, то, скорее всего, у него проблемы с тайм-менеджментом. Скорее всего, в течение дня он тратит свое время на что-то никак не связанное с работой. Здесь, опять же, бывают исключения, но таких людей (“фанатиков”) очень мало, и с ними, как правило, всё понятно.

Снижение качества проекта

Качество проекта никогда не следует понижать. Конечно, соблазн велик, но последствия всегда весьма плачевны. Например, можно сэкономить на тестах, но взамен поймать кучу рандомных багов. Даже в краткосрочной перспективе не стоит понижать качество продукта, т.к., скорее всего, будет сделан очень некачественный продукт, который потом придется переделывать, на что уйдет больше времени, особенно если это придётся делать через несколько месяцев после завершения работы. Возможно даже, что придется переписывать весь проект заново. Гораздо дешевле не допускать ошибок, чем их исправлять.

Жертвовать можно чем угодно, но не качеством. Даже если вы сдадите проект, потерянную репутацию потом будет крайне сложно восстановить.

Балансирование проекта на уровне бизнес-целей

Пересмотр границ проекта

Когда приходит понимание, что в заданные сроки и бюджет требуемую функциональность реализовать не получается, стоит посмотреть, а нельзя ли как-то подвинуть границы проекта -- нельзя ли от какой-то функциональности более-менее безболезненно избавиться (хотя бы на данном этапе). Подобные решения требуют обоснованных переговоров с заказчиком и часто основываются на анализе требований и сценариев использования продукта. Поэтому их можно осуществлять только на уровне бизнес-целей компании. Ну и не всегда можно безболезненно выбросить какую-то функциональность: всё же продукты должны обладать какими-то особенностями, чтобы иметь на рынке конкурентное преимущество. А такие особенности чаще всего и являются тем, что сложно в реализации (иначе у конкурентов оно уже было бы тоже реализовано).

Подстраивание проекта под дедлайны

Часто бывает так, что у продуктов есть фиксированный дедлайн, в который команда не укладывается, однако, дедлайн имеет первостепенную важность. В таком случае следует разбить период времени до дедлайна на несколько этапов, по завершении каждого из которых будет согласовываться дальнейшее направление

развития проекта. В каждой ключевой точке фиксируется функциональность на ближайший этап, при достижении следующей планируется следующий этап и т.д. В итоге набор функциональности можно определять динамически.

Подобный подход работает далеко не всегда. Например, заказчик может захотеть добавить что-то, что просто не вписывается в текущую архитектуру. Также заказчик может хотеть работать по более традиционной схеме и отказаться брать на себя ответственность за разбиение проекта этапы и определение функционости на них.

Работа на опережение

Для работ, которые нельзя сделать параллельно, можно применить следующий подход. Допустим, у нас есть задача A, а также задача B, которая должна быть сделана по завершении работы над задачей A. В таком случае можно начать делать B ещё до завершения A. Например, можно начать реализовывать отдельные компоненты проекта до того, как архитектура будет закончена целиком.

При таком подходе надо действовать аккуратно, т.к. есть большая вероятность того, что придется переделывать решение. Это связано с тем, что принимаются предположения касательно реализации в духе “скорее всего, это будет так”, однако, гарантии этого нет и в результате всё может быть совсем иначе.

Однако, если всё сложится удачно, данный подход может позволить получить хорошую выгоду, однако, возможен и негативный исход, который приведет к уходу в минус. Тем не менее, при грамотном анализе рисков подобная “хитрость” может помочь. С помощью такой техники можно сэкономить до 30-40% времени.

Incremental delivery

Поэтапная разработка: поставлять заказчику изменения в продукте небольшими партиями. Такой подход работает далеко не со всеми проектами (и уж тем более не со всеми заказчиками). Например, это точно не сработает с большими, монолитными системами. Но, как правило, такое бывает довольно редко, а вот заказчики, которые не хотят так работать, встречаются весьма часто. Мы это уже подробно обсуждали в лекциях про гибкие модели разработки.

Прототипирование

Прототипирование -- мощный инструмент, суть которого заключается в том, что до начала работы над проектом создается черновой вариант проекта целиком (или каких-то его частей), но без особого внимания к качеству и деталям. Такой прототип создается просто для теста и получения опыта/знаний о предметной области, поэтому, когда работа над ним завершена, нужно избавиться от него и сделать заново, но уже хорошо.

Этот подход вынуждает потратить больше денег на разработку (приходится делать многое дважды), но, с другой стороны, позволяет снять риски и потратить меньше денег на исправление ошибок. Следует отметить, что очень важно избавиться от первоначального прототипа по завершению его разработки, а не использовать его в качестве основания для “настоящего” проекта, поскольку прототип, как уже было

сказано выше, делается на скорую руку и, скорее всего, будет целиком и полностью основан на “костылях”.

Снижение прибыльности проекта

Пожертвовать прибыльностью проекта -- последний шаг, к которому стоит прибегнуть, если ничего больше не работает. Это подразумевает уход проекта “в минус”, заём денег у других проектов, и т.д. Другой вариант -- жертвовать репутацией и выпускать продукт ненадлежащего качества, позже дедлайна, и т.д. Далеко не все компании согласны на такое, поэтому чаще прибегают к потери прибыли проекта, но сохранению репутации.

Отслеживание прогресса проекта

Для того, чтобы уметь адекватно управлять ходом проекта, менеджер должен постоянно отслеживать кучу самых разных характеристик проекта.

Задачи

Например, следует почаще заглядывать в таск-трекер, и делать соответствующие выводы: сколько задач открыто, сколько закрыто, переоткрыто за какой-либо период; какое среднее количество дней до завершения задачи; среднее время переработок/недоработок по задачам; и т.д.

Это очень важная информация, которую стоит накапливать и отслеживать тренд изменений. Лучше всего нарисовать графики для каждой из наиболее важных метрик, это позволит понять события, происходящие в проекте, более детально. Например, если количество переоткрытий задач растет, нужно уделить больше внимания качеству и ревью, и т.д.

Следует стараться делать задачи небольшими, иначе отслеживать их прогресс будет очень сложно. У каждой задачи должен быть очевидный критерий завершенности, чтобы однозначно определять, когда задачу можно закрыть.

Помимо таск-трекера источником подобной информации являются регулярные статус-митинги. Они также помогают отслеживать прогресс, но более направленно и субъективно. Например, человек не будет писать о сложностях, которые у него возникли при работе, в комментариях к своей задаче, но при разговоре он об этом, скорее всего, скажет.

Люди

Из таск-трекера также можно извлечь много информации персонально по поводу каждого конкретного разработчика. Сколько задач конкретный человек сделал, сколько переоткрыл, сколько сейчас на нём висит, переработка/недоработка, и т.д. Например, если у программиста недоработки и большое количество переоткрытых задач, то, скорее всего, он работает неэффективно и нужно привлечь его внимание к этому вопросу.

Также хорошей практикой являются еженедельные отчеты. Обычно они оформляются в виде таблички, в которой указаны задачи, которыми человек занимался, и количество часов на каждую. С таким видом статистики следует быть осторожным, т.к. на практике менеджер часто хочет увидеть в этой табличке строго 40

рабочих часов за неделю. Не стоит так делать, поскольку если заставить людей писать “40”, то они и будут писать “40”, а такая статистика вряд ли будет соответствовать действительности, т.к. люди будут подгонять время под это количество часов. Как показывает практика, в реальной жизни получается не 8 часов в день, а немного меньше. Поэтому, если проект идет по графику и результат получается хорошим, то 40 часов в неделю может и не быть (и это не страшно). Фраза “должно быть 40” всегда всё портит. Если же этого избегать, то программисты не будут пытаться натянуть свой график на эти мифические 40 часов. Соответственно, оценки будут честные.

Следует также сказать, что написание отчетов дисциплинирует программистов (несмотря на то, что они будут этому активно сопротивляться), а для менеджера такие отчеты очень важны для сбора и анализа данных. Если есть возможность генерации подобных отчетов из таск-трекера -- это большой плюс.

Дефекты

Аналогично людям и задачам, только по известным ошибкам и прочим дефектам: общее количество дефектов, относящихся к сотруднику; количество незакрытых дефектов, относящихся к сотруднику; среднее время исправления дефекта; процент переоткрытых дефектов; распределение дефектов по модулям/компонентам, приоритетам, статусам; количество дефектов для каждой задачи и т.п.

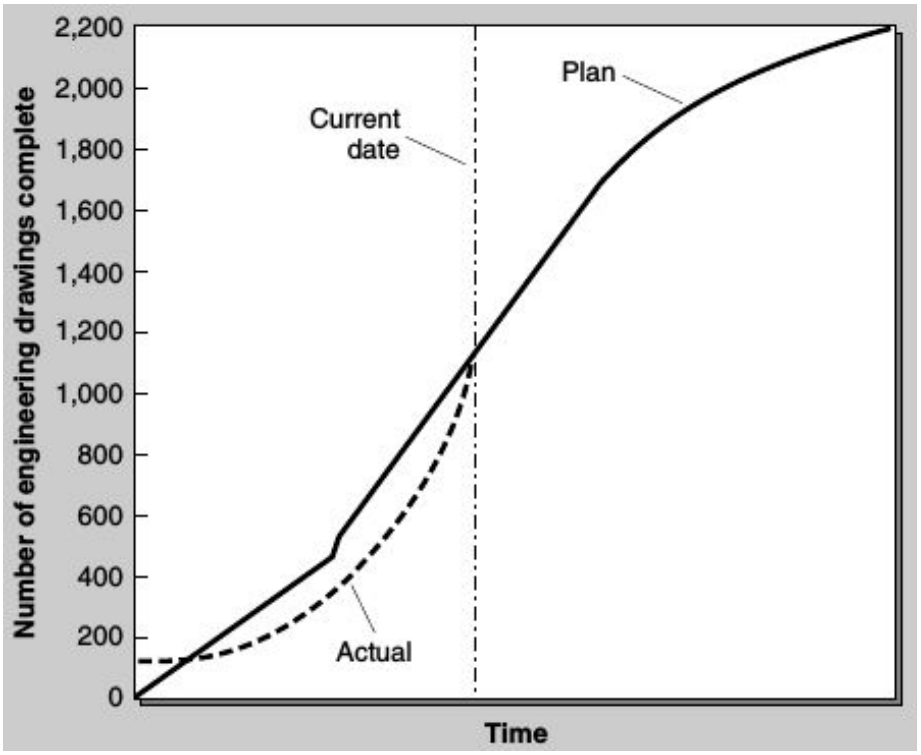
Коммиты

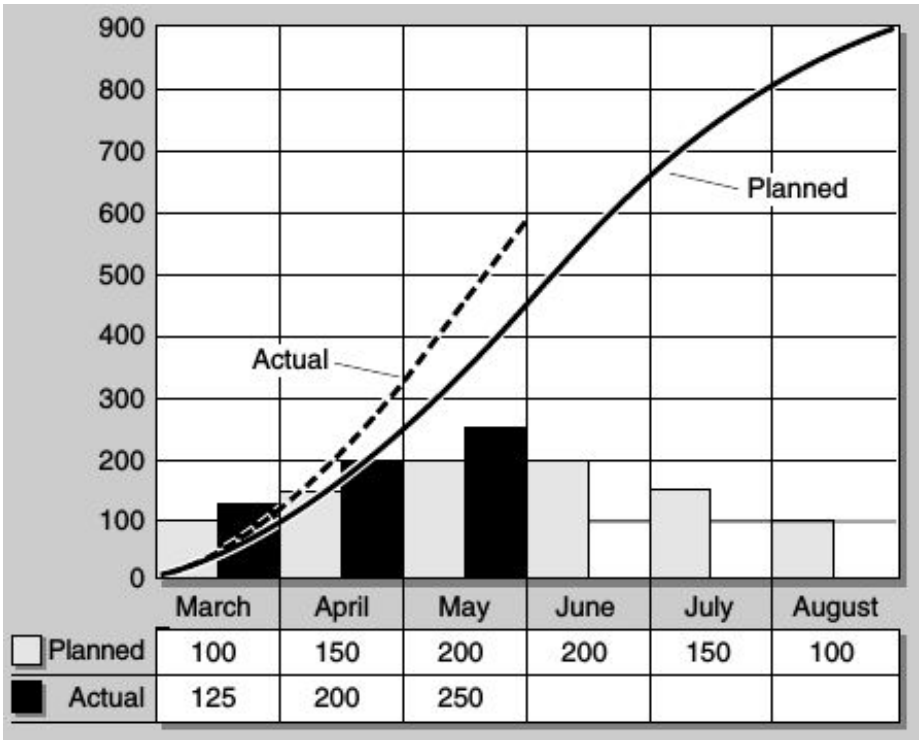
При оценке производительности программиста по коммитам важно учитывать многие факторы. Во-первых, важно не количество коммитов, а их содержательная часть. Просто считать коммиты или количество измененных строчек --- не лучший подход, поскольку изменения бывают разные (например, можно просто заменить все пробелы в проекте на символы табуляции. Количество строчек будет огромным, а их практическая ценность --- нулевой). У всех разный стиль разработки, но всё же обычно есть общепринятая культура в команде, так что интересно смотреть на время коммитов, их регулярность.

Графики

При анализе каких-то численных показателей следует смотреть на общую динамику изменений. Не стоит смотреть на цифры в отдельности, т.к. это не слишком наглядно.

Например, рассмотрим 2 графика: по планируемой/выполняемой работе и по затратам:





Оба графика показывают изменение показателей во времени, но у них есть свои недостатки. Первый график плох тем, что в нем не учитывается сложность задач. Он показывает лишь то, что команда отстает, а в какой-то момент догоняет исходный план. Но это не показывает, какие задачи были сделаны -- возможно, самые простые, а может быть и самые сложные.

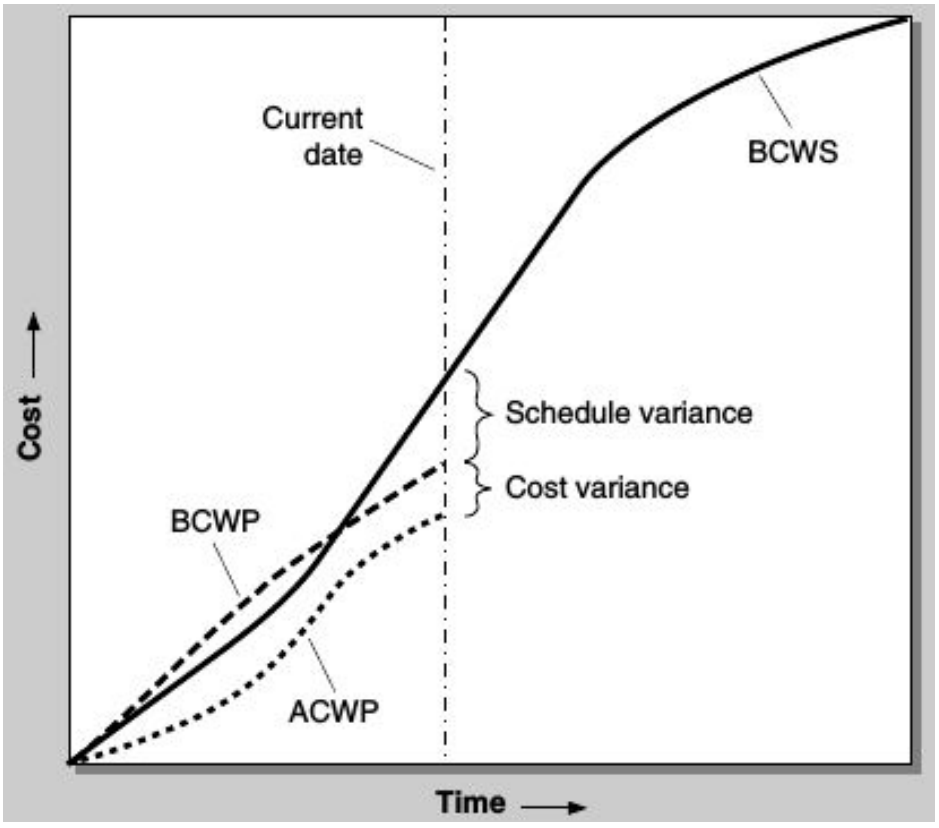
Второй график -- график расходов -- показывает, что команда тратит больше, чем нужно, но, возможно, было выполнено гораздо больше запланированной работы.

Для полноты картины нужно смотреть на происходящее в проекте со всех сторон, иначе особого смысла нет. Без контекста подобную информация можно с лёгкостью интерпретировать неверно.

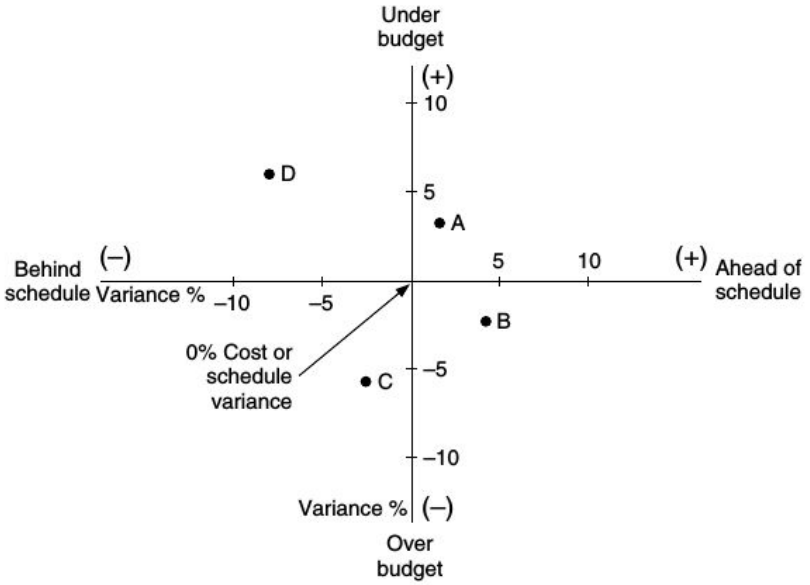
Ещё некоторые показатели:

- Budgeted cost of work performed (BCWP) -- запланированный бюджет
- Actual cost of work performed (ACWP) -- реально израсходованный бюджет
- Cost variance (CV) = BCWP − ACWP -- разница между планом и реальными расходами
- Cost variance percent (CV%) = CV / BCWP -- то же самое, но в процентах
- Cost performance index (CPI) = BCWP / ACWP -- индекс производительности (если больше единицы --- тратится больше, чем запланировано, меньше --- тратится меньше)
- Budget at completion (BAC) -- первоначальная оценка стоимости всего проекта
- Estimate budget at completion (EAC) = BAC / CPI -- планируемая оценка стоимости всего проекта

Какие-то из этих или другие интересующие показатели тоже имеет смысл отслеживать на графиках.

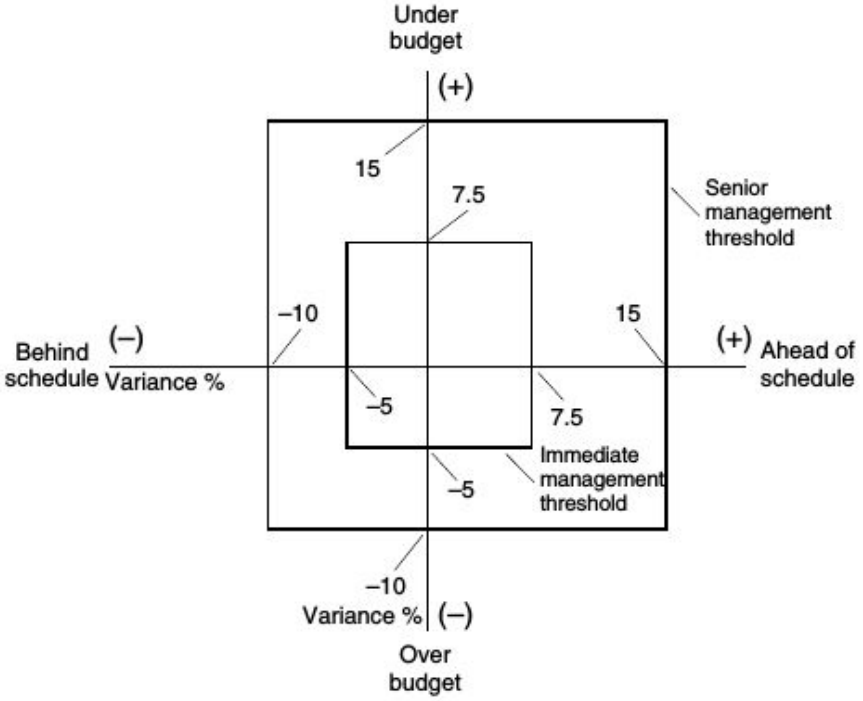


Более интересный график пытается учитывать одновременно и время, и деньги:



На горизонтальной оси тут отклонение по времени, на вертикальной -- по затратам. Например, первый квадрант -- проект идёт с опережением времени и бюджета, третий квадрант -- отставание по времени и перерасход бюджета.

На этом же графике полезно отмечать критические пороги значений параметров, при которых имеет смысл обозначить проблему перед начальством.



Границ может быть несколько в зависимости от критичности значений, и они позволяют более осознанно принимать проектные решения самостоятельно или определять момент, когда уже пора таки обратиться за помощью.

Коммуникации

Чтобы реагировать на изменения, нужно уметь общаться с людьми. Можно выделить 4 вида коммуникаций внутри проекта: с начальством и заказчиками, внутри команды, коммуникации, связанные с управлением изменениями и коммуникации, связанные с завершением проекта. Коммуникации с заказчиками и начальством мы уже рассматривали на одной из прошлых лекций, далее рассмотрим остальные виды.

Команда

Есть четыре основных мотивации для коммуникации внутри команды.

Определение области ответственности. Все члены команды хотят понимать, кто что делает. Очень плохо, когда на двух программистов “повешена” одна и та же задача. Для избежания подобных ситуаций нужно не только использовать таск-трекер и раздавать задачи, но и давать возможность людям общаться друг с другом. Понимать границы ответственности каждого конкретного человека очень важно (хотя все зависит от уровня задач).

Координация работ. Например, при обсуждении с тимлидом нового подпроекта, нужно проработать детали и каналы взаимодействия как внутри команды, так и с людьми за её пределами. Если в проекте есть несколько команд, то нужно время от времени собирать их тимлидов вместе и обсуждать с ними задачи более высокого уровня, не для конкретного человека, а для всей подкоманды. Без личного взаимодействия часто сложно понять, кто в каком направлении движется, и организовать работу так, чтобы команды не наступали друг другу на пятки.

Отслеживание статуса. В процессе выполнения задач можно получить некоторые знания, которые хотелось бы передать, хорошо наладить обмен такими знаниями.

Информация из внешнего мира. Менеджер --- “окно во внешний мир” для команды. Он взаимодействует с заказчиком, начальством и доносит до команды информацию и решения, которые на данном уровне принимаются. Следует время от времени стоит устраивать общие собрания и доносить до программистов некие новости, которые на них влияют. Такая информация поднимает моральных дух разработчиков. Также часто для многих разработчиков бывает полезно напрямую пообщаться с пользователями, понять их потребности и желания.

Совещания

Формат митингов очень сильно зависит от культуры проекта, например, в Scrum’е довольно эффективны короткие stand-up совещания. В любом случае имеет смысл зафиксировать культуру проведения совещаний в проекте, она дисциплинирует и вносит предсказуемость (например, если человека отвлекают от работы и говорят “иди на митинг”, то это, скорее всего, очень сильно его разозлит, но если он будет знать, что в определенное время ему предстоит идти на митинг, то заранее спланирует свой рабочий процесс с учётом этого).

Kickoff meeting

Подобная встреча происходит при старте проекта. Что-то вроде teambuilding’a : все собираются вместе, начальник компании или руководитель проекта произносит речь о том, насколько новаторским и актуальным является проект, который предстоит разрабатывать, рассказывает о перспективах, ключевых точках проекта и т.д. Можно пригласить заказчиков, чтобы они рассказали что-то подобное, главное -- знакомство людей друг с другом. Это начальная точка проекта. Такое собрание поднимает командный дух и дает возможность всем друг друга узнать.

Status meetings

Позволяют регулярно получать информацию о ходе работ от разработчиков, доносить до них важную информацию или обсуждать задачи. Когда люди чувствуют себя частью общего дела, они работают с большим желанием.

Хорошей практикой является регулярное проведение статус-митингов, лучше всего -- если не каждый день, то несколько раз в неделю по заданному расписанию. Присутствие обязательное.

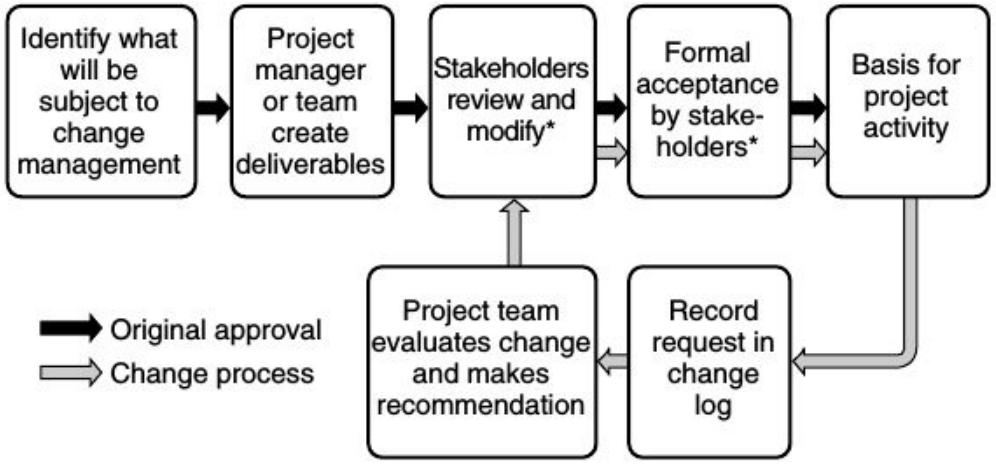
Индивидуальные встречи

А это уже встречи один-на-один с каждым членом команды. Тут беседа идёт в основном не про задачи, а про проблемы, беспокоящие человека. Хорошая практика -- проведение таких встреч раз в неделю по заданному расписанию, но с возможностью не приходить, если всё хорошо. То есть человек должен чувствовать, что у него есть возможность донести свои мысли или проблемы до руководства, если захочется. Ну и не стоит забывать про неформальное общение: вместе ходить на обед, играть в игры, есть пиццу и т.д.

Подробнее про культуру проведения совещаний поговорим на следующей лекции.

Управление изменениями

Запросы на изменения в проект могут приходить самые разные. Есть простые изменения, которые не требуют изменения ни сроков, ни бюджета, а есть серьёзные, которые могут сместить баланс проекта в ту или иную сторону. В таком случае нужно иметь утвержденную процедуру, которая позволит запустить “бюрократическую машину” одобрения и оценки. Следует заставить заказчика формально описать то, что он хочет изменить (например в письме), после чего это письмо вносится должным образом в базу данных (специализированный тул, систему контроля версий или просто страничку в вики), нумеруется, ставится дата. Эта процедура очень важна не только для планирования, но и для отслеживаемости изменений, когда потом встанет вопрос, а почему в коде реализована та или иная функциональность определённым образом.



Кроме того, мало просто зафиксировать просьбу заказчика об изменениях. Нужно сделать оценку графика, бюджета и отправить на утверждение обратно. Это своего рода “фидбэк” от проекта заказчику. После этого заказчик сможет более адекватно оценить необходимость изменений и либо согласиться с оценкой, либо продолжить уточнения и переговоры.

В случае больших и важных проектов бывает полезно даже создать что-то типа комиссии по изменениям, куда входили бы представители команды, представители заказчика, менеджер проекта и кто-то из более высокостоящих его руководителей. Это позволит распространять информацию об изменениях более широко и принимать решения об их принятии или отказе, исходя из большего количества разных точек зрения.

В данном случае бюрократия делает процесс управляемым и более прозрачным. Очень велик соблазн соглашаться на маленькие изменения, а потом вообще уйти от начального плана. Построение процесса согласно вышеописанным правилам не позволит попасть в такую ситуацию.

Завершение проекта

Завершение проекта -- ключевая точка не менее важная, чем его начало. Когда проект подходит к концу, требуется некоторое действие по коммуникации, а именно, некоторые действия, которые финализируют проект и помогают подвести итоги.

Во-первых, это официальное подтверждение от заказчика о завершении проекта, обмен всеми нужными документами, перевод денег и т.п.

Во-вторых, это прекрасная возможность чему-то научиться. Самое время проанализировать профили рисков, планы-графики, и другие подобные артефакты. После завершения проекта можно сделать общий митинг, собрать всех членов команды, обсудить итоги проекта, составить статистику. Можно устроить некоторое подобие kickoff meeting’a, и там попросить участников проекта поделиться тем, что им понравилось, а что -- нет. Это очень ценный опыт.

Аналитика по завершению проекта обязательна и в рамках всей компании в целом. Распространена практика postmortem-совещаний, когда собирается группа менеджеров других проектов и менеджер завершёного проекта делает ретроспективу,

подчёркивая самые важные моменты. В итоге происходит накопление информации и все чему-то учатся.

Если проект закончился (или заканчивается) неудачно, то всё равно стоит собрать команду и рассказать о том, как идут дела, и что это всё для них значит (даже если это увольнение). Вряд ли они окажутся рады, но хотя бы нужно оставаться честными друг с другом.

Ну и если проект подразумевает фазу сопровождения внутри компании, это самое время, чтобы собрать и привести в порядок и передать знания команде сопровождения.